**Sami Hakonen**

# MQTT-protokollan tietoturvallisuuden testaaminen

Tietotekniikan
pro gradu -tutkielma
22. maaliskuuta 2023

**Jyväskylän yliopisto**

**Informaatioteknologian tiedekunta**

**Kokkolan yliopistokeskus Chydenius**

**Tekijä:** Sami Hakonen
**Yhteystiedot:** sami.hakonen@gmail.com
**Puhelinnumero:** .
**Ohjaaja:** Risto T. Honkanen
**Työn nimi:** MQTT-protokollan tietoturvallisuuden testaaminen
**Title in English:** Testing the security of the MQTT protocol
**Työ:** Tietotekniikan pro gradu -tutkielma
**Sivumäärä:** 55
**Tiivistelmä:** Tässä tutkielmassa tarkoituksena oli selvittää älykoti ja IoT-järjestelmissä yleisesti käytetyn MQTT-protokollan tietoturvallisuuden ominaisuuksia, protokollaan kohdistettavia hyökkäyksiä ja hyökkäyksien lievennyskeinoja. Teoriaosuudessa esitellään älykotien ja IoT-järjestelmien arkkitehtuuria, yleisesti tietoturvallisuutta sen testausta ja älykotien tietoturvallisuutta. Tämän jälkeen esitetään tarkemmin MQTT-protokollan ominaisuuksia yleisesti ottaen ja tietoturvanäkökulmasta, sekä protokollaan kohdistuvia hyökkäystapoja.

Tutkielman empiirisessä vaiheessa toteutettiin erilaisia hyökkäystyyppejä protokollaa vastaan. Ensimmäisenä toteutettiin tiedonkeruu välittäjästä käyttäen Nmap-työkalua, testitapauksessa käyttäjän todennus ei ollut käytössä ja havaintona todettiin, että tämä ei tietoturvallisuuden näkökulmasta ole hyvä tapa vaan tiedon luottamuksellisuus, eheys ja saatavuus vaarantuvat. Toisena tapauksena välittäjälle asetettiin todennus ja käyttäjätunnus-salasana-pari hankittiin välittäjältä väsytyshyökkäyksellä Metasploit-työkalua käyttäen. Samaan lopputulokseen päästiin myös ilman väsytyshyökkäystä Wireshark-työkalulla välittäjän ja asiakkaan välistä liikennettä salakuuntelemalla. Viimeisenä tapauksena toteutettiin erityyppisiä palvelunestohyökkäyksiä MQTTSA-ohjelmaa käyttäen. Testissä käytetyistä palvelunestohyökkäyksistä hyötykuormaa pikkuhiljaa kasvattava hyökkäys oli tehokkain estämään järjestelmän toiminnan täysin.

Lopuksi testiympäristöön lisättiin päästä päähän TLS-salaus ja yritettiin toistaa hyökkäystapaukset. Hyökkäyksiä ei pystytty toteuttamaan, kun salaus oli käytössä. Jos hyökkääjällä olisi kuitenkin mahdollisuus vaikkapa fyysiseltä laitteelta saada salaukseen käytetty varmenne käyttöönsä, hyökkäykset olisivat mahdollisia.

Keskeisinä löydöksinä oli, ettei MQTT-protokolla sisällä kovinkaan vahvoja ominaisuuksia hyökkäyksiä vastaan vaan sitä käyttävän järjestelmän tietoturvallisuuden koventamiseksi olisi hyvä käyttää tietoliikenteen salausta.
**Avainsanat:** MQTT, tietoturvatestaus, älykoti, iot

**Abstract:** The purpose of this thesis was to investigate the security features of the MQTT protocol commonly used in smart homes and IoT systems, attacks targeting the protocol, and mitigation methods. The theoretical part introduces the architecture of smart homes and IoT systems, general cybersecurity and its testing, and the cybersecurity of smart homes. This is followed by a detailed presentation of the features of the MQTT protocol in general and from a security perspective, as well as attacks targeting the protocol. In the empirical phase of the thesis, various attack types were implemented as test cases against the protocol.

First, information was gathered from the broker using the Nmap tool, where user authentication was not enabled. It was observed that this is not a good security practice as it compromises the confidentiality, integrity, and availability of information. Second, authentication was enabled on the broker, and the username-password pair was obtained from the broker using a brute-force attack with the Metasploit tool. The same result was achieved without a brute-force attack by listening to the traffic between the broker and the client using the Wireshark tool. Finally, different types of denial-of-service attacks were implemented using the MQTTSA program. Of the denial-of-service attacks used in the test, the attack that gradually increases the payload was the most effective in completely preventing the system from functioning.

Finally, end-to-end TLS encryption was added to the test environment, and the attack scenarios were attempted to be repeated. Attacks could not be executed when TLS encryption was in use. However, if the attacker were able to obtain the certificate used for encryption, for example, from a physical device, it would be possible to execute the attacks. The main finding was that the MQTT protocol does not have very strong features against attacks, and to enhance the cybersecurity of the system using the protocol, it is advisable to use traffic encryption.
**Keywords:** MQTT, security testing, smart home, iot

# Sanasto

| | |
|---|---|
| 6LoWPAN | 6 Low Power Personal Area Network |
| ACL | Access Control List |
| AES | Advances Encryption Standard |
| Bash | Bourne again shell |
| CIA | Confidentiality, Integrity ja Availability |
| DES | Data Encryption Standard |
| DoS | Denial of Service |
| HTTP | Hyper Text Transfer Protocol |
| IEEE | Institute of Electrical and Electronics Engineers |
| IP | Internet Protocol |
| IoT | Internet of Things |
| Kbit | Kilobitti |
| LR-WPAN | Low Rate Wireless Personal Are Network |
| MAC | Medium Access Control |
| MitM | Man in the Middle |
| MQTT | Message Queuing Telemetry Transport |
| MTU | Minimisiirtoyksikkö |
| NOP | No OPeration |
| OSI | Open System Interconnection |
| OWASP | Open Web Application Security Project |
| QoS | Quality of Service |
| SQL | Structured Query Language |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| USB | Universal Serial Bus |
| XML | Extensible Markup Language |

# Sisällys

# 1 Johdanto

Tämän pro-gradu -tutkielman tutkimuksen kohteena on MQTT-protokollan tietoturvallisuus ja sen testaaminen. Lähtökohtana aiheen valintaan oli kirjoittajan kiinnostus tietoturvallisuuteen ja sen testaamiseen sekä halu syventää osaamista älykoti ja IoT-järjestelmissä yleisesti käytettyyn MQTT-protokollaan.

IoT-järjestelmässä olevat laitteet ovat monesti toiminnaltaan rajoittuneita, kuten antureita, joihin kuuluvat radiotaajuiset tunnistuslaitteet, lämpötila-anturit, kosteusanturit, liikeanturit jne. Näillä laitteilla on paljon rajoituksia, kuten rajoitettu muisti, rajoitettu laskentakapasiteetti, alhainen virrankulutus ja rajoitettu kaistanleveys toimintaympäristössään. Eri tyyppisiä kevyitä protokollia on suunniteltu erityisesti IoT-järjestelmien rajoitettua luonnetta varten. MQTT eli Message Queue Telemetry and Transport on yksi yleisimmistä sovelluskerroksen protokollista, joita IoT-järjestelmissä käytetään tiedon jakamiseen. MQTT-protokollalla ei ole vahvaa tietoturvaa tiedon jakamiseen ja jopa perustodennustiedot, kuten salasana, voidaan välittää pelkkänä tekstinä. Näistä lähtökohdista tässä tutkielmassa tutkimuskysymyksiksi nousee:

- Millaisia ovat MQTT-protokollan tietoturvallisuuden ominaisuudet?
- Millaisia hyökkäysmekanismeja MQTT-protokollaa vastaan on olemassa ja miten niitä voidaan lieventää?

Tutkielman teoreettisessa osuudessa tutustutaan ensin älykoteihin ja IoT-järjestelmiin, tämän jälkeen käsitellään teoriaan pohjautuen tietoturvallisuutta ja tietoturvatestausta sekä tietoturvallisuutta älykotien kontekstissa. Älykotien ja IoT-järjestelmien sovellustason protokollista tarkastellaan syvemmin MQTT-protokollaa, sen ominaisuuksia ja siihen kohdistuvia hyökkäystapoja.

Tutkielman empiirisessä vaiheessa toteutetaan MQTT-protokollan tietoturvatestaus testausympäristössä. Testauksessa teoreettisessa osuudessa esiin nousseita protokollan tietoturvallisuuden ominaisuuksia ja heikkouksia pyritään hyväksikäyttämään tunnettuja hyökkäysmenetelmiä käyttäen. Salaamatonta protokollaa vastaan toteutetaan tiedonkeruu, väsytys- ja palvelunestohyökkäykset. Tämän jälkeen samat testitapaukset toteutetaan TLS 1.3-salattua liikennettä vastaan.

Testauksessa voidaan havaita, että MQTT:n sisäänrakennetut tietoturvaominaisuudet eivät riitä yksinään takaamaan tiedon turvallisuutta. Testitapauksissa kaikki käytetyt hyökkäykset toimivat testiympäristössä odotetulla tavalla, kun käytössä ei ollut salausta. Välittäjästä ja sen aiheista saatiin helposti kerättyä tietoa, liikennettä voitiin salakuunnella ja sitä voitiin peukaloida. Käyttämällä salausta tämä pystyttiin estämään. Palvelunestohyökkäyksen osalta valitut hyökkäystavat estivät järjestelmän normaalin toiminnan ja pahimmillaan kaatoivat koko järjestelmän. Salauksen käyttöönotto esti testaukseen valituilla palvelunestohyökkäyksien tavoilla niiden toteutumisen. Huomioonotettavaksi asiaksi nousi kuitenkin se, että fyysiseltä laitteelta salaukseen käytetty varmenne saatetaan saada haltuun ja tämän jälkeen sitä voidaan hyödyntää hyökkäyksien toteuttamiseen.

Tämä pro-gradu tutkielma koostuu johdannon lisäksi kuudesta pääluvusta. Ensimmäisessä käsittelyluvussa 2 kerrotaan mitä älykodit ja IoT ovat ja minkälaisista komponenteista ne koostuvat sekä niiden arkkitehtuurista. Luvussa 3 avataan ylätasolla tietoturvallisuuta sen testausmenetelmiä sekä älykotien tietoturvallisuutta ja sen vaatimuksista.

Luvussa 4 syvennytään sovelluskerroksen protokollista tarkemmin MQTT-protokollaan, sen rakenteeseen, mahdollisiin käyttötapauksiin, tietoturvallisuuden ominaisuuksiin sekä yleisimpiin MQTT-protokollaan kohdistettuihin hyökkäyksiin.

Viides luku 5 sisältää tutkielman empiirisen osuuden. Luvussa kuvataan käytetty testausympäristö sekä testaus ympäristössä toteutettava MQTT-protokollan tietoturvatestaus. Luvussa 6 esitellään tietoturvatestauksen pohdinta ja viimeisessä luvussa 7 esitetään tutkielman johtopäätökset.

# 2 Älykodit ja IoT-teknologioita

Tässä luvussa esitellään yleisellä tasolla mitä älykodit ja älykotijärjestelmät ovat. Lisäksi käydään läpi erilaisia älykotijärjestelmiä ja älykotien kokonaisarkkitehtuuria eri viitekehyksien mukaisesti. Aliluvussa 2.1 esitetään yleisellä tasolla millaisista komponenteista älykotijärjestelmät rakentuvat. Aliluvussa 2.2 esitellään älykoti- ja IoT-järjestelmien arkkitehtuuria ja kuvataan esimerkkejä älykoti- ja IoT-arkkitehtuurimalleista

## 2.1 Älykotijärjestelmät

Viime vuosina kodeissa käytössä olevien verkkoon kytkettyjen laitteiden määrä on kasvanut huomattavasti. Tämä johtuu laitteiden hintojen laskusta sekä niiden paremmasta saatavuudesta. Ghirardello ym. [14] ovat esittäneet arvion, jonka mukaan vuonna 2020 kodeissa olisi käytössä yhteensä 12,8 miljardia verkkoon kytkettyä älylaitetta.

Älylaitteet, kuten Amazon Echo, Google Home ja muut keskittimet mahdollistavat kodin eri älylaitteiden helpon ohjauksen ja etäohjauksen käyttäjien puhelimien, puhetoimintojen ja erilaisten näyttöjen avulla. Nämä älylaitteet voivat ohjata ja hallita erilaisia laitteita, kuten valoja, katkaisimia, ovia, kameroita, termostaatteja ja sensoreita [48]. Myös suuremmat kodinkoneet, kuten uunit, jääkaapit ja saunan kiukaat, voivat olla yhteydessä verkkoon ja ohjattavissa etänä. Nämä älykodin laitteet voidaan yleisesti luokitella IoT-laitteiksi (Internet of Things), joita ovat laitteet, jotka ovat yhdistetty internetiin.

Älykodissa erilaiset teknologiat linkittyvät yhtenäiseksi järjestelmäksi, jolla voidaan tuottaa automaatiota kodin toimintojen hallintaan ja ohjaukseen [28]. Perinteinen tapa ohjata rakennuksia on langallisten johdotettujen laitteiden avulla. Tämä on mm. asennuskustannuksista johtuen melko kallis ratkaisu eikä kovin helposti skaalautuva. Langattomaan sensoriverkkoon pohjautuva järjestelmä on kiinteästi johdotettua edullisempi kustannuksiltaan, kun johdotuksien asennusta ei ole tarvita, tällöin myös järjestelmän skaalautuvuus on parempi.

## 2.2 Älykotien ja IoT-järjestelmien arkkitehtuurimalleista

Älykotijärjestelmä on kokonaisuus, joka koostuu useista komponenteista. Han et al. [15] kuvaavat älykotijärjestelmän kolmea pääkomponenttiryhmää, jotka ovat kotipalvelin, yhdyskäytävä ja älykodin laitteet. Kotipalvelin toimii älykodin järjestelmän keskuksena, jonka tehtävänä on tarjota palveluja, integraatiota ja tiedon jakamista järjestelmän laitteiden välillä. Kotipalvelin voi olla esimerkiksi Raspberry Pi-tietokone, jossa on erilaisia ohjelmistoja ja palveluita, kuten verkkotallennus, ohjelmoitavat liitännät ja muut älykotitekniikan tarvitsemat palvelut.

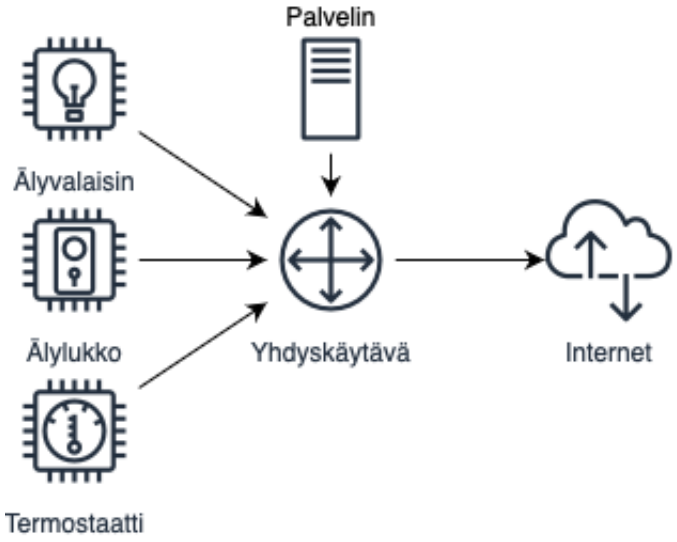
Yhdyskäytävä on toinen tärkeä komponentti älykotijärjestelmässä [15]. Yhdyskäytävä toimii tiedon välittäjänä älykodin verkon ja internetin välillä. Yhdyskäytävä mahdollistaa älykodin laitteiden etäohjauksen ja kommunikoinnin internetin palveluiden kanssa. Joissain tapauksissa kotipalvelin ja yhdyskäytävä voivat olla yhdistetty samaan fyysiseen laitteeseen.

Kolmantena pääkomponenttiryhmänä ovat älykodin laitteet, joita ovat erilaiset sensorit, valvontakamerat, älyvalot, termostaatit, lukot, hälyttimet ja muut kodin laitteet [15]. Nämä laitteet viestivät keskenään langattomasti tai langallisesti ja voivat muodostaa omia sensoriverkkojaan, joissa ne kommunikoivat keskenään ja lähettävät tietoa yhdyskäytävälle tai suoraan internetiin. Esimerkiksi Zigbee-verkon avulla laitteet voivat muodostaa verkon koordinaattorin kautta ja lähettää tietoa verkon ulkopuolelle.

IoT on laaja käsite, joka viittaa erilaisiin laitteisiin ja järjestelmiin, jotka ovat yhteydessä internetiin ja kommunikoivat keskenään. IoT:n arkkitehtuurimallia ei ole laajasti hyväksytty, vaan eri lähteissä esitellään erilaisia kerrosmalleja ja arkkitehtuureja. Yleisesti tietoverkkojen hahmottamiseen käytetty OSI-malli (Open Systems Interconnection) ei sovellu suoraan IoT-kerrosmallien kuvaamiseen, sillä IoT:n erityispiirteet ja tarpeet poikkeavat tietoverkoista.

IoT- ja OSI-kerrosmalleissa on kuitenkin joitain yhtäläisyyksiä, kuten taulukosta 2.1 voidaan huomata. Yhtäläisyyksiä voidaan hyödyntää IoT-arkkitehtuurimallin hahmottamisessa. Esimerkiksi molemmissa malleissa käytetään kerrosrakennetta, jossa kukin kerros vastaa tiettyjä toiminnallisuuksia. IoT-arkkitehtuurissa on kuitenkin huomattava, että kaikkia OSI-mallin kerroksia ei tarvita IoT:n yhteydessä.

Erilaisia IoT-arkkitehtuurimalleja on esitetty eri lähteissä, ja ne voivat sisältää 3-5 eri kerrosta. Esimerkiksi Oak et al. [37] ovat julkaisseet 4-kerroksisen IoT-arkkitehtuurimallin, kun taas Varsney et al. [49] ovat esitelleet 3-kerroksisen mallin. Näissä




Kuva 2.1: Yksinkertainen esimerkki älykodin arkkitehtuurista

malleissa kerrokset ovat yleensä samankaltaisia, mutta niiden nimet ja tarkoitukset voivat vaihdella hieman.

Taulukko 2.1: OSI- ja IoT-kerrosmallien vertailua

| **OSI-kerrosmalli** | **IoT-kerrosmalli 4 kerrosta** | **IoT-kerrosmalli 3 kerrosta** |
|---|---|---|
| 7. Sovelluskerros | Sovelluskerros | Sovelluskerros |
| 6. Esitystapakerros | | |
| 5. Istuntokerros | | |
| 4. Kuljetuskerros | Sovitus-/Palvelukerros | Verkkokerros |
| 3. Verkkokerros | Verkkokerros | |
| 2. Siirtokerros | Havaitsemiskerros | Havaitsemiskerros |
| 1. Fyysinen kerros | | |

Kolmikerroksissa mallissa IoT-arkkitehtuurin kerrokset ovat alimmaisesta ylimpään havaitsemiskerros, verkkokerros ja sovelluskerros [49]. Nelikerroksissa mallissa verkko- ja sovelluskerrosten väliin on lisätty palvelukerros [37], joka tarjoaa integraatioita ja turvallisuutta sovelluksille tai vaihtoehtoisesti nelikerroksisena esi-

tystapana on erottaa verkkokerroksesta erikseen sovituskerros, joka on tässä työssä valittu lähestymistapa.

**Havaitsemiskerros**

Havaitsemiskerros on IoT-prokollapinon alin kerros ja se vastaa OSI-mallista fyysistä ja siirtokerrosta. IEEE 802.15.4-standardissa määritellään LR-WPAN (Low Rate Wireless Personal Area Network) toiminta [37]. Se on yksi yleisimmistä fyysisellä kerroksella ja MAC (Medium Access Control)-kerroksella käytetyistä protokollista. Normaalisti se integroidaan osaksi IoT-järjestelmien havaitsemiskerrosta. LR-WPAN tarjoaa edullisen tiedonsiirron rajoitetuille hitaille ja pienitehoisille nopeuksille. IEEE 802.15.4-standardissa määritellään ainoastaan edellä mainitut kaksi kerrosta ja niiden ominaisuudet, ylempien kerroksien määrittely on jätetty muiden protokollien määritettäväksi.

Standardissa määritellään kaksi verkon peruslaitetta FFD eli Full Function Device ja RFD eli Reduced Function Device [37]. FFD:t voivat toimia verkon koordinaattoreina, kun taas RFD:t ovat yksinkertaisia sensoreita tai kytkimiä. Vasseur ja Dunkelsin kirjan mukaan [50] IEEE 802.15.14-verkon laitteilla on määrittelyn mukaisesti seuraavat ominaisuudet:

1. Pieni pakettikoko, maksimisiirtoyksikkö (MTU) eli yhdellä kertaa siirrettävissä oleva paketti on 127 tavua.
2. Tuki 16-bittisille lyhyille tai 64-bittisille MAC-osoitteille
3. Matala tiedonsiirtonopeus, sallittuja ovat nopeuden alkaen 20 Kbit/s:sta
4. Tuki tähti- ja verkkotopologioille
5. Virralta, muistilta ja suorittimelta rajoitetut laitteet
6. IEEE 802.15.4-verkot ovat yleisesti ad hoc-verkkoja, niiden sijaintia ei ole etukäteen määrätty. Lisäksi osa laitteista saattavat olla liikkeessä, esim. puetut sensorit
7. LoWPAN noodit ovat usein yhdistetty toisiin IEEE 802.15.4-linkeillä, jotka ovat epäluotettavia verrattuna langallisiin yhteyksiin.

8. Verkon noodit ovat yleisesti lepotilassa pitkiä aikoja. Laitteesta riippuen, se voi olla erilaisissa lepotiloissa, joilla on vaikutuksia sen energian kulutukseen ja heräämisnopeuteen.

### Verkkokerros

Verkkokerros vastaa datapakettiin reitittämisestä verkon laitteiden välillä. Verkkokerroksella on vastuu laitteiden välisen loogisen viestintäkanavan perustamisesta ja ylläpidosta [50]. Tyypillisesti Internet ja muut IP-verkot perustuvat IPv4:n (Internet Protocol version 4), joka käyttää 32-bittisiä osoitteita. 32-bittisen osoiteavaruuden ongelmana on sen rajoittuminen 4 294 967 296 ($2^{32}$) yksilölliseen osoitteeseen. Osoitteiden määrä on loppunut jo vuonna 2011 ja erilaisilla teknologioilla, kuten NAT (Network Address Translation) ongelmaa on pystytty kestämään.

IPv6 on kehitetty ratkaisemaan 32-bittisen osoiteavaruuden ehtyminen [50]. IPv6:ssa osoitteilla on käytössä 128 bittiä eli yli 340 sekstiljoonaa ($2^{128}$)-osoitetta. Koska IoT-verkoissa esimerkiksi älykaupunki kontekstissa saattaa olla satoja tuhansia noodeja eli runsaasti yksilöllisen tunnisteen vaativia laitteita IPv6 on ilmeinen valinta käytettäväksi verkkokerroksen protokollaksi. Myös muut ominaisuudet tukevat valintaa kuten autokonfiguraatio, jossa laitteet voivat itse valita itselleen vapaan IPv6-osoitteen, otsikkotietojen keventyminen IPv4:n verrattuna, autentikointiin ja yksityisyyteen liittyvät tiedon perusominaisuuksien suojaamiseen tähtäävät laajennukset sekä pakollinen IPSec (IP Security Architecture). IEEE 802.15.4-verkkojen ei ole kuitenkaan suunniteltu toimivan näin raskaiden protokollien kanssa ja tämän vuoksi väliin tarvitaan edellisessä aliluvussa esitelty sovituskerros eli 6LoWPAN.

### Sovituskerros

6LoWPAN eli 6 Low Power Personal Area Network on sovituskerros IEEE 802.15.4:n tarjoaman langattoman tiedonsiirron ja IPv6 eli Internet Protocol v6-pohjaisen verkkokerroksen välillä [50]. Sovituskerros on välttämätön siksi, että näiden kahden protokollan välillä on merkittävä ero siirrettävien tietojen koossa.

6LoWPANin tärkeimmät toiminnot ovat pakettien fragmentointi ja kokoaminen uudelleen sekä otsikon pakkaus [50]. Ensimmäinen toiminto liittyy siihen, että IPv6-verkkokerroksen minimisiirtoyksikkö (MTU) on 1280 tavua, kun taas alempi kerros, joka käyttää IEEE 802.15.4-standardia, voi siirtää vain 102 tavua (127 tavua - 25 tavua). Tämä tarkoittaa sitä, että pakettien koko on jaettava useisiin pienempiin osiin,

jotta niitä voidaan siirtää langattoman yhteyden yli.

Toinen tärkeä toiminto on otsikon pakkaus [50]. IPv6-protokollan otsikko on 40 tavua, ja IEEE 802.15.4-protokollan yli voidaan siirtää enintään 81 tavua kerrallaan. Kun otetaan huomioon, että ylemmän kerroksen protokolla, kuten TCP (engl. Transmission Control Protocol) tai UDP (engl. User Datagram Protocol), vaatii myös otsikkotietoja, jää sovelluskerrokselle vain hyvin pieni määrä hyötykuormaa. Esimerkiksi TCP:n otsikko on 20 tavua ja UDP:n otsikko 8 tavua, mikä tarkoittaa sitä, että sovelluskerros saa käytännössä vain muutaman tavun hyötykuormaa käyttöönsä.

Sovituskerros, kuten 6LoWPAN, mahdollistaa langattomien verkkolaitteiden kommunikoinnin tehokkaasti ja vähävirtaisesti [50]. Sovituskerros tekee tarvittavat muunnokset tietojen siirtämiseksi langattoman yhteyden yli, jotta laitteet voivat kommunikoida keskenään ilman kaapelien tai muiden fyysisten yhteyksien tarvetta. Tämä on erityisen tärkeää esimerkiksi IoT-laitteiden, kuten anturien ja toimilaitteiden, yhteydessä, joissa tarvitaan tehokasta kommunikointia ja vähävirtaisuutta.

#### Sovelluskerroksen protokollia

Sovelluskerros tuottaa käyttäjälle käyttöliittymän, johon IoT-järjestelmässä luotu tieto voidaan toimittaa. Seuraavassa esitellään tyypillisimpiä IoT-järjestelmien sovelluskerroksen protokollia, MQTT-protokollaan syvennytään tarkemmin luvussa 4.

HTTP eli Hyper Text Transfer Protocol on synkroninen pyyntö- ja vastausprotokolla, jota käytetään internetsovelluksissa, kuten verkkosivujen siirtämiseen www-palvelimelta käyttäjän laitteen selaimeen [37]. Vaikka HTTP on melko yksinkertainen tapa tiedon välittämiseen, se ei sovellu kovin hyvin IoT-järjestelmien rajoittuneiden laitteiden käyttöön, koska siinä tarvitaan lähetettävän tiedon lisäksi jonkin verran overheadia, eli protokollaan liittyviä otsikkotietoja esim. reitityksestä.

CoAP eli Constrained Application Protocol on Internet Engineering Task Force:n suunnittelema synkroninen julkaisija/tilaaja sekä pyyntö- ja vastausprotokolla [37]. Muista yleisimmistä sovelluskerroksen protokollista eroten CoAP käyttää siirtoon UDP:tä TCP:n sijaan. TCP:stä poiketen UDP:ssä ei pyydetä uudelleenlähetystä puuttuvista paketeista. CoAP:n overhead on tästä johtuen melko pieni, mutta tämä toteutuu sitten pakettien toimitusluotettavuuden kustannuksella. IoT-järjestelmissä käytetyistä protokollista CoAP on MQTT:n lisäksi käytetyimpiä.

AMQP eli Advance Message Queuing Protocol on rahoitusalalla, kuten pankkitoiminnassa yleisesti käytetty asynkroninen julkaisija/tilaajamallinen protokolla [37]. Se on itseasiassa kehitettykin sijoituspankki JPMorganin toimesta. AMQP käyt-

tää siirtoprotokollanaan TCP:tä ja keskeisenä ominaisuutena on luotettavuus ja tietoturvallisuus, jonka osalta salaus ja todennus hoidetaan TLS tai SASL eli Simple Authentication and Security Layeriä käyttäen.

XMPP eli Extensible Messaging and Presense Protocol on lähes reaaliaikaista kommunikointia, kuten chat- ja viestipalveluita varten kehitetty protokolla [37]. Siirtoprotokollana XMPP käytetään TCP:tä ja se tarjoaa sekä julkaisija/tilaajamallisen, että pyyntö-vastaajamallisen tiedon vaihdon. Muista IoT-sovellustason protokollista, poislukien HTTP, poiketen XMPP ei tue QoS:ä eli Quality of Serviceä ja tästä syystä se ei ole kovinkaan soveltuva laitteiden väliseen tiedonsiirtoon. XMPP:ssä viestin rakenne on XML-muotoista ja XML:n käyttämistä tageista syntyy overheadia viestintään. Myös XML:n jäsentäminen saattaa aiheuttaa ylimääräistä laskentatehon tarvetta, jota rajoitetun kapasiteetin laitteilla ei välttämättä ole. Tämä myös lisää laitteen virrankulutusta.

Taulukkossa 2.2 on esitetty IoT-järjestelmissä yleisimmin käytetyt sovelluskerroksen protokollat. Taulukossa voidaan nähdä MQTT:n keveys verrattuna muihin vastaavan tarkoituksen protokolliin.

Taulukko 2.2: Sovelluskerroksen protokollien vertailua Oak&Daruwala [37]

| **Protokolla** | **RESTfull Arkkitehtuuri** | **Siirto** | **Julkaisija/ Tilaaja** | **Pyyntö/ Vastaus** | **Turvallisuus** | **QoS** | **Otsikon koko tavuina** |
|---|---|---|---|---|---|---|---|
| COAP | On | UDP | On | On | DTLS | On | 4 |
| MQTT | Ei | TCP | On | Ei | SSL | On | 2 |
| XMPP | Ei | TCP | On | On | SSL | Ei | XML |
| AMQP | Ei | TCP | On | Ei | SSL | On | 8 |
| HTTP | On | TCP | Ei | On | SSL | Ei | JSON/XML |

# 3 Tietoturvallisuus ja tietoturvatestaus

Tämän luvun aliluvussa 3.1 kuvataan mitä termi tietoturvallisuus tarkoittaa sekä avataan tietoturvallisuuden peruskäsitteitä. Aliluvussa 3.2 esitellään mitä tietoturvatestaus on, sekä OWASP eli Open Web Application Security Project:n testausohjeen mukaisesti erilaisia tapoja toteuttaa tietoturvatestausta. Viimeisessä aliluvussa 3.3 käydään läpi älykotien tietoturvallisuutta ja tieturvallisuuden vaatimuksia.

## 3.1 Tietoturvallisuus

Tietoturvallisuus määritellään eri lähteissä eri tavoilla. Yleensä se kuitenkin viittaa erityisesti tietojen, tietojärjestelmien ja tietojenverkkojen suojaamiseen luvattomalta käsiksipääsyltä, käytöltä, paljastamiselta, häiriöltä, muuttamiselta tai tuhoamiselta. Tämä edellyttää erilaisten turvatoimien ja käytäntöjen toteuttamista tietojen luottamuksellisuuden, eheyden ja saatavuuden varmistamiseksi sekä tietomurtojen, kyberhyökkäysten ja muiden turvallisuusuhkien estämiseksi [42]. Tällaisia ovat esimerkiksi tiedon salaaminen, tietojen varmuuskopiointi, tietokoneiden ja verkkojen suojaamisen, tietoturvapolitiikkojen hallinta ja käyttäjien koulutus. Edellä mainitut luottamuksellisuus, eheys ja saatavuus ovat tiedon pääominaisuuksia muodostavat ns. CIA-triadin (engl. Confidentiality, Integrity, and Availability). CIA-triadin alkuperästä ei ole tarkkaa käsitystä, mutta siitä huolimatta se on vakiintunut tapa jaotella tietoturvallisuutta ja tiedon ominaisuuksia.

1. Luottamuksellisuus tarkoittaa sitä, että tietoihin pääsee käsiksi ainoastaan ne joilla on siihen oikeus.

2. Eheys tarkoittaa, että tieto ei muutu muulloin kuin silloin, kun sitä tarkoitus muuttaa. Eheyden varmistamisessa on keskeistä, että tehdyt muutokset pystytään jäljittämään.

3. Saatavuus tietoturvallisuudessa tarkoittaa, että käyttäjä, jolla on oikeus tietoon pääsee siihen myös käsiksi.

Tietoturva ja kyberturva ovat kaksi erillistä käsitettä, jotka liittyvät toisiinsa mutta keskittyvät eri asioihin [12]. Termejä käytetään arkipuheessa synonyymeina toisil-

leen, mutta niiden sisällössä on kuitenkin eroja. Tietoturva on käsitteenä laajempi ja keskittyy suojaamaan tietojärjestelmiä ja tietoja niiden sisällä. Se kattaa kaiken tiedon, joka liittyy tietojärjestelmän suojaamiseen, kuten tiedonsiirron, tietojen tallennuksen, tietojen käsittelyn ja tietojen arkiston hallinnan myös digitaalisen maailman ulkopuolella.

Kyberturvan voidaan ajatella olevan yksi osa tietoturvaa [12]. Kyberturva kattaa kaiken, mikä liittyy verkkoihin ja internetiin [42]. Se keskittyy suojaamaan verkkoja ja tietoverkkoja haittaohjelmilta, verkkohyökkäyksiltä ja muilta kyberuhilta. Kyberturva voi sisältää palomuuri- ja tunkeutumisen havaitsemisjärjestelmien hallinnan, verkkosuojauksen, verkon valvonnan ja kyberuhkien hallinnan. Yksinkertaistuksena voidaan sanoa, että tietoturva keskittyy suojaamaan tietojärjestelmiä ja tietoja niiden sisällä, kun taas kyberturva keskittyy suojaamaan verkkoja ja tietoverkkoja.

## 3.2 Tietoturvatestaus

Tietoturvatestauksessa kohdejärjestelmää tai sovellusta tarkastellaan järjestelmällisesti, jotta sieltä löydetään mahdollisia heikkouksia, tietoturva-aukkoja ja haavoittuvuuksia. Tavoitteena on löytää ja korjata ne ennen kuin niitä päästään hyväksikäyttämään. Tässä aliluvussa käydään läpi OWASP:n Testing guide v4:n mukaisesti ylätasolla tietoturvatestauksen osa-alueita. Tässä esitetyt testaustavat eivät ole kaiken kattava listaus vaan näiden lisäksi on myös muita testaustapoja.

#### Manuaalinen tarkastelu

Manuaalinen tarkastelu on ihmisen tekemään testausta, jossa kohteena on ihmisten, prosessien ja käytäntöjen turvallisuusvaikutukset [39]. Testauksessa voidaan myös tarkastella teknologioihin liittyviä päätöksiä, kuten arkkitehtuurisuunnitelmia. Manuaalisessa tarkastelussa testaus suoritetaan yleensä dokumentaatiota analysoimalla tai haastattelemalla järjestelmän suunnittelijoita tai omistajia.

Manuaalinen tarkastelu saattaa vaikuttaa yksinkertaiselta toimenpiteeltä, mutta se on kuitenkin paikassaan erittäin tehokas tekniikka. Kysymällä joltakin, miten jokin toimii ja miksi se toteutettiin tietyllä tavalla, testaaja voi nopeasti määrittää, ovatko turvallisuusongelmat ilmeisiä. Manuaaliset tarkastukset ja tarkistukset ovat yksi harvoista tavoista testata itse ohjelmistokehityksen elinkaariprosessia ja varmistaa, että käytössä on asianmukaiset tietoturvapolitiikat sekä taito. Manuaalista

tarkastelua tehtäessä on OWASP:n [39] mukaan suositeltavaa käyttää luota-mutta-varmista-mallia, sillä kaikki mitä testaajalle kerrotaan tai esitellään ei välttämättä ole paikkaansa pitävää.

Manuaalisen tarkastuksen soveltuvuus on erityisen hyvä silloin, kun halutaan testata ovatko ihmiset ymmärtäneet tietoturvaprosessin, ovatko he tietoisia käytännöistä ja onko heillä asianmukaiset taidot turvalliseen työskentelyyn. Muut toiminnot, mukaan lukien asiakirjojen manuaalinen tarkistaminen, suojatut koodauskäytännöt, turvallisuusvaatimukset ja arkkitehtoniset suunnitelmat, tulisi kaikki suorittaa manuaalisilla tarkastuksilla.

#### Uhkamallinnus

Uhkamallinnus on systemaattinen prosessi, jossa pyritään löytämään kohteeseen liittyviä uhkia, arvioidaan niiden suuruutta, merkitystä sekä mahdollisia lievennys eli mitigointikeinoja. Tietojärjestelmien uhkamallinnusta voidaan ajatella järjestelmään tehtävänä riskinarviointina [39]. Uhkamallinnus mahdollistaa järjestelmän ylläpitäjälle tai kehittäjälle resurssien kohdentamisen sellaisten mitigointikeinojen kehittämiseen, joiden vaikuttavuus on kokonaisuuden kannalta tehokkainta.

Tietojärjestelmien uhkamallinnukseen on olemassa useita eri lähestymismalleja kuten Microsoftin kehittämä STRIDE eli Spoofing, Tampering, Repudiation, Infomation disclosure, Denial of service ja Elevation of privilege, jossa on tunnistettu yleisimmät tietojärjestelmiin kohdistuvat uhkat [33]. Mallinnusta voi lähestyä myös dynaamisesta hyökkääjänäkökulmasta kuten P.A.S.T.A. eli The Process for Attack Simulation and Threat Analysis-mallissa [51] sekä puolustuksen ja riskienhallinnan näkökulmasta, kuten avoimen lähdekoodin Trike-mallissa [41]. Mallin valinta määrittää sen, mistä näkökulmasta uhkamallinnusta tehdään.

#### Penetraatiotestaus

Penetraatiotestaus on yksi tietoturvatestausmenetelmistä, sen avulla voidaan löytää hyväksikäytettäviä tietoturva-aukkoja järjestelmistä ja laitteista [39]. Penetraatiotestaus on ollut yleinen verkkojen turvallisuuden testaamistekniikka jo vuosia. Siitä käytetään myös nimityksiä black box -testaus tai eettinen hakkerointi.

Penetraatiotestauksessa pyritään löytämään tietoturva-aukkoja ennalta sovitusta järjestelmästä tai käynnissä olevasta sovelluksesta [39]. Tyypillisesti testaajilla on pääsy käyttäjätason oikeuksilla testin kohteeseen. Testaaja toimii testauksessa, ku-

ten hyökkääjä ja pyrkii löytämään ja hyödyntämään kohteesta haavoittuvuuksia.

Järjestelmän heikkoja kohtia hyödynnetään penetraatiotestauksessa luvallisen simuloidun hyökkäyksen avulla [53]. Kun haavoittuvuus on tunnistettu, sitä hyväksikäyttäen yritetään päästä käsiksi esimerkiksi arkaluontoisiin tietoihin tai saada haltuun pääkäyttötunnuksia. Penetraatiotestaus kertoo, ovatko järjestelmässä käytetyt suojatoimenpiteet riittävän vahvoja estämään mahdolliset tietoturvaloukkaukset. Testauksen jälkeen testaaja raportoi ja ehdottaa vastatoimia, joilla voidaan lieventää järjestelmän hakkeroinnin riskiä löydettyjä heikkouksia käyttämällä. Usein penetraatiotestausta tehdään erilaisilla automatisoiduilla työkaluilla. Automatisoinnin tarkoituksena on vähentää testaukseen tarvittavaa aika- ja henkilöresurssia.

## 3.3 Älykotien tietoturvallisuus ja sen vaatimukset

IoT-älykotilaitteiden suuresta kysynnästä johtuen valmistajat julkaisevat yleensä nopeasti uusia laitteita, jotka keskittyvät uusiin toimintoihin riittävän tietoturvatestauksen ja turvallisiksi suunnittelun (secure by design) sijaan. Tämä saattaa johtaa tietoturvaongelmiin [35]. Älykotilaitteiden suunnitteluun ja kokoamiseen liittyvien laajasti hyväksyttyjen ja käyttöönotettujen ohjeiden puuttuminen on johtanut tilanteeseen, jossa laitteet noudattavat lukemattomia eri standardeja [14]. Älykotijärjestelmien kerroksellisuus lisää niiden turvallisuuden kompleksisuutta, kun mukana on niin laitteita, ohjaimia, pilvi- ja mobiiliohjelmia [35]. Tämä saattaa johtaa älykodissa tilanteeseen, jossa eri laitevalmistajien laitteiden vuorovaikuttaessa toistensa kanssa sekä laitevalmistajan pilvipalvelun kanssa, muodostavat sellaisen verkon, missä käyttäjien turvallisuutta ja yksityisyyttä voi olla mahdotonta taata [14]. Myös jonkin järjestelmän osan heikko toteutus saattaa luoda mahdollisuuden hyökätä laitteeseen ja päästä sen kautta sisälle järjestelmään [35].

Yllä mainittujen älykotijärjestelmien turvallisuuden riskien vuoksi hyökkääjällä saattaa olla houkutus päästä älykotiin joko saadakseen haltuunsa tietoja asukkaasta tai tunkeutuakseen sisälle. Riittävällä osaamisella ja työkaluilla hyökkääjä voi hallita tai poistaa käytöstä kodin älylaitteita ja näin saada pääsyn taloon, kun älylukot ovatkin auki tai hälytysjärjestelmä ei laukea tai ei lähetä ilmoituksia vaikkapa pilven kautta omistajalle [35].

Päästessään käsiksi vaikkapa laitteiden lokitietoihin, hyökkääjä voi saada haltuunsa asukkaan luottamuksellisia tietoja tai tietoa tämän päivittäisistä aikatauluista, joita voidaan sitten hyväksikäyttää taloudellista hyötyä tavoitellen, esimerkiksi

asuntomurron aikataulun suunnitteluun [35]. Hyökkääjä voi käyttää haavoittuvaisia laitteita myös asukkaan häiriköimiseen ja kiusantekoon. Pahimmillaan hyökkääjä saattaa aiheuttaa vahinkoa kiinteistölle tai henkilöille esimerkiksi aiheuttamalla tulipalon laittamalla etäohjattavan kiukaan päälle, kun saunatilassa on pyykkiä kuivumassa.

Älykodin tietoturvallisuuden saaminen riittävälle turvallisuuden tasolle jää monen valmistajan laitteita sisältävässä järjestelmässä käyttäjän harteille. Käyttäjän vastuulla on laitteiden ja sovellusten päivityksistä huolehtiminen, liiallisten ja turhien oikeuksien rajoittaminen, sekä järjestelmän suojaaminen esimerkiksi palomuurilla kotiverkon ulkopuoliselta käytöltä. Riittävän kompleksisella salasanakäytännöillä ja oletuskirjautumistunnuksien vaihtamisella voidaan myös pienentää riskiä laitteiden päätymisestä väärinkäytön kohteeksi.

### 3.3.1 Älykotien tietoturvallisuuden vaatimuksia

Älykotien palveluiden tietoturvavaatimukset on Han et al. [15] julkaisussa jaettu tietoturvallisuudessa yleisesti käytetyn CIA-mallin mukaisesti. Seuraavissa kappaleissa esitetään lähteenmukaisesti tieturvallisuusvaatimukset vapaasti käännettynä ja sovitettuna.

Luottamuksellisuuden varmistamiseksi älykodin laitteiden välisen viestinnän aikana käyttäjän yksityisyystietoja ja salausalgoritmin keskeisiä tietoja tulee hallita turvallisesti niin, etteivät ulkopuoliset pääse niihin käsiksi [15]. Älykodin laitteissa luotua ja niiden välillä liikkuvaa tietoa ei tule lähettää selkokielisenä. Älykodin laitteen tunnistetietoja pitää hallita turvallisesti, jotta ulkopuolinen ei pysty replikoimaan tai muokkaamaan niitä. Älykodin laitteiden tulisi tarjota erittäin turvallinen tapa salasanan asettamiseksi ja sen säännöllinen vaihtamistoiminne. Yhdyskäytävän pitää koventaa turvallisuutta riittävän vahvan ja monimutkaisen salasanavaatimuksen avulla.

Eheyden varmistamiseksi älykodin laitteiden luotettavuuden ja turvallisuuden ylläpitämiseksi valtuuttamattomia laitteita tai käyttäjiä ei tule sallia [15]. Älykodin laitteiden välisen viestinnän aikana käyttäjän yksityisyystietoja ja salausalgoritmin keskeisiä tietoja ei saa päästä väärentämään tai peukaloimaan. Älykodin laitteelta tuotettua tietoa ulos tai toiseen laitteeseen lähetettäessä tiedon eheys tulee varmistaa. Älykodin palveluita tarjoavien laitteiden on muodostettava luotettava viestintäympäristö keskinäistä autentikointia hyödyntäen.

Saatavuuden varmistamiseksi tulee järjestelmässä olla ulkoisten hyökkäysten

havaitsemisominaisuudet, jotta turvallisuusuhkiin, kuten kyberhyökkäyksiin ja hakkerointiin voidaan vastata [15]. Älykodin laitteissa tulee olla ominaisuudet, jotka mahdollistavat sen ohjelmiston päivitykset. Laitteen ominaispiirteet ja tekniset tiedot on otettava huomioon, kun sen suojauskäytäntöjä mietitään. Laitteiden fyysisen tilatiedon ylläpitämiseksi tulisi olla laitehallintajärjestelmä. Laitteiden tilatiedon seuranta ja tarpeettoman etäkäytön estäminen tulee mahdollistaa. Jos älykodin laitteista havaitaan epänormaalia toimintaa, tulisi järjestelmän luoda tästä historiatietoa.

### 3.3.2 IoT-järjestelmien tietoturvallisuuden uhkia

Tietoturvallisuudessa uhkia voidaan jaotella useilla eri tavoilla. Dorsemaine et. al. [10] esittävät seminaarijulkaisussaan ylätason jaottelun IoT-järjestelmien tietoturvallisuuden uhkille ja mahdollisille hyökkäyksille. Heidän julkaisussaan IoT-arkkitehtuuri on jaoteltu 4-kerroksisella mallilla sen mukaisesti mihin uhat kohdistuvat. Ensimmäisenä kerroksena on paikallinen ympäristö, jossa sijaitsevat verkon sensorit ja laitteet, toisena kerroksena on siirtokerros taulukossa 2.1 esitettynä nämä sijoittuvat havaitsemiskerrokseen. Kolmas kerros on tiedon varastointi ja louhintakerros ja neljäntenä kerroksena on saatavuuskerros, jotka taulukossa 2.1 sijoittuvat sovelluskerrokseen.

Koska IoT-laitteet ovat fyysisesti käyttöympäristössä, niihin saatetaan päästä helpostikin käsiksi [10]. Hyökkääjä voi esimerkiksi yrittää käyttää laitteen erilaisia portteja, kuten USB ja ethernet tai helposti vaihdettavia massamuistilaitteita kuten SD-kortit saadakseen laitteen haltuunsa tai päästäkseen sen kautta kiinni verkossa liikkuvaan tietoon. Kun laitteeseen päästään fyysisesti käsiksi, on mahdollista myös saada pääsy ja ns. peukaloida kyseisen laitteen laiteohjelmistoa (engl. Firmware). Tämä antaa hyökkääjälle mahdollisuuden käyttää kyseiseen laiteohjelmistoon mahdollisesti liittyviä haavoittuvuuksia, kuten kovakoodattuja salasanoja tai takaovia [7].

Tiedonsiirtokerros mahdollistaa IoT-laitteiden tiedonsiirron erilaisiin varastointi ja tiedon louhintaan liittyviin palveluihin. Tähän kerrokseen liittyvissä uhissa ja hyökkäyksissä käytetään hyväksi eri tiedonsiirtovälineisiin liittyviä heikkouksia ja haavoittuvuuksia. Tällaisia ovat esimerkiksi tekeytyminen langattoman verkon tukiasemaksi (engl. Rogue access point), jolloin verkon laitteet saattavat käyttää verkon oikean tukiaseman sijaan huijarin tukiasemaa tietoliikenteeseensä [45]. Tämä mahdollistaa laitteiden välisen suojaamattoman liikenteen salakuuntelun.

Tiedonsiirtoon kohdistuu myös MAC-osoitteiden väärentäminen, jolloin jokin laite käyttää verkkoon jo kuuluvan laitteen MAC-osoitetta esittääkseen tätä ja päästäkseen sisään verkkoon. Päästyään sisälle verkkoon, laite voi laukaista esimerkiksi Man in the middle -hyökkäyksen, eli tekeytyä verkon laitteeksi, jolloin se voi salakuunnella verkon liikennettä, lähettää verkkoon virheellistä tai muuteltua tietoa tai aiheuttaa palvelunestohyökkäyksen [29].

Hyökkäykset ja uhat tiedon varastointia ja louhintaa kohtaan rajautuvat tämän tutkielman ulkopuolelle, koska niissä hyökkäysvektoreina toimivat erilaiset virtuaalipalvelimet ja -koneet, mukaan lukien pilvipalvelut [10]. On hyvä kuitenkin huomioida, että niiden kautta voidaan päästä käsiksi IoT-verkon laitteisiin ja tietoon ja toisaalta IoT-verkon laitteiden kautta on mahdollista vaikuttaa myös toiseen varastointi ja louhintapalveluiden suuntaan.

Myös rajapinta- ja käyttöliittymävektoreihin liittyvät uhat ja hyökkäykset rajautuvat tämän tutkielman ulkopuolelle. Näihin liittyviä uhkia ja hyökkäyksiä ovat esimerkiksi erilaiset koodi-injektiot [10]. Näissä syötteitä vastaanottaviin järjestelmiin syötetään erilaisia merkkijonoja, joilla pyritään saamaan se toimimaan vastoin sen tarkoitettua toimintaa. Tällaisia ovat esimerkiksi erilaiset SQL-injektiot, Cross Site Request Forgery sekä järjestelmien virheellisten asetuksien hyväksikäyttö.

# 4 MQTT-protokolla

Tässä luvussa esitellään MQTT-protokolla, mihin sitä käytetään, sen tietoturvallisuuden ominaisuuksia ja tunnistettuja tietoturvallisuuden heikkouksia. Ensimmäisessä aliluvussa 4.1 esitellään MQTT-protokollan ominaisuuksia. Aliluvussa 4.2 syvennytään MQTT-paketin rakenteeseen ja aliluvussa 4.3 käydään läpi MQTT-protokollan tietoturvallisuuden ominaisuuksia. Tämän luvun viimeisessä aliluvussa 4.4 esitellään eri lähteisiin perustuen hyökkäyksiä MQTT-protokollaa vastaan.

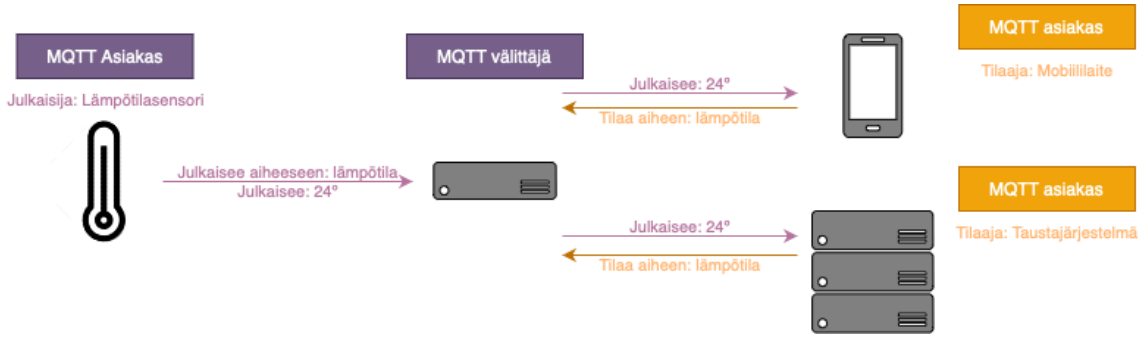
## 4.1 MQTT:n ominaisuuksia

MQTT eli Message Queue Telemetry and Transport-protokolla on yksi yleisimmistä IoT-järjestelmissä käytetyistä sovelluskerroksen protokollista [37]. Tyypillisimpiä käyttötapauksia MQTT-protokollalle on erilaiset kotiautomaatiojärjestelmän toiminnot, kuten valojen ohjaus, virrankulutuksen seuranta, erilaiset lääkintälaitteet, älykodit ja mobiiliohjelmat [37]. Se on asynkroninen julkaisija/tilaaja -mallinen protokolla, jonka IBM julkaisi vuonna 1999 kevyeksi ratkaisuksi laitteiden väliseksi (machine to machine, M2M) viestimisprotokollaksi.

Keskeisenä suunnitteluperusteena MQTT-protokollalle ovat olleet vaatimukset pienelle kaistan tarpeelle ja vähäiselle virrankulutukselle. Protokollapinossa MQTT asettuu TCP:n päälle, eli OSI-mallissa tasoille 5-7. Sitä käytetään kahden laitteen väliseen tiedon jakamiseen. Protokollan suunnittelussa keskeisimpänä ohjaavana tekijänä on ollut protokollan keveys. Tämän vuoksi siinä ei ole juurikaan sisäänrakennettuja turvallisuusominaisuuksia [35].

MQTT-protokollan julkaisija/tilaajamalli rakentuu asiakkaasta (engl. Client), välittäjästä (engl. Broker) sekä aiheesta (engl. Topic), nämä esitellään lyhyesti seuraavissa kappaleissa. Osille ei ole vakiintuneita nimikkeitä suomen kielellä, tässä tutkielmassa käytetään edellä esitettyjä. Kuvassa 4.1 on esitetty MQTT:n julkaisija/tilaaja-arkkitehtuurista havainnekuva.

Aiheet määritellään UTF8-tyyppisillä merkkijonoilla ja ne on järjestetty hierarkisesti [40]. Kunkin tason erottaa vinoviiva (/) samalla tavalla kuin verkkopoluissa, tasojen käyttö ei kuitenkaan ole pakollista. Esimerkiksi aihe voisi olla vaikka




Kuva 4.1: MQTT julkaisija/tilaaja-arkkitehtuuri [34]

"`/home/room1/light-sensor`" tai yhtälailla pelkkä "`sensor`". Aiheessa voidaan käyttää kahta erilaista jokerimerkkiä, `+` ja `#`. Ne vastaavat yksittäisiä (+) ja useita (#) tasoja hierarkiassa. Jokerimerkkejä voidaan myös väärinkäyttää, jos asiakas voi tilata #-aiheen, se pystyy esimerkiksi vastaanottamaan kaikki toisilta asiakkailta lähetetyt viestit. Välittäjä käyttää aiheiden määrityksiä viestien suodattamiseen.

Asiakas tarkoittaa MQTT:ssä verkkoon liittyneitä laitteita, jotka lähettävät ja vastaanottavat tietoa. Tällaisia ovat esimerkiksi sensorit, mobiililaitteet ja tietokoneet [54]. Asiakas voi olla julkaisija tai tilaaja tai sitten molempia. Tilaajat ja julkaisijat eivät kuitenkaan ole suoraan toisiinsa yhteydessä vaan väliin tarvitaan aina välittäjä.

Välittäjä on asiakkaan lisäksi MQTT:n oleellisin osa. Välittäjä toimittaa vastaanottamiaan viestejä eteenpäin niille asiakkaille, jotka ovat kyseiseen aiheeseen liittyneet viestit tilanneet [38]. Välittäjä on käytännössä ohjelma, jota suoritetaan jollain tehtävään sopivalla tietokoneella, kuten palvelimella tai kevyellä yhden piirilevyn tietokoneella.

Välittäjän pääasiallisia tehtäviä ovat asiakkaiden hyväksyminen mukaan tiedonsiirtoon, asiakkaiden lähettämien sovellusviestien vastaanotto, asiakkaiden lähettämien tilaus- ja tilauksen lopettamispyyntöjen käsittely sekä sovellusviestien välittäminen aiheen tilanneille asiakkaille. Hyväksyessään asiakkaan mukaan tiedonsiirtoon välittäjä voi todentaa asiakkaan käyttäjänimen ja salasanan, varmenteen tai symmetrisen salausavaimen avulla [40].

## 4.2 MQTT-paketti

MQTT-protokolla toimii viestipakettien kaksisuuntaisella välityksellä, mikä toteutetaan ennalta määritellyllä tavalla [38]. Viestipakettien vähimmäiskoko on kaksi tavua ja ne koostuvat kolmesta osasta, jotka ovat aina samassa järjestyksessä.

Ensimmäisenä osana viestipakettia on kiinteä otsikko (engl. Fixed header), tämä on aina mukana kaikissa viestipaketeissa [38]. Kiinteän otsikon pituus on aina vähintään kaksi tavua, sen rakenne on kuvattuna taulukossa 4.1. Viestipakettien ohjaus tapahtuu paketin ensimmäisen tavun neljän ensimmäisen bitin toimesta. Paketin tyypille on 15 mahdollista arvoa 0-15, koska arvo 0 ei ole käytössä. Mahdolliset pakettityypit on esitetty taulukossa 4.2. Ensimmäisen tavun jäljelle jäävät neljä bittiä sisältävät mahdolliset liput. Yhdessä nämä kaksi kenttää muodostavat viestipaketin ohjauskentän (engl. Control field). Viestipaketin kiinteän otsikon toinen tavu kertoo kokonaislukuna sen, kuinka monta bittiä paketin pituus on.

Taulukko 4.1: Kiinteän otsikon rakenne [38]

| **Bitti** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| tavu 1 | Paketin tyyppi (Type) | | | | Pakettityypin liput (Flags) | | | |
| tavu 2 | Paketin pituus (Remaining length) | | | | | | | |

Toisena osana MQTT-viestipaketissa tulee mahdollinen vaihtuva otsikko (engl. Variable header) [38]. Vaihtuva otsikko sisältää yleensä pakettitunnisteen (engl. Packet identifier). Vaihtuvan otsikon sisältö riippuu paketin tyypistä.

Kolmantena viestipaketin osana on mahdollinen hyötykuorma (engl. Payload) [38]. MQTT:ssa paketit jakaantuvat hyötykuormallisiin eli paketteihin, joiden sisältöön asiakas voi vaikuttaa ja hyötykuormattomiin eli paketteihin, joiden sisältö on ennalta määritelty. Pakettityypeistä CONNECT, PUBLISH, SUBSCRIBE, SUBACK, UNSUBSCRIBE ja UNSUBACK ovat hyötykuormallisia ja loput sitten hyötykuormattomia. Hyötykuorman sisältö riippuu paketin tyypistä ja esimerkiksi CONNECT-paketissa ovat mahdollisina kenttinä User Name ja Password, joita voidaan käyttää palvelin päässä eli välittäjällä asiakkaan todentamiseen ja valtuutukseen.

Kuvassa 4.2 on esitetty MQTT-pakettien liikenne [38]. Ensimmäisessä vaiheessa julkaisija tai tilaaja lähettää välittäjälle CONNECT-paketin. CONNECT-paketti sisältää tiedot yhteyttä ottavasta asiakkaasta. Pakollisia kenttiä CONNECT-paketissa

Taulukko 4.2: MQTT:n pakettien mahdolliset tyypit [38]

| **Paketin nimi** | **Arvo** | **Suunta** | **Kuvaus** |
|---|---|---|---|
| Reserved | 0 | Kielletty | Varattu myöhempään käyttöön |
| CONNECT | 1 | Välittäjälle | Yhteyspyyntö |
| CONNACK | 2 | Asiakkaalle | Yhteyden hyväksyminen |
| PUBLISH | 3 | Molemmat | Viestin julkaisu |
| PUBACK | 4 | Molemmat | Julkaisun kuittaus (QoS 1) |
| PUBREC | 5 | Molemmat | Julkaisu vastaanotettu (QoS 2 1/3) |
| PUBREL | 6 | Molemmat | Julkaisun julkaisu (QoS 2 2/3) |
| PUBCOMP | 7 | Molemmat | Julkaisu valmis (QoS 2 3/3) |
| SUBSCRIBE | 8 | Välittäjälle | Julkaisun tilaaminen |
| SUBACK | 9 | Asiakkaalle | Julkaisun tilaamisen kuittaus |
| UNSUBSCRIBE | 9 | Välittäjälle | Tilauksen lopettaminen |
| UNSUBACK | 11 | Asiakkaalle | Tilauksen lopetuskuittaus |
| PINGREQ | 12 | Välittäjälle | PING-kysely |
| PINGRESP | 13 | Asiaakkaalle | PING-vastaus |
| DISCONNECT | 14 | Molemmat | Yhteyden katkaiseminen |
| AUTH | 15 | Molemmat | Varmenteiden vaihto |

ovat clientID ja cleanSession. Näistä clientID on jokaisen asiakkaan yksilöllinen tunniste merkkijonona. Jos tunniste on jollain verkon laitteella jo käytössä, välittäjä katkaisee yhteysyrityksen. Toisena pakollisena tietona on cleanSession, siinä asiakas kertoo välittäjälle sen, onko kyseessä uusi istunto (engl. session) vai kokonaan uusi.

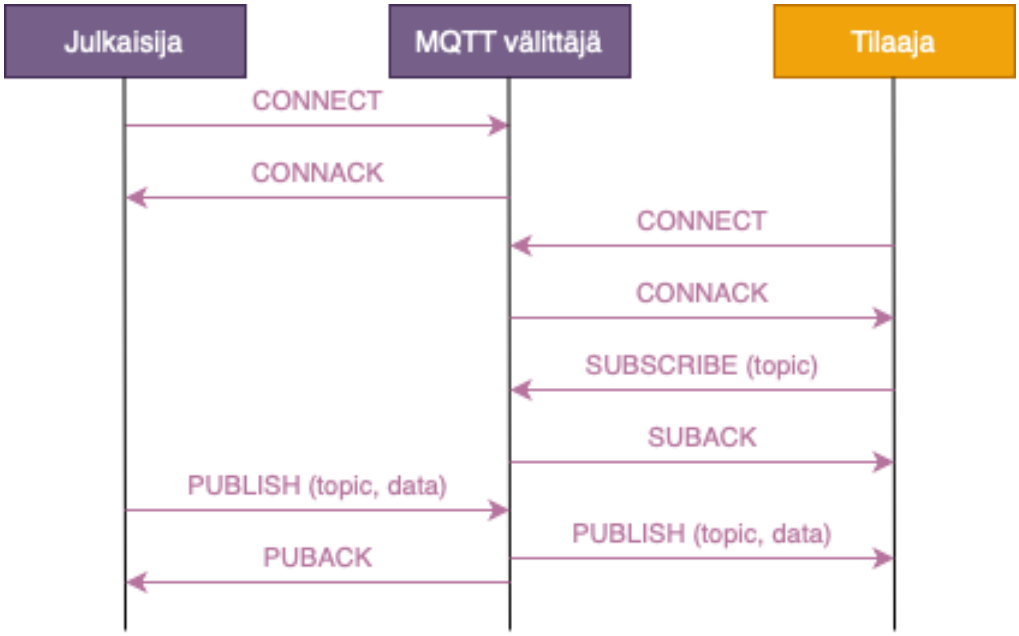
Yhteysyrityksessä CONNECT-paketin vastaanotettuaan välittäjä lähettää asiakkaalle CONNACK-paketin [38]. CONNACK-paketissa on mukana vähintään kaksi muuttujaa. SessionPresent joka on totuusarvo TRUE tai FALSE, ja joka kertoo asiakkaalle, että aiempi istunto löytyy välittäjältä ja sen voi palauttaa, sekä returnCode, joka on numeroarvo, jossa välittäjä kertoo onko yhteys hyväksytty vai hylätty, sekä hylkäämisen syyn. Näiden lisäksi mukana voi olla muita parametreja, jotka kertovat esimerkiksi mahdollisista rajoituksista pakettien kokoon tai QoS eli Quality of Service:n liittyen.

Yhteyden muodostamisen jälkeen julkaisija voi lähettää PUBLISH-paketin [38]. Siinä on mukana kuusi mahdollista muuttujaa: packetID, joka on yksilöllinen jokaiselle komennolle, aihe eli topicName, hyötykuorma, QoS, joka asettaa laatutason, retainFlag, joka kertoo välittäjälle halutaanko viesti jättää välittäjälle talteen sekä dupFlag joka ilmoittaa sen onko sama viesti lähetetty jo kerran. Välittäjä kuittaa tämän paketin PUBACK-paketilla, jossa on mukana kuitattava packetID sekä syykoodi, joka kertoo onko julkaisu hyväksytty vai hylätty ja hylkäyksen syyn.

Tilaaja taas voi yhteyden muodostamisen jälkeen ilmoittaa välittäjälle tilaavansa jonkin aiheen [38]. SUBSCRIBE-paketissa on mukana paketin tunniste packetID, sekä yksi tai useampi aihe ja aiheen QoS:n arvo, joita tilaaja haluaa saada välittäjältä. Välittäjä kuittaa tilauksen SUBACK-paketilla, jossa on mukana tilauksen packetID sekä palautuskoodi, jossa välittäjä kertoo hyväksyykö se tilauksen ja sen toivotun QoS:n vai hylkääkö se tilauksen ja hylkäyksen syyn.

Last Will (myös Last Will and Testament) ja QoS ovat MQTT-protokollaan liittyviä ominaisuuksia. Kuten aiemmin on mainittu QoS tarkoittaa palvelun laatutasoa [19]. Tämä ominaisuus määrittää viestin tärkeysasteen julkaisijan, välittäjän ja tilaajan välillä. Viestin julkaisija ja tilaaja voivat käyttää eri palvelun laatutasoja, jotta voidaan antaa erilaisia tärkeysasteita eri viesteille ja vastaanottajille. MQTT-protokolla tarjoaa kolme eri palvelun laatutasoa: 0, 1 ja 2, ja viestiketjun matalin laatutaso määrittää koko ketjun palvelun laadun.

Last Will ("LWT") ominaisuutta käytetään ilmoittamaan muille asiakkaille, että jokin asiakas on katkaissut yhteyden odottamattomasti [17]. Jokainen asiakas voi määrittää LWT-viestin liittyessään välittäjään CONNECT-paketin mukana. LWT-




Kuva 4.2: MQTT-pakettiliikenne [38]

viesti on tavallinen MQTT-viesti, jossa on aihe, retainFlag, QoS ja hyötykuorma. Välittäjä tallentaa viestin, kunnes se havaitsee, että asiakas on katkaissut yhteyden odottamattomasti. Kun yhteys katkeaa, välittäjä lähettää tallennetun LWT-viestin kaikille LWT-viestin aiheen tilanneille asiakkaille. Jos asiakas katkaisee yhteyden sujuvasti lähettämällä oikean DISCONNECT-viestin, välittäjä poistaa tallennetun LWT-viestin.

## 4.3 MQTT:n tietoturvallisuuden ominaisuudet

Kuten luvun alussa todettiin MQTT-protokollaa ei ole suunniteltu turvallisuuden näkökulmasta [37]. Käytännössä protokollaan sisältyvät turvallisuusominaisuudet liittyvät asiakkaiden tunnistamiseen ja todentamiseen. MQTT-protokollassa voidaan käyttää asiakkaiden todennukseen käyttäjänimeä ja salasanaa [18]. Käyttäjänimi ja salasana kulkevat CONNECT-viestissä, kun asiakas lähettää yhteyspyynnön välittäjälle. Käyttäjänimi on paketissa UTF-8 enkoodattuna merkkijonona, eli selväkielisenä. Salasana välitetään binaaridatana ja maksimissaan 65535 bitin kokoisena. Välittäjä vastaa asiakkaan lähettämään CONNECT-viestiin CONNACK-paluuviestillä koodilla ”0”, kun yhteys on hyväksytty. Käytetystä MQTT-versiosta riippuen vas-

taus, kun yhteydestä kieltäydytään väärän käyttäjänimen tai salasanan vuoksi, on MQTT v.5:ssä ”134” ja MQTT v.3.1.1 ”4”. Kun yhteydestä kieltäydytään käyttöoikeuden puuttumisen vuoksi vastauksena on ”135” ja ”5” [38]. Koska tunnukset lähetetään selkokielisenä välittäjälle on mahdollista, että salakuuntelemalla ne päätyvät salakuuntelijan tietoon. Tunnusten turvalliseen lähettämiseen vaaditaan liikenteen salaamista jollain ratkaisulla.

Käyttäjätunnus-salasana-parin lisäksi MQTT-protokollassa voidaan käyttää myös muita todennustapoja [18]. Jokaisella MQTT-asiakkaalla on yksilöllinen asiakastunnus. Asiakastodennuksessa asiakas toimittaa tämän tunnuksen CONNECT-viestissä välittäjälle. Yleinen käytäntö on käyttää 36-merkkiä pitkää ns. UUID eli Universal Unique Identifieria tai jotain muuta yksiköllistä asiakastietoa. Tämä voi olla myös laitteen verkkoliitännän MAC-osoite tai laitteen sarjanumero. Todennusprosessissa mahdollisena tapana asiakkaan todentamiseen on ensin vahvistaa käyttäjätunnus-salasanapari ja sen jälkeen asiakastunnuksen täsmääminen niiden kanssa. Voidaan myös käyttää pelkkää asiakastunnusta ilman käyttäjätunnus-salasanaparia, mutta tämä ei ole tietoturvallisuusnäkökulmasta kovinkaan hyvä tapa.

MQTT-asiakas voi tehdä käytännössä kahta asiaa sen jälkeen, kun se on saanut yhteyden välittäjään [18]. Se joko julkaisee viestejä tai tilaa aiheita. Ilman minkäänlaisia rajoituksia eli asetettuja valtuutuksia, jokainen todennettu asiakas voi julkaista ja tilata kaikkia mahdollisia aiheita. Jotta tällaiselta tilanteelta vältyttäisiin voidaan välittäjällä ottaa käyttöön käyttöoikeudet aiheisiin eli käyttöoikeuslista (engl. Access Control List, ACL) [47]. Näillä voidaan rajoittaa sallittuja aiheita, sallittuja toimintoja (julkaisu, tilaus, molemmat) ja sallittuja QoS-tasoja.

Edellisten tapojen lisäksi lisäturvallisuutta MQTT-protokollaan saadaan lisäämällä sen alapuolelle protokollapinoon salauskerros. Tämä voidaan toteuttaa TLS-salausta käyttämällä, joko asiakasvarmenteilla tai päästä-päähän salaamalla [18].

Salaustyyppejä on kaksi perustyyppiä. Symmetrisessä salauksessa käytetään yksityisiä avaimia lähettävän ja vastaanottavan pään välillä [37]. Epäsymmetrisessä salauksessa käytetään sekä julkista, että yksityistä avainta päiden välillä. Symmetrinen salaus voidaan toteuttaa kahdella eri tavalla, joka lohkosalauksella (engl. Block cipher) tai jonosalauksella (engl. Stream cipher) [23]. Lohkosalaukselle on muutamia standardeja DES eli Data Encryption Standard, AES eli Advances Encryption Standard sekä Blowfish, näistä AES on ehkä käytetyin [31]. Lohkosalauksessa selväkielinen teksti jaetaan lohkoihin, jokainen lohko salataan samalla yksityisellä avaimella. Jonosalauksessa salaus suoritetaan bitti kerrallaan, jolloin salaus on nopeampaan,

mutta turvallisuus ei ole niin varmaa.

Asiakasvarmennetta käytettäessä TLS-kättelyn yhteydessä asiakas toimittaa julkisen avaimensa sisältävän varmenteensa palvelimelle eli MQTT:ssä välittäjälle sen jälkeen, kun palvelimen varmenne on todennettu [2]. Kyseessä on siis epäsymmetrinen salaus. Asiakkaan varmenteen perusteella palvelin voi varmentaa asiakkaan identiteetin ja tarpeen mukaan katkaista kättelyn, jos varmenteen varmennus epäonnistuu.

Toisena salaustapana on liikenteen salaaminen TLS-salausta käyttäen [27]. TLS (engl. Transport Layer Security) on salausprotokolla, jota käytetään tietoliikenteen suojaamiseen salaamalla se päästä päähän. TLS-salauksen lisääminen vaatii enemmän suorituskykyä ja rajoitetuilla ominaisuuksilla varustetuille laitteilla se voi olla raskas, johtuen sekä TLS:n käyttämästä kättelymekanismista että TLS:n käyttämistä kryptografisista algoritmeista [9]. TLS-salaus perustuu TCP-kättelyssä vaihdettavaan varmennetiedon käyttöön tiedon salaamisessa. Välittäjä ja asiakas suorittavat ensin TCP-kättelyn ja tämän jälkeen TLS-kättelyn. TLS-kättelyssä asiakas kertoo mitä salausalgoritmeja (engl. Cipher Suite) se tukee ja pyrkii arvaamaan mitä avaimenvaihtoprotokollaa käytetään sekä lähettää oman avainjakonsa (engl. Key Share) palvelimelle. Palvelin vastaa asiakkaalle käytetyn avaimenvaihtoprotokollan ja oman avainjakonsa sekä varmenteensa. Tämän jälleen osapuolien välinen liikenne on salattua.

TLS-salauksen käyttö lisää overheadia tietoliikenteelle, koska jokainen paketti salataan [20]. Välittäjän päässä, jos laskentateho ei ole kovin rajoittunutta tämä ei välttämättä aiheuta ongelmia. Asiakkaina saattaa kuitenkin olla laskentateholtaan hyvin rajoittuneita laitteita ja niiden prosessoriteho ja muisti ei välttämättä riitä TLS-salauksen käyttöön.

## 4.4 Hyökkäyksiä MQTT-protokollaa vastaan

#### Tiedonkeruu välittäjästä

Yleensä hyökkäyksiä edeltää tiedonkeruu hyökkäyksen kohteesta. Tiedon kerääminen ei itsessään ole vielä hyökkäämistä, mutta siitä saatua tietoa voidaan hyödyntää, kun suunnitellaan hyökkäystä. Jussilan [24] diplomityössä tavoitteena on ollut selvittää MQTT-protokollan tietoturvaan liittyvien ominaisuuksien toimivuutta viestiliikenteessä, sekä protokollaan liittyviä tietoturvahaavoittuvuuksia. Tutki-

muskysymyksinä Jussilalla on ollut Miten palvelutaso näkyy viestiliikenteessä? Onko viestin pysyvä tallennus tietoturvauhka? Miten MQTT:ssä asiakkaan salasana ja viesti salataan? Miten viestin kohdentaminen halutulle asiakkaalle parantaa tietoturvaa? Testiasetelmassaan Jussila on mitannut viestiliikennettä eri QoS-tasoilla sekä ilman salausta että TLS-salausta käyttämällä. Jussila on työssään havainnut, että QoS-tason nosto 0:sta 1:een tai 2:een nostaa viestiliikenteen määrää. Tietoturvatestauksessaan Jussila on todennut, että kryptatulla salasanalla kirjautuminen välittäjälle ei onnistunut vaan tarvittiin selkokielinen salasana. Lisäksi havaintona oli, että välittäjä ei rajoittanut kirjautumisyritysten määrää mitenkään vaan mahdollisti rajoittamattomasti kirjautumisyrityksiä. TLS-salauksen käyttämisestä liikenteen salaamiseen Jussila on havainnut, että salattua liikennettä ei voitu tulkita verkkoliikenteestä.

Andy et al. [3] ovat konferenssijulkaisussaan tutkineet erilaisia hyökkäysskenaarioita MQTT-protokollaa vastaan sekä tehneet sille turvallisuusanalyysiä. He ovat hakeneet internettiin auki olevia MQTT-välittäjiä Shodan-hakukoneella. MQTT käyttää oletusporttina porttia 1883 ja TLS-salausta käytettäessä porttia 8883. Näiden porttien auki olo kohdejärjestelmässä paljastaa skannaajalle, että palvelimella mahdollisesti on käynnissä MQTT-välittäjä. Skannauksessa löydetyn palvelimen palveluiden sormenjälkiä (engl. Fingerprinting) tutkimalla, saadaan helposti selville käytetyn MQTT-ohjelmiston versio ja nimi. Testiasetelmassa anonyymit asiakkaat sallivalta välittäjältä voidaan tilata kaikki aiheet ja toisaalta välittäjälle voidaan julkaista peukaloitua tietoa. Tämä myös mahdollistaisi palvelunestohyökkäyksen välittäjää ja tilaajia vastaan. Kun tiedetään, millainen ja millä versiolla oleva järjestelmä on käytössä, voidaan etsiä esimerkiksi tietyn sovelluksen tiettyyn versionumeroon liittyviä haavoittuvuuksia ja hyväksikäyttää niitä. Testauksessa havaitaan mm., että salaamatonta liikennettä voidaan seurata Wireshark-ohjelmalla. Lisäksi havaitaan, että tietoliikennettä seuraamalla saadaan selville salaamattomasta liikenteestä legitiimin asiakkaan kirjautumistiedot. Andy et al. toteavat johtopäätöksenään, että MQTT-protokollan kanssa tulisi käyttää turvallisuutta parantavaa mekanismia, kuten TLS-salausta.

Myös Harsha et al. [16] ovat analysoineet MQTT:n haavoittuvuuksia käyttäen Shodan-hakukonetta. He ovat havainneet aiempien tutkimuksien tapaan, että ilman asiakkaiden tunnistamista viestiliikennettä voidaan seurata ja siihen voidaan vaikuttaa ulkopuolelta. Tunnistautumismekanismeina he esittävät käyttätunnus-salasana-paria tai asiakasvarmennetta. Kuten aiemmissakin tutkimuksissa havaintona

on, että salaamattomassa liikenteessä käyttäjätunnus ja salasana saadaan selkokielisenä kaapattua viestiliikenteestä. Mitigointikeinona he esittävät TLS:n käyttöä tai viestiliikenteen hyötykuorman kryptaamista. Havaintona on myös se, että käyttämällä jokerimerkkiä aiheessa, tilaaja voi tilata aiheita, joihin hänellä ei välttämättä pitäisi olla pääsyä. Keinona estää tämä Harsha et al. esittävät pääsynvalvontaluettelon käyttämistä.

#### Väsytyshyökkäys

Johtuen MQTT:n ominaisuuksista voidaan yksinkertaisia käyttäjätunnus-salasanapareja saada selville murtamalla ne. Sanakirjahyökkäyksessä käytetään sanalistoja (engl. Wordlist), joissa on tuhansia sanoja esimerkiksi tietovuodoista saatuja salasanoja tai yleisimpiä sanakirjasanoja [43]. Sanalistojen sanoja yritetään yksi kerrallaan välittäjälle, kunnes joku niistä osoittautuu oikeaksi, tätä kutsutaan väsytykseksi (engl. Brute-force). Yleisesti käytetyssä Metasploit-työkalussa on esimerkiksi tähän moduuli, joka tekee työn automaattisesti sen jälkeen, kun sille on tarvittavat parametrit syötetty [22]. Hyökkäyksen toiminta perustuu MQTT:n perusominaisuuteen, jossa välittäjä vastaa sille CONNECT-viestissä lähetettyyn käyttäjätunnus-salasanapariin edellisessä aliluvussa esitetyllä tavalla, jolloin vastauksena saadusta CONNACK-viestistä voidaan havaita milloin arvauksessa osutaan oikeaan.

#### Palvelunestohyökkäykset

Frilander [13] on pro-gradu -tutkielmassaan tutkinut Kali Linux-käyttöjärjestelmän käyttöä palvelunestohyökkäyksien ennaltaehkäisemisessä sekä selvittänyt viimeaikaisia tietoverkkopalveluiden uhkia. Hän on empiirisessä vaiheessaan tehnyt penetraatiotestaamalla palvelunestohyökkäyksiä www-palvelinta vasten ja arvioinut valitun käyttöjärjestelmän soveltuvuutta tähän. Tutkimuksessaan Frilander on nostanut hajautetut palvelunestohyökkäykset ja palvelunestohyökkäykset yleisimmiksi uhkiksi kirjallisuuskatsauksensa perusteella.

Palvelunestohyökkäyksessä pääasiallinen tavoite on ylittää palvelimen resurssit ja estää pääsy sen oikeilta asiakkailta. MQTT-protokollaa vastaan tehdyssä palvelunestossa hyökkääjä pyrkii tuottamaan niin paljon häiritsevää liikennettä, kuten julkaisu-, tilaus- ja yhteyspyyntöviestejä, että lopulta kaikki verkon resurssit kuluvat niihin ja normaalia palvelua ei pystytä tuottamaan [5]. Tätä kutsutaan englanniksi nimellä flooding attack, suomenkielinen nimitys voisi olla vaikkapa sitten tul-

vahyökkäys. Tulvahyökkäys voidaan jaotella neljään kategoriaan sen mukaan mitä mekanismia tulvan aikaansaamiseksi käytetään [46].

Perushyökkäyksessä hyökkääjä lähettää vain suuren määrän CONNECT-paketteja ja pyrkii ylittämään sen prosessointikyvyn aitojen pyyntöjen todentamisessa [46]. Viivästetyssä tulvassa hyökkääjä viivästyttää CONNECT-pakettien lähettämistä istunnon luomisen jälkeen, tämä johtaa suureen määrään puoliksi avattuja istuntoja välittäjän päässä, kun se odottaa CONNECT-pyynnön loppuun saattamista.

Hyötykuormatulvassa hyökkääjä kasvattaa hyötykuormallisen paketin kokoa kasvattamalla hyötykuorman kokoa [46]. Tämä johtaa kohdepalvelimella sekä koko kaistanleveyden että prosessointikyvyn käyttöön estäen siten uusien yhteyksien käsittelyn. Neljäntenä tapana asiakas, jolla on oikeat tunnistetiedot, mutta ei oikeuksia pystyy tulvalla hukuttamaan välittäjän virheellisillä tilaus- tai julkaisupyynnöillä. Tämä johtaa välittäjän prosessointikyvyn kulumisen yksittäisten pyyntöjen tarkistamiseen.

Palvelunestohyökkäys voidaan myös tehdä hyväksikäyttäen tietyn ohjelmistoversion haavoittuvuutta. Esimerkiksi Mosquitto-broker-ohjelmiston tietty versio on haavoittuvainen suurelle hyötykuormalle. Hyökkääjä pystyy kaatamaan haavoittuvaisen välittäjän lähettämällä sille sopivan kokoisen hyötykuorman, hitaalla siirtonopeudella. Samalla aikaa avataan uusia samanlaisia yhteyksiä ja lopputuloksena haavoittuvaisen järjestelmän muisti tulee täyteen jolloin se kaatuu [11].

#### Man in the middle -hyökkäys

Man in the middle -hyökkäyksessä hyökkääjä tunkeutuu kahden viestijän väliseen tiedonsiirtoon ilman, että viestijät itse huomaavat sitä [44]. Hyökkääjä pystyy esittämään olevansa jompikumpi viestivä osapuoli ja saa siten pääsyn molempiin viestijöihin. Tarkoituksena hyökkääjällä on joko salakuunnella liikennettä, vuorovaikuttaa siihen tai vaikuttaa viestijöihin. Hyökkääjä voi manipuloida viestejä ja niiden sisältöä, ohjata saman viestin toiselle käyttäjälle tai pysäyttää tiedonsiirron lähettäjän ja vastaanottajan välillä [5].

MitM-hyökkäyksen päämääränä on yleensä saada haltuun tietoa, kuten käyttäjätunnuksia, salasanoja ja luottokorttinumeroita [44]. MQTT-protokollassa MitM-hyökkäys voidaan toteuttaa välittäjän ja julkaisijan välillä. MitM-hyökkäyksen suorittamiseen voidaan käyttää työkaluina esimerkiksi Kali Linux:ia ja Ettercap:ia. MQTT-protokollassa välittäjän ja asiakkaan välinen kättely on altis MitM-hyökkäyksille.

# 5 MQTT:n tietoturvatestaus

Tässä luvussa esitellään tutkimuksen empiirinen osuus. Ensimmäisessä aliluvussa 5.1 esitellään testauksessa käytetty ympäristö. 5.2 aliluvussa kuvataan kevyesti ympäristön pystytys. Aliluvussa 5.3 esitetään tiedonkeruu MQTT-välittäjästä. Hyökkäystapauksina esitetään aliluvussa 5.4 väsytyshyökkäys ja aliluvussa 5.5 palvelunestohyökkäys. Viimeisessä aliluvussa 5.6 käydään hyökkäystapaukset uudelleen TLS-salausta käyttävää ympäristöä vastaan.

## 5.1 Ympäristön kuvaus

Testausta varten tarvitaan testausympäristö. Fyysisenä tietokoneena testauksessa käytetään minitietokonetta, jossa on AMD Ryzen$^{tm}$ 7 4700U-prosessori, 64 Gt muistia ja 1 Gb verkkokortti. Johtuen siitä, että tietokoneelle oli valmiiksi asennettu VMWare ESXi 6.7-hypervisori, niin ympäristö rakennettiin tämän päälle. Virtualisoitavaksi käyttöjärjestelmäksi valittiin Kali Linux, koska siinä on valmiiksi asennettuna useimmat tässä testauksessa käytettävistä ohjelmista.

Kali Linux on Debian GNU:n perustuva Linux-käyttöjärjestelmän jakelu, joka on kehitetty erityisesti penetraatio- ja tietoturvatestaukseen [8]. Vuoteen 2013 asti Kali Linuxin nimi oli BackTrack [25]. Kali Linux-käyttöjärjestelmä on saatavilla ilmaiseksi, ja sitä ylläpidetään ja päivitetään Offensive Securityn toimesta. Kali Linux sisältää monia tietoturvaan ja sen testaukseen liittyviä työkaluja, kuten Nmap-porttiskannerin, Metasploit Frameworkin, Wiresharkin, John the Ripper -salasanan purkutyökalun, hyökkäystyökalupakki Social Engineering Toolkitin (SET) ja monia muita. Kaiken kaikkiaan Kali Linux sisältää yli 600 erilaista työkalua, mukaan lukien verkkoanalyysityökalut, salausohjelmat, haavoittuvuusanalyysityökalut, etäyhteyden hallintaohjelmistot, tietokannan arviointityökalut, järjestelmän testaustyökalut, langattomat verkkotyökalut ja paljon monia muita [30]. Näiden avulla voidaan suorittaa erilaisia verkkotestauksia ja turvallisuusarviointeja, kuten penetraatiotestaus, haavoittuvuuksien etsimistä ja salauksiin ja niiden purkuun liittyviä tehtäviä.

Virtuaalikone käyttää hypervisoria virtualisoidakseen laitteistoresursseja, kuten

prosessoria, muistia, IO-laitteistoa [6]. Hypervisori mahdollistaa käytännössä usean virtuaalisen tietokoneen käyttämisen yhdellä fyysisellä tietokoneella. Nämä virtuaaliset tietokoneet voivat käyttää erilaisia sovelluksia samalla tavalla kuin fyysisetkin tietokoneet. Virtuaalikoneella taas käytetään Docker-ohjelmistoa ohjelmien ajamiseen konteissa.

Docker on kevyt virtualisointiteknologia, joka tarjoaa standardoidun tavan paketoida sovelluksia ja niiden riippuvuuksia yhdeksi suoritettavaksi yksiköksi, joita kutsutaan konteiksi [5]. Dockerin avulla sovellukset voidaan käyttöönottaa nopeasti eri ympäristöissä ja eri käyttöjärjestelmissä, mikä helpottaa kehittäjän työtä ja vähentää ympäristöjen asennukseen ja konfigurointiin käytettävää aikaa.

Kontit ovat suljettuja ympäristöjä, joissa sovellukset ja niiden riippuvuudet voidaan suorittaa eristettynä muusta järjestelmästä [1]. Kontit toimivat virtuaalisina paketteina, jotka sisältävät kaiken tarvittavan sovelluksen ajamiseen, kuten kirjastot, suorituskoneen ja asetukset. Ne ovat myös siirrettäviä ja skaalautuvia, mikä tarkoittaa, että niitä voidaan käyttää helposti eri ympäristöissä, kuten kehityksessä, testauksessa ja tuotannossa.

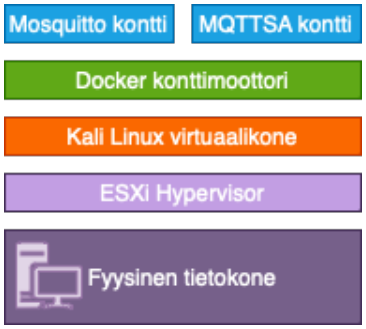
Kontit auttavat eristämään sovellukset toisistaan, mikä parantaa tietoturvaa ja vähentää konfliktien mahdollisuutta. Ne myös helpottavat sovellusten jakelua ja käyttöönottoa, sillä sovellukset voidaan pakata yhteen yksinkertaiseen konttiin, joka voidaan sitten asentaa ja käynnistää helposti eri ympäristöissä [1]. Tämä vähentää myös sovellusten yhteensopivuusongelmia ja mahdollistaa sujuvan ja nopean käyttöönoton eri ympäristöissä.

Käytettäessä kontteja sovellusten käyttöönotto ja hallinta on joustavampaa, nopeampaa ja tehokkaampaa. Suosittuja konttialustoja ovat esimerkiksi Docker ja Kubernetes, jotka tarjoavat monipuolisia ja skaalautuvia ratkaisuja sovellusten käyttöönottoon ja hallintaan. Testausta varten luodun ympäristön arkkitehtuuria kuvataan 5.1 kuvassa.

## 5.2 Ympäristön pystytys

Testausta varten ESXi-hypervisorille tehtiin uusi virtuaalikone. Koneelle allokoitiin 4 prosessoriydintä ja 4 Gt muistia. Ensimmäisenä hypervisorin päälle asennettiin käyttöjärjestelmäksi Kali Linux 2022.4 versiolla. Tämän jälkeen Kali Linuxiin asennettiin Docker.io-paketti pakettihallintaa käyttäen.

Dockerin asentamisen jälkeen virtuaalikoneelle käynnistetään Eclipse Mosquit-




Kuva 5.1: Testiympäristön arkkitehtuuri

to -kontti kuvan 5.2 mukaisesti. Eclipse Mosquitto on ilmainen avoimen lähdekoodin MQTT-välittäjä.

Mosquitto on suosittu MQTT-välittäjä, joka on saatavilla monille eri alustoille, kuten Linuxille, Windowsille ja macOS:lle. Mosquittoa käytetään laajalti IoT-laitteiden viestinvälitykseen, koska se on helppokäyttöinen ja skaalautuva. Palmieri et al. [40] ovat julkaisussaan todenneet, että noin 75% käytössä olevista MQTT-välittäjistä ovat Mosquitton eri versioita, tästä syystä tässä testauksessa valittiin Mosquitto välittäjäksi.

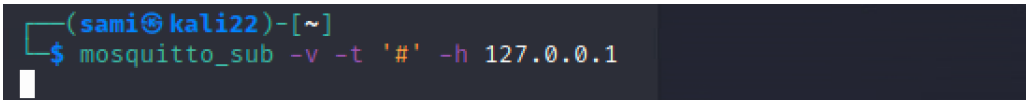
Ennen testaustapauksien aloittamista tehdään vielä hieman hienosäätöä välittäjään. Oletusasetuksena Mosquitto estää anonyymit yhteysyritykset asiakkailta eikä kuuntele muuta kuin omaa osoitettaan, tämän vuoksi testiskenaariota varten Mosquitton asetustiedostoon lisätään:

```
listener 1883 0.0.0.0
allow_anonymous true
```

```
┌──(sami㉿kali22)-[~/gradu/mosquitto]
└─$ sudo docker run -it --user mosquitto  -p 1883:1883 -p 9001:9001 -v /home/sa
mi/gradu/mosquitto/:/mosquitto/ eclipse-mosquitto
```

Kuva 5.2: Eclipse Mosquitton käynnistys kontissa

Seuraavana testiympäristöön tarvitaan tilaaja. Uuteen pääteikkunaan avataan mosquitto_sub-ohjelma kuvan 5.3 mukaisesti. Mosquitto_sub toimii testiympäristössä MQTT-tilaajana. Ohjelmalle annetaan parametrina `-t '#'` eli se kuuntelee kaikkia mahdollisia aiheita.

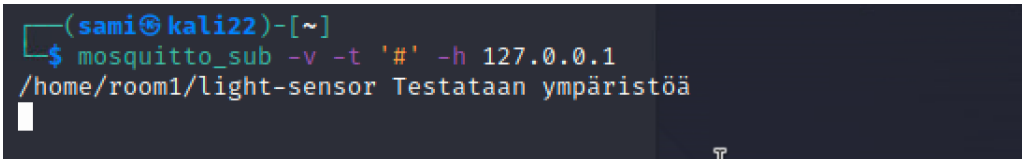



Kuva 5.3: mosquitto_sub käynnistys

Tämän jälkeen tarvitaan vielä julkaisija. Julkaisijana käytetään mosquitto_pub-ohjelmaa. Ohjelmalle annetaan parametrina `-t` eli aihe `'/home/room1/light-sensor'` ja parametrina `-m` eli viesti `'Testataan ympäristöä'` kuvan 5.4 mukaisesti.


Kuva 5.4: mosquitto_pub käynnistys ja viestin lähetys

Tilaajan päätteessä nähdään lähetetty viesti kuvan 5.5 mukaisesti ja voidaan todeta, että testiympäristö toimii. Kun ympäristö on saatu pystytettyä voidaan siirtyä itse testaamiseen.




Kuva 5.5: Lähetty viesti nähdään mosquitto_sub-ohjelman ikkunassa

Jotta aidon oloista tilannetta voidaan testata simuloidaan liikennettä luomalla pieni bash-skripti, joka generoi julkaisuviestejä välittäjälle kuvan 5.6 mukaisesti.

## 5.3 Tiedonkeruu välittäjästä

Kuten aliluvussa 4.4 esitettiin, ensimmäisenä vaiheena ennen hyökkäystä tehdään yleensä tiedonkeruu kohteena olevasta järjestelmästä. Tiedonkeruussa käytetään Nmap-ohjelmaa.

Nmap on ilmainen verkon skannaustyökalu, sen nimi on lyhennys sanoista ”Network mapper” [36]. Nmapin on kehittänyt Gordon ”Fyodor” Lyon vuonna 1997.

```
 GNU nano 7.2                          simulaattori.sh
#!/bin/bash

while :
do
    mosquitto_pub -t 'updates' -h 127.0.0.1 -m "{\"d\": \"$(date)\"}" -r
    sleep 5
    mosquitto_pub -t 'updates' -h 127.0.0.1 -m "{\"d\": \"$(date)\"}" -r
    sleep 5
    mosquitto_pub -t "updates" -h 127.0.0.1 -m "{\"d\": \"^FLAG{Th1s_1s_n0t_0K}$\"}"
    sleep 3
done
```

Kuva 5.6: Luodaan bash-skripti, jolla voidaan generoida liikennettä

```
┌──(sami㉿kali22)-[~]
└─$ mosquitto_sub -v -t '#' -h 127.0.0.1
/home/room1/light-sensor Testataan ympäristöä
updates {"d": "Sun Feb 12 10:43:24 PM EET 2023"}
/home/room1/light-sensor Testataan ympäristöä
updates {"d": "Sun Feb 19 11:03:32 AM EET 2023"}
updates {"d": "Sun Feb 19 11:03:37 AM EET 2023"}
updates {"d": "^FLAG{Th1s_1s_n0t_0K}$"}
updates {"d": "Sun Feb 19 11:03:45 AM EET 2023"}
updates {"d": "Sun Feb 19 11:03:50 AM EET 2023"}
updates {"d": "^FLAG{Th1s_1s_n0t_0K}$"}
updates {"d": "Sun Feb 19 11:03:58 AM EET 2023"}
updates {"d": "Sun Feb 19 11:04:03 AM EET 2023"}
updates {"d": "^FLAG{Th1s_1s_n0t_0K}$"}
```

Kuva 5.7: Skriptin toiminta nähdään mosquitto_sub-ohjelman ikkunassa

Ohjelma on ilmainen avoimen lähdekoodin työkalu, jota käytetään verkon tutkimiseen ja tietoturvatestaukseen. Nmapille annetaan parametrina verkko- tai ip-osoite, jolloin se hakee perustiedot kohteesta. Jos parametrina annetaan ip-osoite, niin skanneri palauttaa isäntänimen, joka osoitteeseen on liitettynä. Tämän jälkeen skanneri käy läpi kohdekoneen portit ja raportoi auki olevat portit sekä niihin liittyvät palvelut. Skanneri myös pyrkii palauttamaan arvion kohdekoneen käyttöjärjestelmästä [4]. Alunperin Nmap suunniteltiin skannaamaan nopeasti suuria verkkoja, mutta se toimii hyvin myös yksittäisiä isäntiä vastaan. Nmap toimii kaikissa yleisimmissä käyttöjärjestelmissä ja viralliset binääripaketit ovat saatavilla Linuxille, Windowsille ja macOS:lle.

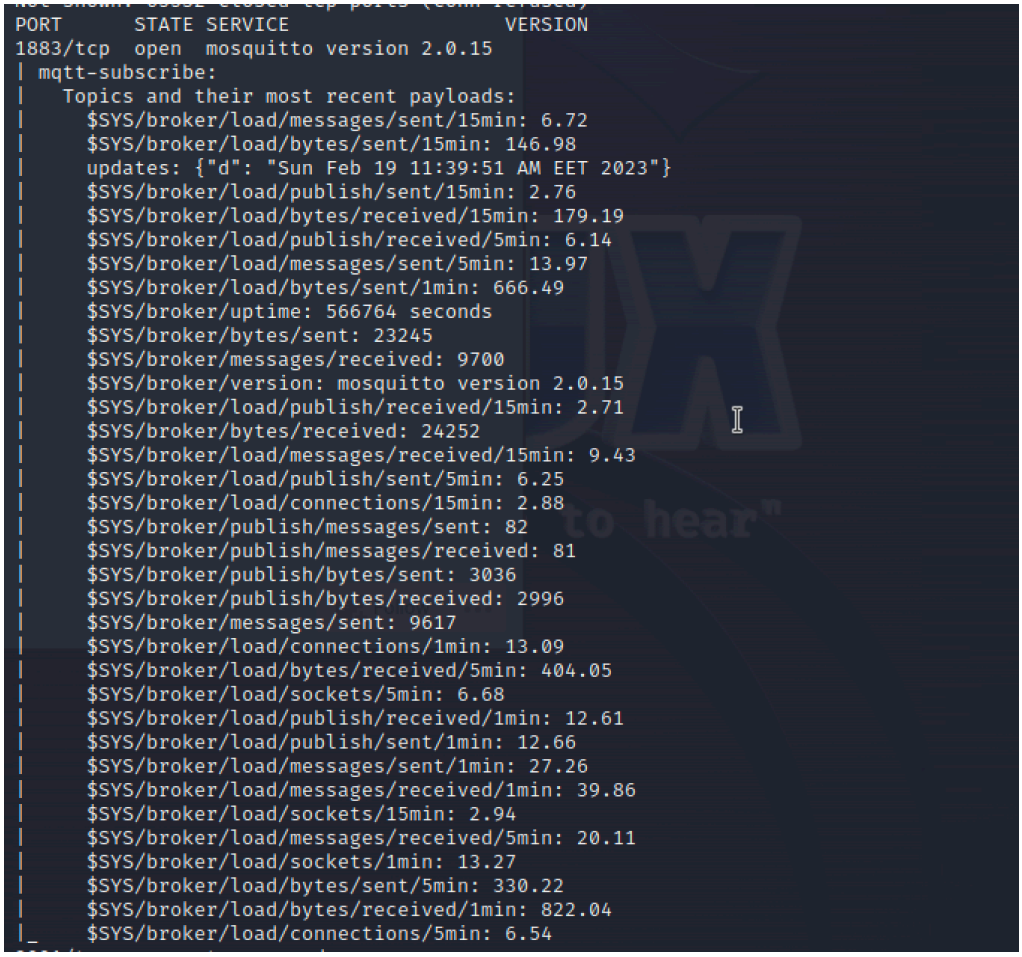
Nmap-ohjelmalle annetaan parametreina `-sV` eli service/version detection, jolloin se pyrkii selvittämään avoimien porttien tarjoamien palveluiden versiot. Kuvassa 5.8 `-sV`-parametrilla saatu tieto nähdään tulostusrivillä `1883/tcp open mosquitto version 2.0.15`.

Seuraavassa parametrissa `-sC` ohjataan Nmap käyttämään siinä valmiina olevia skriptejä, pieniä koodeja, joilla skannauksesta saadaan lisätietoja. Tässä skannauksessa esimerkiksi kuvassa 5.8 riviltä `mqtt-subscribe:` alkavat tiedot ovat `-sC`-parametrin käytön seurausta. Kuvassa nähdään myös `$SYS`-alkuisia aiheita, nämä ovat järjestelmä (engl. system) aiheita. Järjestelmäaiheet kertovat välittäjän tai asiakkaan tilatietoja.

Seuraavana parametrina annetaan `-p` eli skannattavat portit. Parametrilla voitaisiin rajoittaa skannattava portti johonkin tiettyyn, useampaan tai sitten jollekin välille. Kun halutaan skannata kaikki mahdolliset portit poislukien portti 0, käytetään parametria `-p-` eli tässä skannattava porttialue on 1-65535. Viimeinen parametri ennen IP-osoitetta on `-v`, joka lisää Nmapin tulostaman tiedon määrää. Viimeinen parametri on kohteen IP-osoite, joka tässä testiympäristössä osoittaa käytettyyn koneeseen itseensä. Kokonaisuudessaan käytetty komento on alla:

```
nmap -sV -sC -p- -v 127.0.0.1
```

Saadun tiedonkeruun perusteella nähdään, että välittäjä on portissa 1883, käytetty ohjelmisto on mosquitto ja sen versio 2.0.15. Tulosteessa nähdään myös, että välittäjällä on ainakin yksi aihe `updates`. Kun käyttäjien tunnistaminen ei ole käytössä, hyökkääjä voisi halutessaan seurata viestiliikennettä kuvassa 5.3 käytetyllä mosquitto_sub-ohjelmalla ja samassa kuvassa näkyvällä komennolla.




Kuva 5.8: Nmap skannauksen tuloksesta MQTT:n liittyvät löydökset.

## 5.4 Väsytyshyökkäys

Edellisessä aliluvussa esitetty viestien seuraaminen ei onnistu yhtä helposti, jos käytössä on käyttäjien tunnistaminen. Testauksessa käytetyssä Mosquitto-ohjelman versiossa tämä olikin asennettaessa kunnossa ja ensimmäistä skenaariota varten kytketty pois. Seuraavaa testausta varten edellisessä tehty muutos Mosquitton asetustiedostoon käydään korjaamassa siis takaisin alkuperäiseen muotoon:

```
#allow_anonymous true
```

Tämän lisäksi lisätään käyttöön tunnukset sisältävä tiedosto lisäämällä asetustiedostoon:

```
password_file /mosquitto/config/password_file.txt
```

Tämän jälkeen luodaan mosquitto_passwd-ohjelmalla tarkoituksellisesti turvallisuudeltaan heikko käyttäjätunnus-salasanapari:

```
mosquitto_passwd -c -b password_file.txt admin salasana123
```

Vaikka asianmukaiset käyttäjän tunnistamisen vaatimukset olisivat päällä, voidaan viestejä kuitenkin päästä seuraamaan, jos tunnukset saadaan hankittua jollain keinolla käyttöön. Yksinkertaisen todennusmekanismin, kuten käyttäjänimen ja heikon salasanan käyttö Mosquitto-välittäjään yhdistämisessä, voi vaarantaa suojauksen. Tässä testauksessa käytetään väsytyshyökkäyksenä sanakirjahyökkäystä avuksi tunnuksien haltuun saamiseksi.

Sanakirjahyökkäykseen voidaan käyttää monia eri työkaluja. Tässä testitapauksessa käytetään esiasennettuna Kali Linux-käyttöjärjestelmästä löytyvää Metasploitia. Metasploit Framework on avoimen lähdekoodin työkalu, jota voidaan hyödyntää haavoittuvuuksien tutkimiseen ja hyväksikäyttöön [21]. Metasploitin avulla voidaan käyttää valmiita tai räätälöityjä skriptejä heikkouksien löytämiseen ja pääsynsaamiseksi haavoittuvaan järjestelmään.

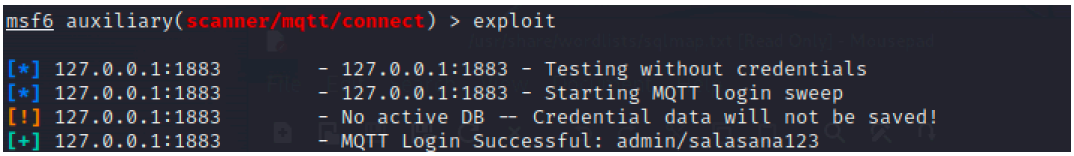
Metasploit on saavuttanut vahvan jalansijan tietoturva-alalla ja se on käytännössä de facto-standardi haavoittuvuuksien hyväksikäyttämisien kehittämiseen. Metasploit tarjoaa valtavan määrän erilaisia hyökkäystyökaluja, jotka ovat helposti saatavilla sen käyttöliittymän kautta. Näitä työkaluja kutsutaan Metasploitissa moduuleiksi [32]. Moduulit jaetaan niiden käyttötarkoituksen perusteella seitsemään ryhmään: hyötykuormat, hyväksikäytöt, enkoodaajat, välttelyt, NOP-generaattorit, apumoduulit ja jälkikäyttömoduulit.

Metasploit tarjoaa käyttäjille erilaisia toimintoja, kuten haavoittuvuuksien tunnistamista, haavoittuvuuksien hyväksikäyttöä, takaovien luomista ja erilaisia verkkotyökaluja. Sen avulla käyttäjät voivat testata järjestelmiä ja sovelluksia haavoittuvuuksien varalta ja kehittää automaattisia hyökkäyksiä, jotta voidaan testata, miten kohdejärjestelmät ja -sovellukset reagoivat tällaisiin hyökkäyksiin.

MQTT:n sanakirjahyökkäystä varten Metasploitista löytyy valmiina apumoduuli MQTT skannaukseen `auxiliary/scanner/mqtt/connect`. Moduuli tarvitsee parametrikseen sanalista-tiedoston. Englanninkielisiä sanalistoja on jonkin verran tarjolla valmiiksi asennettuna Kali Linuxin mukana. Nämä löytyvät Kali Linuxissa sijainnista `/usr/share/wordlists`. Kali Linuxin valmiista sanalistoista löytyy kuitenkin melko huonosti suomenkielen sanoja. Yhtenä keinona on tuottaa itse omaan käyttötarkoitukseen sopiva sanalista jostain valmiista lähteestä esimerkiksi Kotimaisten kielten keskuksen nykysuomen sanalistasta, josta löytyy noin 94000 sanatietuetta [26].

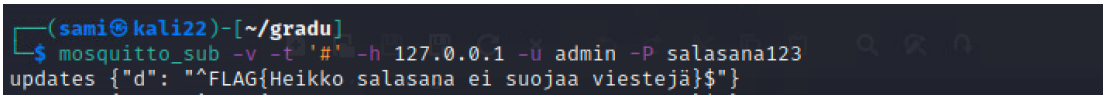
Tätä testausta varten luodaan yksinkertainen sanalista, joka sisältää joitakin salasanoja. Käyttäjätunnuksien murtamiseen käytetään Kali Linuxin mukana tulevaa `/usr/share/wordlists/metasploit/unix_users.txt`-sanalistaa.

Suorittamalla valittu moduuli exploit-komennolla saadaan hyvin nopeasti onnistunut kirjautuminen kuvan 5.9 esittämällä tavalla. Kun tunnus ja salasana on




Kuva 5.9: Metasploit tunnus-salasanaparin murtaminen

saatu selville voidaan jälleen käynnistää mosquitto_sub-ohjelma ja antaa sille parametrina saadut tunnukset. Kuten kuvassa 5.10 voidaan nähdä, voidaan liikennettä jälleen seurata. Vastaavasti hankittuja tunnuksia voitaisiin käyttää peukaloitujen




Kuva 5.10: Murretun käytäjätunnus-salasanaparin käyttäminen

viestien syöttämiseksi liikenteen sekaan mosquitto_pub-ohjelmalla kuten kuvassa

5.4.

Kirjautumistunnukset voidaan saada haltuun myös ilman sanakirjahyökkäystä. Kun tiedossa on, että kohdejärjestelmässä on MQTT-välittäjä voidaan MQTT-liikennettä pyrkiä seuraamaan. Tämä voidaan toteuttaa esimerkiksi Wireshark-ohjelmalla.

Wireshark on ilmainen avoimen lähdekoodin verkkotyökalu, joka mahdollistaa tietoliikenteen tarkkailun ja analysoinnin [52]. Se on suunniteltu tukemaan satoja erilaisia protokollia, kuten TCP, UDP, MQTT ja HTTP. Wireshark pystyy tallentamaan ja esittämään tietoa verkon liikenteestä reaaliajassa tai tallentamaan sen myöhemmin analysoitavaksi.

Wiresharkin käyttöliittymä koostuu eri osista, jotka helpottavat tietoliikenteen analysointia. Käyttäjä voi valita, mitä protokollaa tai osaa tietoliikenteestä haluaa tarkkailla ja tarkastella siitä syntyviä paketteja. Wiresharkin avulla käyttäjä voi myös suorittaa erilaisia tehtäviä, kuten etsiä tiettyä tietoa, suodattaa tietoliikennettä tietyillä kriteereillä, tunnistaa virheitä tai ongelmia verkossa. Ohjelman käyttö edellyttää jonkin verran tietoa verkkoteknologioista ja -protokollista, jotta voidaan tulkita oikein kerätty tieto.

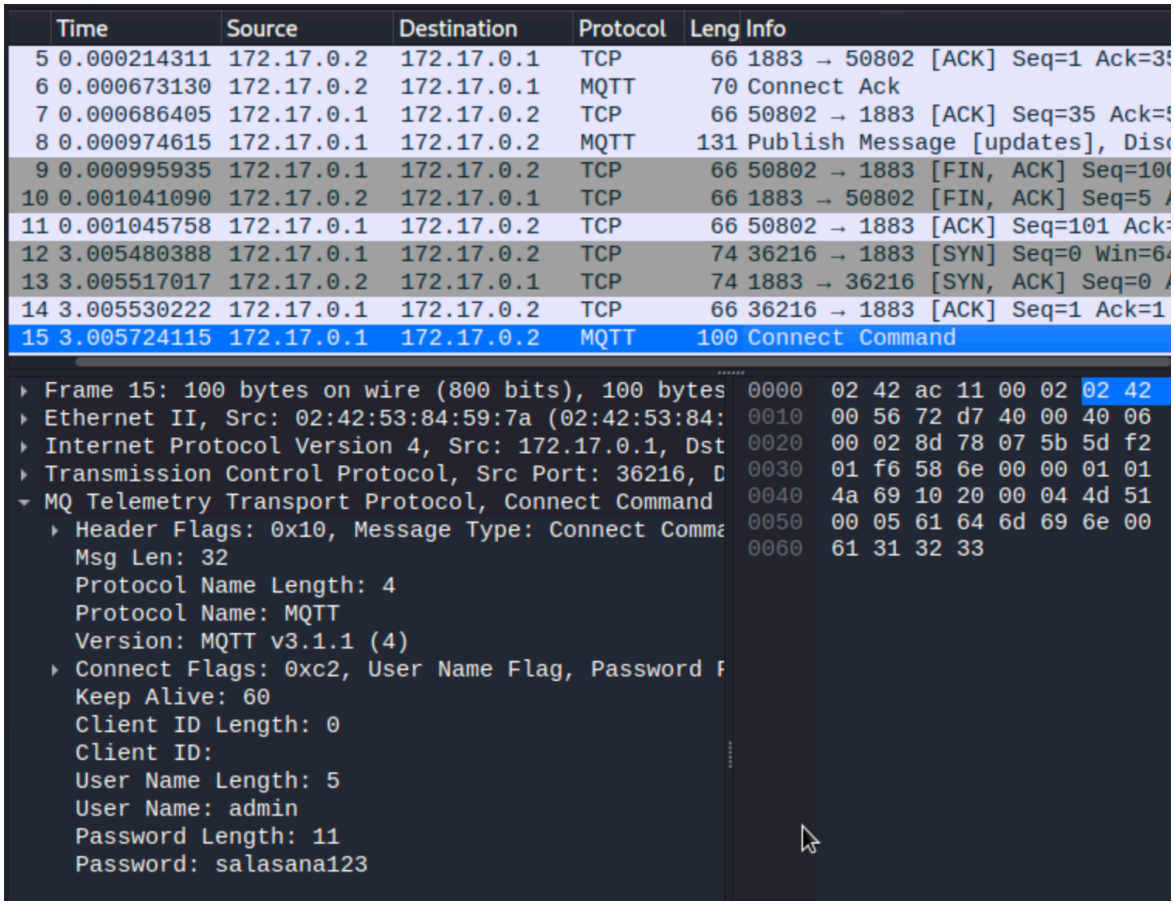
Wiresharkin avoimen lähdekoodin luonteen vuoksi sitä kehitetään ja laajennetaan jatkuvasti. Käyttäjät voivat lisätä uusia protokollia tai ominaisuuksia kehittämällä omia laajennuksiaan. Wireshark on suosittu työkalu monien verkkotyökalujen ja -sovellusten kehittäjien keskuudessa.

Wireshark asetetaan seuraamaan virtuaaliverkkokorttia, jossa kontit toimivat. Ohjelmalla kaapataan välittäjän ja asiakkaan välistä liikennettä tilanteesta, jossa julkaisija julkaisee viestinsä. Kaapatusta liikenteestä saadaan helposti suodattamalla selkokielisenä selville asiakkaiden kirjautumistiedot. Tämä nähdään 5.11 kuvassa MQTT CONNECT-paketin lipuissa.

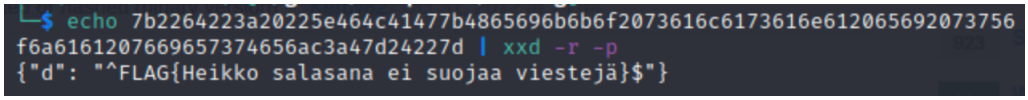
Liikenteestä voidaan nähdä myös aiheet selkokielisenä ja viestit heksamuodossa, joka voidaan helposti muuttaa selkokieliseksi, esimerkiksi Kali Linuxin komentotulkissa kuvassa 5.12 esitetyllä tavalla.

## 5.5 Palvelunestohyökkäys

Palvelunestohyökkäykseen toteuttamiseksi tarvitaan jokin keino tuottaa hyötykuorma tai liikennettä, jolla voidaan estää normaali MQTT-viestintä. Hyökkäyksen tekemiseksi voisi käyttää tiedonkeruu vaiheessa saatua tietoa välittäjästä ja sen versios-




Kuva 5.11: Käyttäjätunnus ja salasana paljastuu liikennettä seuraamalla




Kuva 5.12: Heksamuodossa olevan viestin purkaminen komentotulkissa

ta ja tietojen perusteella kohdistaa jonkin kyseiseen versioon kohdennetun haavoittuvuuden välittäjään. Testauksessa käytetystä Mosquitton versiosta ei kuitenkaan ollut julkaistuja haavoittuvuuksia ja näin ollen keinoksi jäi yleisempien palvelunestotapojen käyttäminen.

Tiedonhankinnan perusteella hyökkäyksen tekemisen työkaluksi valikoitui MQTTSA. MQTTSA eli MQTT Security Assistant on ohjelma, jolla voidaan havaita haavoittuvuuksia MQTT-välittäjistä. Palmieri et al. [40] ovat kehittäneet MQTTSA:n parantaakseen IoT-järjestelmien turvallisuutta ja lisätäkseen järjestelmien kehittäjien turvallisuustietoisuutta. Ohjelma suorittaa joukon erilaisia tunnettuja hyökkäystapoja MQTT-välittäjää vastaan. Hyökkäysten tarkoituksena on paljastaa protokollaan ja välittäjään liittyviä tunnettuja haavoittuvuuksia. Hyökkäyksen jälkeen ohjelma antaa joukon lieventämistoimenpiteitä havaittuja haavoittuvuuksia vastaan.

Ensimäisessä vaiheessa yhteysmoduuli yrittää yhdistää välittäjään. CONNECT-paketin paluukoodin perusteella yritetään selvittää, onko käyttäjien tunnistaminen käytössä [40]. Tämän jälkeen, jos tunnistautuminen vaaditaan, ohjelma pyrkii annetuilla parametreilla tunnistautumaan välittäjälle ja tarpeen mukaan, joko liikennettä seuraamalla tai sanakirjahyökkäyksellä saamaan selville tunnistautumiseen käytettyjä käyttäjätunnus-salasanapareja. Tässä testauksessa näitä ominaisuuksia ei käytetä, koska tunnukset on saatu jo selville aiemmissa testaustapauksissa. Ohjelma käyttää kuitenkin saman tyyppisiä hyökkäystapoja, kuin testaustapauksissa 5.3 ja 5.4.

Johtuen siitä, että MQTTSA-sovellus on kehitetty jo joitain vuosi sitten, sen ajaminen testauksen ajanhetkellä osoittautui hieman haastelliseksi, mm. nykyisen Python3 ympäristön liiallisesta eroavaisuudesta sovelluksen kehittämisajankohtaan nähden. Näistä haasteista johtuen sovellusta ajettiin testaustilanteessa kontissa, jossa oli käyttöjärjestelmänä Ubuntu 20.04 ja Python3 ympäristönä 3.8.10.

```
sudo docker run --rm -it ubuntu:20.04
```

Tämän jälkeen konttiin asennettiin tarvittavat sovellukset:

```
apt update && apt install -y git python3 python3-pip mosquitto
mosquitto-clients tshark
```

Jonka jälkeen haettiin git-ohjelmaan käyttäen MQTTSA:n lähdekoodi:

```
git clone https://github.com/stfbk/mqttsa.git
```

Lopuksi vielä käännettiin koodi:

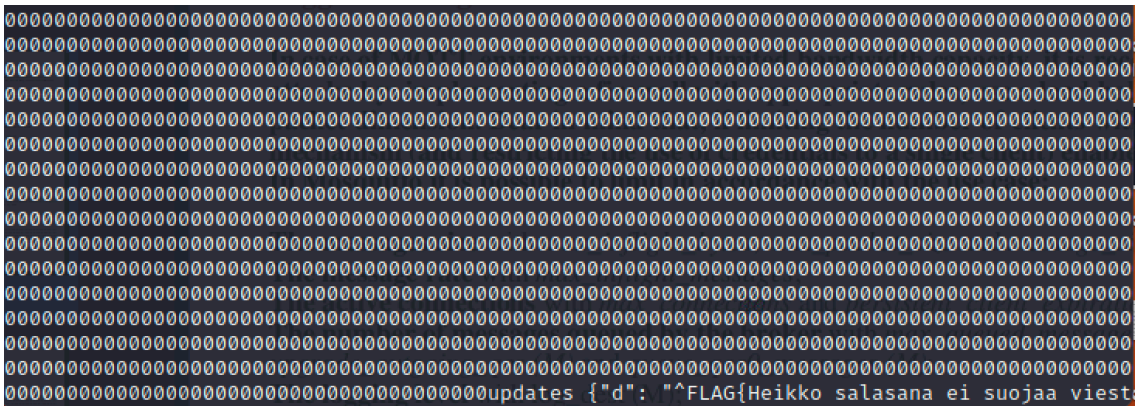
```
cd mqttsa && make
```

Palvelunestohyökkäyksen lisäksi MQTTSA tekee useita muita testauksia kohdejärjestelmää vastaan. Ohjelma suoritettiin testauksessa komennolla:

```
python3 mqttsa.py -u admin -w sana.txt -t 10 -fc 1000 -sc 1000
-mq 1500 -mp 300 --md 172.17.0.2
```

Oleelliset parametrit palvelunestohyökkäyksen toteuttamiseksi ovat `-fc` eli ohjelman yrittämien yhteyksien määrä, jolla se pyrkii tekemään perustulvahyökkäyksen, `-sc` joka asettaa yritettävien yhteysyrityksien määrän viivästetty tulva -tyypisessä hyökkäyksessä. Parametri `-mq` asettaa viestien määrän, joita koitetaan ajaa välittäjän jonoon ja `-mp` parametrilla asetetaan testauksessa käytettävän hyötykuorman maksimikoko hyötykuormatulvahyökkäyksessä.

Työkalu myös tuottaa tehdystä hyökkäyksestä valmiin raportin. Raportin perusteella havaitaan järjestelmän olevan haavoittuva hyötykuorman koon kasvattamiselle, sekä viivästetylle tulvalla. Tämä voidaan käytännössä myös todeta seuraamalla tilaajaikkunan tapahtumia. Kun hyökkäys on menossa ei normaaliliikenne pääse lainkaan läpi kuvan 5.13 mukaisesti.




Kuva 5.13: Normaali viestintä estyy hyökkäyksen aikana

## 5.6 Salauksen käyttöönotto

Kuten aiemmissa tutkimuksissa oli havaittu, voidaan edeltävissä testitapauksissa esitettyjä hyökkäyksiä mitigoida käyttämällä tietoliikenteen salausta. Viimeisessä

testitapauksessa tietoliikenne salattiin käyttäen TLS 1.3-salausta ja yritettiin aiempia testitapauksia uudelleen.

TLS-salausta varten tarvitaan varmenteita. Ohjelma, jota varmenteiden luomiseen käytetään on OpenSSL. OpenSSL on avoimen lähdekoodin kryptografian kirjasto, joka tarjoaa monia erilaisia kryptografisia toimintoja, kuten salaus, purku, digitaalinen allekirjoitus ja varmennus. Se sisältää myös työkaluja, kuten komentorivipohjaisen käyttöliittymän, jolla voidaan käyttää kirjaston toimintoja erilaisissa sovelluksissa. Ensimmäisenä luodaan varmentajalle (engl. Certificate Authority, CA) yksityinen avain. Avaimella voidaan allekirjoittaa varmenteita. Yksityinen avain voidaan luoda Kali Linuxissa suoraan komennolla:

```
openssl genrsa -aes256 -out mqtt_ca.key 2048
```

Komento `genrsa` kertoo ohjelmalle, että halutaan luoda uusi yksityinen avain. Parametrilla `-aes256` asetetaan avaimen salaukseen käytettävä salausmuoto AES-256:ksi. Parametri `-out mqtt_ca.key` asettaa nimen avaintiedostolle ja viimeinen parametri `2048` kertoo avaimen pituuden tavuina. Tämän jälkeen luodaan juurivarmenne ja allekirjoitetaan se edellä luodulla yksityisellä avaimella.

```
openssl req -new -x509 -days 3650 -key mqtt_ca.key
-out mqtt_ca.crt
```

Varmenteen luomiseen käytetään jälleen OpenSSL-ohjelmaa. Ohjelmalle annetaan komentona `req` eli pyydetään varmennetta. Parametri `-new` kertoo ohjelmalle, että halutaan luoda uusi varmenne. Parametrilla `-x509` tässä kerrotaan ohjelmalle, että halutaan luoda itse allekirjoitettu varmenne. Varmenteen voimassaoloaika asetetaan `-day` parametrilla 3650 päiväksi. Varmenteen allekirjoitukseen käytetty avain annetaan `-key` parametrilla ja lopuksi kerrotaan pyydetylle varmenteelle annettava nimi.

Seuraavaksi luodaan Mosquitto-välittäjälle yksityinen avain komennolla:

```
openssl genrsa -out mqtt_srv.key 2048
```

Avain tarvitsee vielä allekirjoittaa varmentajan toimesta, tätä varten luodaan varmennepyyntö.

```
openssl req -new -out mqtt_srv.csr -key mqtt_srv.key
```

Tässä parametrit kertovat ohjelmalle, että halutaan luoda allekirjoituspyyntötiedosto nimeltään `mqtt_srv.csr` avaimelle `mqtt_srv.key`.

Viimeiseksi vielä varmennetaan ja allekirjoitetaan varmennepyyntö.

```
openssl x509 -req -in mqtt_srv.csr -CA mqtt_ca.crt -CAkey
mqtt_ca.key -CAcreateserial -out mqtt_srv.crt -days 3650
```

Tämän jälkeen muutetaan vielä Mosquiton asetustiedostossa välittäjä kuuntelemaan porttia 8883 sekä käyttämään luotuja varmenteita ja avainta.

```
listener 8883 0.0.0.0
cafile /mosquitto/config/mqtt_ca.crt
certfile /mosquitto/config/mqtt_srv.crt
keyfile /mosquitto/config/mqtt_srv.key
```

Aiemmat testaustapaukset suoritetaan nyt TLS-suojattua liikennettä vastaan. Nmapilla saatiin selville, että kohdekoneella oli portti 8883 auki ja että siihen liittyi ssl/mqtt-palvelu, muttei muuta kuten kuvasta 5.14 voidaan nähdä.

```
└─$ nmap -sV -p- -v localhost
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-09 21:58 EET
NSE: Loaded 45 scripts for scanning.
Initiating Ping Scan at 21:58
Scanning localhost (127.0.0.1) [2 ports]
Completed Ping Scan at 21:58, 0.00s elapsed (1 total hosts)
Initiating Connect Scan at 21:58
Scanning localhost (127.0.0.1) [65535 ports]
Discovered open port 42125/tcp on 127.0.0.1
Discovered open port 8883/tcp on 127.0.0.1
Discovered open port 9001/tcp on 127.0.0.1
Completed Connect Scan at 21:59, 1.79s elapsed (65535 total ports)
Initiating Service scan at 21:59
Scanning 3 services on localhost (127.0.0.1)
Completed Service scan at 22:00, 86.10s elapsed (3 services on 1 host)
NSE: Script scanning 127.0.0.1.
Initiating NSE at 22:00
Completed NSE at 22:00, 0.06s elapsed
Initiating NSE at 22:00
Completed NSE at 22:00, 1.01s elapsed
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000069s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 65532 closed tcp ports (conn-refused)
PORT      STATE SERVICE    VERSION
8883/tcp  open  ssl/mqtt
9001/tcp  open  tcpwrapped
42125/tcp open  unknown
```

Kuva 5.14: Nmap TLS-salausta käyttävää välittäjää vastaan

Metasploitablella ajettiin uudelleen 5.4 aliluvussa kuvattu apumoduuli, sillä erolla, että kohdeportiksi vaihdettiin 8883. Väsytyshyökkäys ei kuitenkaan enää onnistunut lainkaan, vaan ohjelma ilmoitti, ettei se saa oikeita tietoja CONNECT-paketin otsikosta kuvan 5.15 mukaisesti.

```
msf6 auxiliary(scanner/mqtt/connect) > exploit

[*] 127.0.0.1:8883        - 127.0.0.1:8883 - Testing without credentials
[*] 127.0.0.1:8883        - Error: 127.0.0.1: MQTT::ProtocolException Invalid flags in
CONNECT packet header
[*] 127.0.0.1:8883        - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Kuva 5.15: Metasploitable ei onnistu enää murtamaan käyttäjätunnusta ja salasanaa

Viimeiseksi suoritettiin palvelunestohyökkäys MQTTSA:ta käyttäen, aiempaan erona `-p 8883` parametrin lisäys, jotta ohjelma ymmärtää yrittää hyökkäystä oikeaa porttia vastaan.

```
python3 mqttsa.py -u admin -w sana.txt -t 10 -fc 1000 -sc 1000
-mq 1500 -mp 300 --md 172.17.0.2 -p 8883
```

Hyökkäys ei onnistunut lainkaan, koska ohjelma ei saanut muodostettua välittäjään yhteyttä. Kuvassa 5.16 nähdään Wiresharkilla seurattuna verkon liikenteessä välittäjän ja MQTTSA-ohjelman TLS-kättely, joka päätyy yhteyden hylkäämiseen välittäjän osalta, koska asiakkaalla ei ollut käytössä oikeaa varmennetta.

| No | Time | Source | Destination | Protoc | Len | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | 172.17.0.2 | 172.17.0.3 | TCP | 74 | 33357 → 8883 [SYN] Seq=0 Win=64 |
| 2 | 0.000019216 | 172.17.0.3 | 172.17.0.2 | TCP | 74 | 8883 → 33357 [SYN, ACK] Seq=0 A |
| 3 | 0.000030027 | 172.17.0.2 | 172.17.0.3 | TCP | 66 | 33357 → 8883 [ACK] Seq=1 Ack=1 |
| 4 | 0.000148369 | 172.17.0.2 | 172.17.0.3 | TLS… | 80 | Ignored Unknown Record |

Kuva 5.16: MQTTSA TLS-kättely epäonnistuu

Aiemmin luotu varmenne `mqtt_ca.crt` ladattiin MQTTSA-ohjelmaa suorittavaan konttiin ja annettiin parametrina ohjelmalle. Tämän jälkeen hyökkäys onnistui samalla tavalla kuin testitapauksessa 5.5.

# 6 Pohdinta

Tässä luvussa pohditaan testauksen tuloksia ja tehdään niistä johtopäätöksiä. Tarkoituksena oli selvittää MQTT-protokollaan hyökkäysmekanismeja ja miten niitä pystyttäisiin mitigoimaan.

## 6.1 Tiedonkeruu ja liikenteen salakuuntelu

Tiedonkeruuvaiheessa saatiin selville järjestelmässä käytetyn välittäjäohjelman nimi ja versio. Tämän lisäksi nähtiin välittäjällä tarjolla olevat aiheet. Näistä suurin osa oli järjestelmäaiheita, joissa välittäjä kertoi tilatietojaan. Järjestelmäaiheissa voi välittyä myös luottamuksellista tietoa, jonka vuoksi niihin pääsyä tulisi rajoittaa. Rajoitus voitaisiin asettaa käyttämällä pääsynvalvontaluetteloa. Pääsynvalvontaluetteloa käytettäessä asetetaan tietylle käyttäjällä oikeudet lukea, kirjoittaa, lukea ja kirjoittaa tai ei oikeutta lukea eikä kirjoittaa tiettyyn aiheeseen. Hyvänä ratkaisuna olisi rajoittaa asiakkaita olemasta vuorovaikutuksesta muihin aiheisiin, kuin niihin, joissa etuliitteenä on niiden oma clientID sekä estää anonyymejä käyttäjiä tilaamasta $SYS-aiheiden viestejä.

Ensimmäisessä testitapauksessa ajettiin Eclipse Mosquitto -välittäjää niin, että anonyymikäyttö oli sallittua. Viestiliikennettä pystyttiin salakuuntelemaan helposti ja liikenteen joukkoon olisi voitu syöttää yhtä helposti sinne kuulumattomia viestejä. Tämän mahdollistamiseksi testatussa Mosquitto versiossa manuaalisesti muutettiin asetuksia niin, että anonyymikäyttö mahdollistui, koska oletuksena se oli estettynä. Koska mitään rajoituksia viestien julkaisulle tai tilaamiselle ei ollut asetettu, kuka tahansa pystyi liittymään verkkoon toimittamaan näitä tehtäviä. Tietoturvanäkökulmasta tiedon luottamuksellisuus, eheys ja saavutettavuus vaarantuivat tässä testiasetelmassa.

Onkin melko turvallista todeta, että välittäjää ei ole syytä ajaa testiasetelman mukaisilla asetuksilla vaan anonyymikäyttö tulee aina olla estetty. Poikkeuksena tähän voisi olla tilanne, jossa liikenteeseen ei ole minkäänlaista pääsyä mistään viestinnän ulkopuolelta. Jos salauksen käyttöönotto on mahdollista, olisi hyvä salata liikenne TLS-salausta käyttäen. TLS-salauksen käyttö edellyttää, että kullakin asiakkaalla on

hallussaan varmenne, jolla se pystyy salaamaan ja purkamaan sen ja välittäjän välistä viestintää. Jos varmenne olisi valvomattomalla fyysisellä laitteella, se saattaisi olla mahdollista kaapata ja käyttää sitä pääsykeinona viestiliikenteeseen.

## 6.2 Asiakkaan tunnistaminen

Toisessa testitapauksessa anonyymikäyttö laitettiin pois päältä ja käyttöön valittiin tarkoituksella heikko käyttäjätunnus-salasanapari. Melko pienellä vaivalla heikko salasana saatiin murrettua käyttäen sanakirjahyökkäystä ja käytännössä oltiin taas ensimmäisen testaustapauksen lähtötilanteessa.

Sanakirjahyökkäyksiä vastaan hyviä salasanoja ovat riittävän pitkät ja monimutkaiset salasanat. Perusmuotoisia sanoja kannattaa välttää, koska esimerkiksi Kotuksen sivuilta voi ladata ilmaiseksi nykysuomen sanaston, josta pienellä vaivalla saa generoitua suomenkielen sanalistan. Yleisesti voidaan todeta, että suomenkielen taivutusmuotojen käyttö salasanoissa lisää vaikeusastetta huomattavaksi ja on siksi suositeltavaa. Sanakirjahyökkäyksiä vastaan tehokas mitigointikeino olisi myös X.509 asiakasvarmenteiden käyttö asiakkaiden todennukseen, asiakasvarmenteiden käyttöä rajautui kuitenkin tämän tutkimuksen ulkopuolelle.

Käyttäjätunnuksia ja salasanoja voidaan myös poimia liikennettä salakuuntelemalla. Salaamattomasta liikenteestä voidaan liikennettä seuraamalla saada asiakkaan CONNECT-paketista selkokielisenä sen käyttämä käyttäjätunnus ja salasana. Näissä tilanteissa salasanan monimutkaisuudella ei ole käytännössä mitään merkitystä.

Koska käyttäjätunnukset ja salasanat saatiin haltuun, voidaan todeta, että tietoturvanäkökulmasta tiedon luottamuksellisuus ja eheys vaarantuivat. TLS 1.3-salauksen avulla pystyttiin viimeisessä testausvaiheessa estämään tietojen poimiminen liikenteenjoukosta, myöskään valittu Metasploitable-moduuli ei pystynyt tunnuksia väsytyshyökkäyksellä murtamaan.

Salausta käytettäessä verkkoliikennettä seuraamalla nähtiin selkokielellä ainoastaan TLS-kättely ja tämän jälkeen liikenne oli salattua. Aiempiin TLS-versioihin verrattuna TLS 1.3:ssa varmenteetkin liikkuvat salattuna, etkä niitä näin ollen ole mahdollista poimia liikenteestä. Vaikka salakuuntelijalla olisi ollut käytössä salaukseen käytetty varmenne, se ei olisi mitä ilmeisimmin pystynyt purkamaan muiden asiakkaiden ja välittäjän välistä salausta ilman, että se olisi saanut jostain käsiinsä istunnon avaimen. Omia viestejään hyökkääjä olisi kuitenkin pystynyt syöttämään välit-

täjälle, jos sillä olisi ollut varmenne käytössään, jolloin tiedon eheys olisi vaarantunut.

## 6.3 Palvelunesto

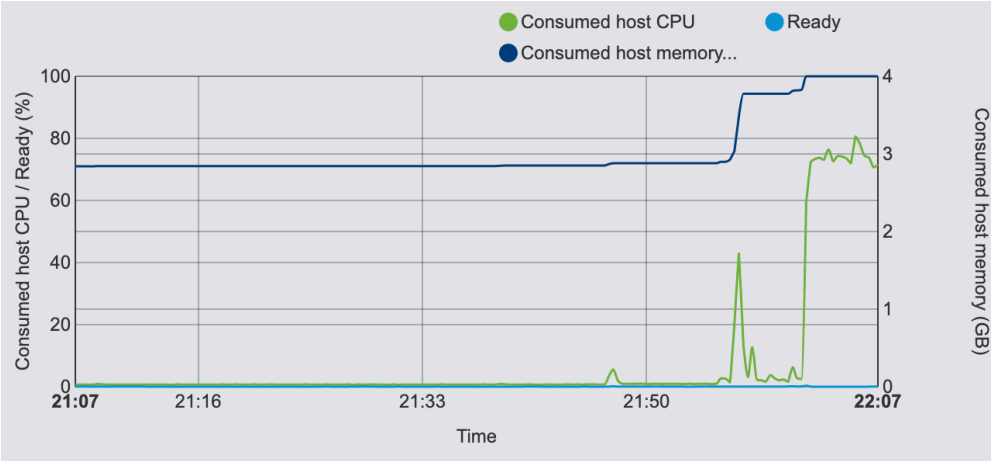
Kolmannessa tapauksessa edellisessä testauksessa haltuun saaduilla tunnuksilla toteutettiin palvelunestohyökkäys viestiliikennettä vastaan. Hyökkäyksessä mallinnettiin eri palvelunestohyökkäystapoja. Hyökkäys toteutettiin MQTTSA-ohjelmaa käyttäen. MQTTSA toteuttaa automaattisesti eri palvelunestohyökkäyksiä sekä testaa välittäjää tunnettujen haavoittuvuuksien osalta. MQTTSA toteuttaa hyökkäyksiä päällekkäin, mutta selkeyden vuoksi ne tässä eroteltu kolmeksi eri hyökkäykseksi. Ohjelma tekee myös muita testauksia mm. viestijonoihin ja väärinmuodostettuihin paketteihin liittyen, mutta niitä ei tässä testauksessa huomioitu.

Ensimmäisessä hyökkäyksessä käytettiin tulvahyökkäystä, jossa ohjelma teki välittäjälle yhteyspyyntöjä CONNECT-pakettien avulla ja tämän jälkeen julkaisi jokaisella avatulla yhteydellä 100 viestiä. Ohjelma myös mittasi aikaa, joka julkaistun lähetyksen ja välittäjältä saadun PUBACK-paketin välillä kului. Havaittiin, että maksimimäärä yhtäaikaisia yhteyksiä oli 340. Alkuperäisten asiakkaiden ja välittäjän välinen yhteys ei katkennut. Jos näin olisi tapahtunut, olisi salaamattomassa liikenteessä ollut mahdollista salakuunnella asiakkaiden kirjautumistiedot, kun ne pyrkivät avaamaan uudelleen yhteyden välittäjään.

Toisessa hyökkäyksessä käytettiin viivästettyä tulvaa, avaamalla useita yhteyksiä välittäjään ja sen jälkeen jättämällä yhteys auki ja välittäjä odottamaan seuraavaa pakettia. Parametrina hyökkäykselle annettiin 1000 yhteysyritystä. Raportin mukaan välittäjä hyväksyi 240 yhtä aikaista yhteyttä, eikä tämän jälkeen enää ottanut uusia yhteyksiä vastaan. Tietoturvanäkökulmasta tämä olisi estänyt tietojen saatavuuden, jos jokin uusi asiakas olisi pyrkinyt välittäjään yhdistämään.

Kolmannessa hyökkäyksessä käytettiin mekanismina hyötykuormatulvaa. MQTTSA:n toteuttamistapa ei dokumentaation perusteella täysin selvinnyt, mutta lähdekoodia tutkimalla pystyi päättelemään, että hyökkäyksessä käytettiin sekä julkaisijaa että tilaajaa. Julkaisun hyötykuormaa kasvatettiin 5 Mt kerrallaan, kunnes välittäjän resurssit eivät enää riittäneet ja se ei välittänyt viestiä tilaajalle. Hyökkäyksen parametreista hyötykuorman koko piti rajoittaa 240 Mt:n, koska sitä suuremmilla arvoilla Mosquitto-ohjelmaa pyörittänyt kontti kaatui. Tämä oli ohjelman suorittamista hyökkäyksistä selkeästi raskain ja käytännössä hyökkäys kulutti käy-

tetylle virtuaalikoneelle allokoidun muistin kokonaisuudessaan ja prosessoritehoakin runsaasti, joka voidaan nähdä kuvassa 6.1. Muistin lisäämisellä virtuaalikoneeseen ei ollut järin suurta vaikutusta lopputulokseen, testi ajettiin uudelleen vielä 8 Gt muistilla ja sekin tuli kulutettua täyteen. Loppujen lopuksi hyökkäys jumitti




Kuva 6.1: Virtuaalikoneen resurssien tyhjentyminen

koko virtuaalikoneen melko totaalisesti. Voidaankin todeta, että palvelunestämisen näkökulmasta hyökkäys oli hyvin tehokas.

TLS-salauksen käyttöönotto esti lähtötilanteessa MQTTSA:lla toteutetun hyökkäyksen, koska ohjelma ei pystynyt ottamaan yhteyttä välittäjään. Kun ohjelmalle annettiin käyttöön salaukseen käytetty varmenne, se pystyi toteuttamaan palvelunestohyökkäykset yhtä hyvin kuin ilman salaustakin. Tämän tapauksen toteuttaminen vaatisi hyökkääjältä huomattavasti enemmän vaivaa muihin tapauksiin verrattuna. Pystyäkseen toteuttamaan hyökkäyksen, hyökkääjän tulee ensin saada varmenne haltuunsa. Kuten aiemmin esitettiin niin tämä voisi tapahtua kuitenkin jonkin valvomattoman fyysisen laitteen haltuunsaamisen kautta.

# 7 Yhteenveto ja johtopäätökset

MQTT on yksi yleisimmistä sovelluskerroksen protokollista. Se on suunniteltu tarkoituksella kevyeksi ja näin ollen se soveltuu hyvin erilaisiin kotiautomaatio ja IoT-järjestelmiin, joissa käytetään rajoitetuilla prosessointi ja tiedonsiirtokapasiteeteilla olevia laitteita. MQTT:ssä on sisäänrakennettuna melko niukasti tietoturvallisuutta lisääviä ominaisuuksia. Käytännössä asiakkaiden tunnistaminen käyttäjätunnus-salasanaparilla, asiakastunnisteella tai asiakasvarmenteilla ovat ainoat tavat tunnistaa laitteita. Asiakasvarmenteita ei tässä tutkimuksessa tarkasteltu. MQTT:tä käytettäessä tietoliikennettä ei lähtökohtaisesti salata millään ja tämä altistaa tunnukset ja viestinnän salakuuntelulle ja mahdollistaa tiedon peukaloinnin.

Tässä tutkimuksessa oli tarkoitus tutkia ja testata MQTT-protokollan tietoturvallisuuden ominaisuuksia. Teorian ja testaustapauksien avulla pyrittiin selvittämään hyökkäysmekanismeja MQTT-protokollaa vastaan ja niiden lievennyskeinoja.

Työn teoreettisessa osassa käytiin läpi ylätasolla älykoteja ja IoT-järjestelmiä sekä niiden arkkitehtuurimalleja. Tämän jälkeen käsiteltiin yleisesti tietoturvallisuutta ja tietoturvatestausta, sekä älykotien tietoturvallisuuden vaatimuksia. Älykotien ja IoT-järjestelmien sovelluskerroksen protokollista syvennyttiin MQTT-protokollaan sekä sen ominaisuuksiin, sekä MQTT-protokollaan kohdistettaviin hyökkäyksiin.

Työn empiirisessä osassa testatiin MQTT-protokollan tietoturvallisuuden ominaisuuksia testausympäristössä erilaisia hyökkäyksiä vastaan. Testaus toteutettiin virtualisoidulla Kali Linux-käyttöjärjestelmällä käyttäen siinä valmiiksi asennettuja ohjelmistoja Nmap, Metasploitable sekä Wireshark sekä yhtä valitun käyttöjärjestelmän ulkopuolista ohjelmaa MQTTSA:ta. Kali Linuxilla käytettiin Docker-konttialustaa, jossa suoritettiin Eclipse Mosquitto MQTT-välittäjää, johon valitut hyökkäystavat kohdistettiin. Hyökkäykset toteutettiin ensin salaamatonta liikennettä käyttävää järjestelmää vastaan ja tämän tämän jälkeen TLS 1.3-salausta käyttävää järjestelmää vastaan.

Teoreettisen taustan ja empiirisen testauksen perusteella tehokas tapa suojata MQTT-protokollaa käyttävää järjestelmää on tietoliikenteen suojaaminen TLS-salauksella. Testattuja hyökkäystyyppejä vastaan TLS-salaus toimi kaikissa tämän tutkimuksen testitapauksissa. Testauksessa kuitenkin havaittiin, että varmenteiden tur-

vallisuudesta tulee kuitenkin pitää huolta. Jos salaukseen käytetty varmenne päätyy hyökkääjälle esimerkiksi jonkin haltuunotetun fyysisen laitteen kautta, mahdollistaa se järjestelmän normaaliin toimintaan vaikuttamisen ja tietoturvallisuuden vaarantamisen.

Tutkimuksessa todettiin TLS-salauksen tehokkuus MQTT:n tietoturvallisuuden parantamiseen. Salauksen käyttö kuitenkin lisää ns. overheadia liikenteeseen. Jatkotutkimusaiheena voitaisiin tutkia tämän resurssitarpeen eroja, kun käytetään TLS-salausta ja kun käytetään X509. asiakasvarmenteita. Myös asiakasvarmenteen vaikutusta viestiliikenteen turvallisuuteen voitaisiin testata vastaavan tapaisissa testaustapauksissa kuin tässäkin tutkimuksessa.

# Lähteet


[1] AGARWAL, S., JAIN, S., JA KUMAR, A. GUI Docker implementation: Run common graphics user applications inside Docker container. Julkaisusarjassa *2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART)* (2021), 424–427.

[2] ALGHAMDI, K., ALQAZZAZ, A., LIU, A., JA MING, H. IoTVerif: An automated tool to verify SSL/TLS certificate validation in android MQTT client applications. Julkaisusarjassa *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (New York, NY, USA, 2018), CODASPY '18, Association for Computing Machinery, 95102.

[3] ANDY, S., RAHARDJO, B., JA HANINDHITO, B. Attack scenarios and security analysis of MQTT communication protocol in IoT system. Julkaisusarjassa *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)* (2017), 1–6.

[4] BAIRWA, S., MEWARA, B., JA GAJRANI, J. Vulnerability scanners-a proactive approach to assess web application security. *International Journal on Computational Science & Applications 4* (03 2014).

[5] CHEN, F., HUO, Y., ZHU, J., JA FAN, D. A review on the study on MQTT security challenge. Julkaisusarjassa *2020 IEEE International Conference on Smart Cloud (SmartCloud)* (2020), 128–133.

[6] CHEN, L., LIU, J., XIAN, M., JA WANG, H. Docker container log collection and analysis system based on ELK. Julkaisusarjassa *2020 International Conference on Computer Information and Big Data Applications (CIBDA)* (2020), 317–320.

[7] COSTIN, A., ZADDACH, J., FRANCILLON, A., JA BALZAROTTI, D. A large scale analysis of the security of embedded firmwares. Julkaisusarjassa *USENIX Security 2014, 23rd USENIX Security Symposium, August 20-22, 2014, San Diego, USA* (San Diego, USA, 2014), Usenix, Ed.

[8] DELIJA, D., PETROVI, ., SIROVATKA, G., JA AGAR, M. An analysis of wireless network security test results provided by Raspberry Pi devices on Kali Linux. Julkaisusarjassa *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)* (2021), 1219–1223.

[9] DIMOV, V., KIRDAN, E., JA PAHL, M.-O. Resource tradeoffs for TLS-secured MQTT-based IoT management. Julkaisusarjassa *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium* (2022), 1–6.

[10] DORSEMAINE, B., GAULIER, J.-P., WARY, J.-P., KHEIR, N., JA URIEN, P. A new approach to investigate IoT threats based on a four layer model. Julkaisusarjassa *2016 13th International Conference on New Technologies for Distributed Systems (NOTERE)* (2016), 1–6.

[11] ECLIPSE. Mosquitto server shutdown attack, 2018. URL `https://bugs.eclipse.org/bugs/show_bug.cgi?id=529754`, `viitattu22.1.2023`.

[12] F-SECURE. Mitä on kyberturvallisuus?, 2022. URL `https://www.f-secure.com/fi/home/articles/what-is-cyber-security`, `viitattu22.1.2023`.

[13] FRILANDER, S. Palvelunestohyökkäykset ja kuinka Kali Linux pystyy niihin vastaamaan? Pro Gradu, Jyväskylän Yliopisto, 2022.

[14] GHIRARDELLO, K., MAPLE, C., NG, D., JA KEARNEY, P. Cyber security of smart homes: Development of a reference architecture for attack surface analysis. Julkaisusarjassa *Living in the Internet of Things: Cybersecurity of the IoT - 2018* (London, UK, 2018), 1–10.

[15] HAN, J.-H., JEON, Y., JA KIM, J. Security considerations for secure and trustworthy smart home system in the IoT environment. Julkaisusarjassa *2015 International Conference on Information and Communication Technology Convergence* (Jeju, South Korea, 2015), 1116–1118.

[16] HARSHA, M. S., BHAVANI, B. M., JA KUNDHAVAI, K. Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs. Julkaisusarjassa *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (2018), 2244–2250.

[17] HIVEMQ. Last will and testament - MQTT essentials: Part 9. URL https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament/, viitattu 26.2.2023.

[18] HIVEMQ. MQTT security fundamentals. URL https://www.hivemq.com/blog/mqtt-security-fundamentals-authentication-username-password/, viitattu 28.12.2022.

[19] HIVEMQ. Quality of Service (QoS) 0,1, & 2 MQTT essentials: Part 6. URL https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/, viitattu 26.2.2023.

[20] HIVEMQ. TLS/SSL - MQTT security fundamentals. URL https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/, viitattu 12.3.2023.

[21] IMPERVA. What is Metasploit? URL https://www.imperva.com/learn/application-security/metasploit/, viitattu 25.2.2023.

[22] INFOSECMATTER. MQTT authentication scanner - metasploit, 2021. URL https://www.infosecmatter.com/metasploit-module-library/?mm=auxiliary/scanner/mqtt/connect, viitattu 22.1.2023.

[23] IYER, S., BANSOD, G. V., PRAVEEN, N. V., JA GARG, S. Implementation and evaluation of lightweight ciphers in MQTT environment. Julkaisusarjassa *2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)* (2018), 276–281.

[24] JUSSILA, J. Esineiden internetin turvallisuusuhat ja protokollat. Diplomityö, Oulun Yliopisto, 2022.

[25] KALI LINUX. What is Kali Linux? URL https://www.kali.org/docs/introduction/what-is-kali-linux/, viitattu 11.3.2023.

[26] KOTIMAISTEN KIELTEN KESKUS. Kotimaisten kielten keskuksen nykysuomen sanalista, 2007. URL https://kaino.kotus.fi/sanat/nykysuomi/, viitattu 25.2.2023.

[27] LESJAK, C., HEIN, D., HOFMANN, M., MARITSCH, M., ALDRIAN, A., PRILLER, P., EBNER, T., RUPRECHTER, T., JA PREGARTNER, G. Securing smart

maintenance services: Hardware-security and TLS for MQTT. Julkaisusarjassa *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)* (2015), 1243–1250.

[28] LI, M., GU, W., CHEN, W., HE, Y., WU, Y., JA ZHANG, Y. Smart home: Architecture, technologies and systems. *Procedia Computer Science 131* (01 2018), 393–400.

[29] LIU, S. Mac spoofing attack detection based on physical layer characteristics in wireless networks. Julkaisusarjassa *2019 IEEE International Conference on Computational Electromagnetics (ICCEM)* (2019), 1–3.

[30] LÓPEZ DE JIMÉNEZ, R. E. Pentesting on web applications using ethical - hacking. Julkaisusarjassa *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI)* (2016), 1–6.

[31] MAHENDRA, M., JA PRABHA, P. S. Classification of security levels to enhance the data sharing transmissions using blowfish algorithm in comparison with data encryption standard. Julkaisusarjassa *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)* (2022), 1154–1160.

[32] METASPLOIT. Metasploit modules. URL `https://docs.metasploit.com/docs/modules.html`, `viitattu11.3.2023`.

[33] MICROSOFT. The stride threat model. URL `https://learn.microsoft.com/en-us/previous-versions/commerce-server/ee823878(v=cs.20)?redirectedfrom=MSDN`, viitattu 14.12.2022.

[34] MQTT.ORG. MQTT publish / subscribe architecture, 2021. URL `https://mqtt.org/`, `viitattu22.1.2023`.

[35] NKUBA, C. K., KIM, S., DIETRICH, S., JA LEE, H. Riding the IoT wave with VFuzz: Discovering security flaws in smart homes. *IEEE Access 10* (2022), 1775–1789.

[36] NMAP. Introduction. URL `https://nmap.org`, viitattu 26.2.2023.

[37] OAK, A., JA DARUWALA, R. Assessment of Message Queue Telemetry and Transport (MQTT) protocol with Symmetric Encryption. Julkaisusarjassa *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)* (Jalandhar, India, 2018), 5–8.

[38] OASIS. MQTT version 5.0. URL https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, viitattu 8.11.2022.

[39] OWASP. OWASP Testing Guide v.4. URL https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Testing_Guide_v4.pdf, viitattu 6.10.2022.

[40] PALMIERI, A., PREM, P., RANISE, S., MORELLI, U., JA AHMAD, T. MQTT-SA: A tool for automatically assisting the secure deployments of MQTT brokers. Julkaisusarjassa *2019 IEEE World Congress on Services (SERVICES)* (2019), vol. 2642-939X, 47–53.

[41] SAITTA, P., LARCOM, B., JA MICHAEL, E. Trike v.1 methodology document, 2005. URL http://www.octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf, viitattu 14.12.2022.

[42] SANASTOKESKUS. Kyberturvallisuuden sanasto, 2018. URL https://sanastokeskus.fi/tiedostot/pdf/Kyberturvallisuuden_sanasto.pdf?file=pdf/Kyberturvallisuuden_sanasto.pdf, viitattu 22.1.2023.

[43] SECURITY CAFÉ, IMOROSAN. IOT pentesting 101: How to hack MQTT the standard for IOT messaging. URL https://securitycafe.ro/2022/04/08/iot-pentesting-101-how-to-hack-mqtt-the-standard-for-iot-messaging/, viitattu 5.10.2022.

[44] SINGH, A. P., KUMAR, A., JA KUMAR, V. A study on MQTT protocol and its cyber attacks. *International Advanced Research Journal in Science, Engineering and Technology* (01 2022).

[45] SRIRAM, V. S. S., SAHOO, G., JA AGRAWAL, K. K. Detecting and eliminating rogue access points in IEEE-802.11 WLAN - a multi-agent sourcing methodology. Julkaisusarjassa *2010 IEEE 2nd International Advance Computing Conference (IACC)* (2010), 256–260.

[46] SYED, N. F., BAIG, Z., IBRAHIM, A., JA VALLI, C. Denial of service attack detection through machine learning for the IoT. *Journal of Information and Telecommunication 4*, 4 (2020), 482–503.

[47] UPADHYAY, Y., BOROLE, A., JA DILEEPAN, D. MQTT based secured home automation system. Julkaisusarjassa *2016 Symposium on Colossal Data Analysis and Networking (CDAN)* (2016), 1–4.

[48] UR REHMAN, S., JA GRUHN, V. An approach to secure smart homes in cyber-physical systems/internet-of-things. Julkaisusarjassa *2018 Fifth International Conference on Software Defined Systems* (Barcelona, Spain, 2018), 126–129.

[49] VARSHNEY, T., SHARMA, N., KAUSHIK, I., JA BHUSHAN, B. Architectural model of security threats & their countermeasures in IoT. Julkaisusarjassa *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)* (2019), 424–429.

[50] VASSEUR, J.-P., JA DUNKELS, A. *Interconnecting Smart Objects with IP : The Next Internet.* Morgan Kaufmann, Burlington, MA, 2010.

[51] VERSPRITE. What is pasta threat modeling? URL `https://versprite.com/blog/what-is-pasta-threat-modeling/`, viitattu 14.12.2022.

[52] WIRESHARK. Wireshark frequently asked questions. URL `https://www.wireshark.org/about.html`, viitattu 11.3.2023.

[53] YADAV, G., ALLAKANY, A., KUMAR, V., PAUL, K., JA OKAMURA, K. Penetration testing framework for IoT. Julkaisusarjassa *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)* (2019), 477–482.

[54] YALCINKAYA, F., AYDLEK, H., ERTEN, M., JA NANÇ, N. Iot based smart home testbed using MQTT communication protocol. *Uluslararas Muhendislik Arastirma ve Gelistirme Dergisi* (01 2020), 317.