TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

OSSI PERÄ

AR DRONEN KAAPPAUS JA TURVALLISUUS

Kandidaatintyö

Tarkastaja: Seppälä Jari.
Tarkastaja ja aihe hyväksytty
28.8.2017

TIIVISTELMÄ

OSSI PERÄ: Otsikko
Tampereen teknillinen yliopisto
Kandityö, 23 sivua, 2 liitesivua
Helmikuu 2018
Automaatiotekniikan koulutusohjelma
Pääaine: Automaatiotekniikka
Tarkastajat: Seppälä Jari.
Avainsanat: Esineiden internet, lennokki, tietoturva, automaatio

Tietoturva on ollut enenevässä määrin esillä moderneissa ohjelmistoprojekteissa. Etenkin automaation aloilla on enemmän herätty miettimään, mitä uhkia ja mahdollisuuksia etenevä teknologia tuo mukanaan ja miten niitä voitaisiin välttää tai hyödyntää. Tämän työn tarkoitus on demonstroida, millaisia seurauksia voi syntyä tietoturvan puutteellisuudesta ja sen väärinkäytöstä esineiden internetin sovelluksessa. Lisäksi työ pyrkii etsimään ratkaisuja kyseisen käyttökohteen turvaamiseksi.

Työssä tarkoituksena on kaapata toisen käyttäjän hallussa oleva Parrot AR Drone v2. Laitteen kaappaus suoritetaan Raspberry Pi 2:lla, johon on yhdistetty WLAN-adapteri, joka tukee paketti-injektointia. Tämän lisäksi työssä selvitetään, millaisella ympäristöllä kaappaus pystytään suorittamaan. Työssä kaappauksen toteuttamiseen käytetään valmista koodipohjaa ja sitä muokataan työn ympäristöön sopivaksi. Sen lisäksi koodipohjan toimintaan perehdytään huolellisesti, koska se määrittää miten ja miksi hyökkäys toimii. Tämän jälkeen analysoidaan lennokin tietoturvaongelmia ja selvitetään mahdollisia ratkaisuja löydetyille vioille.

Työssä todettiin, että kyseisen laitteen kaappaus on mahdollista toteuttaa kunhan käytössä on paketti-injektointia tukeva WLAN-adapteri, sekä adapterin kanssa yhteensopiva tietokone. Työssä lennokin suurimpana tietoturvaongelmana pidettiin autentikoinnin puutetta. Tästä ongelmasta johtuu täydellinen hallinnan menetys. Tähän ratkaisuna ehdotettiin muun muassa autentikoinnin lisäämistä. Työn tulokseksi todettiin, että tietoturvan puute voi aiheuttaa täydellisen hallinnan menetyksen. Tämän laitteen tapauksessa hyökkääjä ei saavuta mitään hyödyllistä laitteen kaappaamisesta, mutta työ toimii erinomaisena opettavana esimerkkinä laajempaa tapausta varten.

SISÄLLYS

KUVALUETTELO

OHJELMALUETTELO

LYHENTEET JA MERKINNÄT

apt	engl. Advanced package tool, Paketinhallinta ohjelma.
IP address	engl. Internet protocol address, Internet protokollaa käyttävien laitteiden tunniste numerosarja.
MAC	engl. Media access control, verkkorajapinnan yksilöllinen tunniste
npm	engl. Node package manager, Node ympäristön ohjelmien paketinhallinta
regex	engl. regular expression, merkkijonohahmojen tulkkaamiseen käytettävä kirjasto
REPL	engl. Read evaluate print -loop, vuorovaikutteinen silmukka ohjelmassa, jossa käyttäjältä otetaan vastaan komentoja, jotka evaluoidaan, jonka jälkeen tulostetaan tapahtunut toiminta.
SSH	engl. Secure shell, salatun tietoliikenteen protokolla
tar	engl. Tape archive, tiedon paketointi ohjelma
UML	engl. Unified modeling language, standardoitu graafinen mallinnuskieli
URL	engl. Uniform resource locator, määrittää resurssin sijainnin internetissä
USB	engl. Universal serial bus, liitäntä standardi
WEP	engl. Wired equivalent privacy, langattoman verkon salaus protokolla
WLAN	engl. Wireless local area network, langaton lähiverkko
WPA	engl. Wi-fi protected access, langattoman yhteyden turvaamiseen käytettävä protokolla

1. JOHDANTO

Tietototurva on noussut enenevässä määrin puheenaiheeksi teknologian kehityksen myötä. Kun informaatiota siirretään aiempaa enemmän pilvipalveluihin ja digitaalisesti saatavilla oleviin paikkoihin, on sitä havittelevalla taholla aivan uudenlaiset keinot hakea tietoa. Tämä tarkoittaa väistämättä myös sitä, että niillä jotka haluavat tämän informaation luvatta haltuunsa, on aivan uudenlaiset menetelmät ja työkalut sen hankkimiseen.

Automaation näkökulmasta tietoturva on erityisessä asemassa. Etenkin esineiden internetin kasvun myötä, kun kaikki laitteet voidaan laittaa kommunikoimaan saman pilven kautta, tietovuodolla voi olla erittäin merkittävät seuraukset. Tilanne on vielä kriittisempi, jos automaation pettämisestä voi seurata ihmishenkien menettäminen. Automaation tietoturvan murtamisen klassisina esimerkkeinä on Ukrainan sähkönjakeluun tehty hyökkäys "BlackEnergy Malware"[1], sekä Iranin ydinvoimalan rikastamon rikkomiseen luotu Stuxnet [10].

Tämän työn tavoitteena on demonstroida ja luoda demo-ympäristö siitä, kuinka yksinkertaisesta tietoturvariskistä voi seurata täydellinen ohjauksen menetys. Lisäksi työ demonstroi pienessä mittakaavassa, mitä seurauksia tällaisesta ohjauksen menetyksestä voi syntyä. Työssä käydään läpi, miten käytännössä kaapataan toisen käyttäjän hallinnassa oleva Parrot AR Drone v2. Kaappauksen tekemiseen käytetään valmiiksi suunniteltua ja kirjoitettua ohjelmaa, jota muutetaan hieman työn käyttöön toimivaksi. Alkuperäisen ohjelman kirjoittaja oli Kamkar Samy [16].

Työ aloitetaan käymällä läpi, millaisessa ympäristössä kaappaus on mahdollista tehdä. Tämän jälkeen selvitetään, miten kaappauksen suorittava ohjelma toimii ja lopuksi analysoidaan lennokin tietoturvaongelmaa, käydään läpi sen ongelman seurauksia ja arvioidaan, miten ongelman voisi korjata.

2. YMPÄRISTÖ

Ympäristöllä tarkoitetaan kokonaisuutta, jonka jokin ohjelma tarvitsee käyttöönsä toimiakseen oikein. Ympäristöön kuuluu laitteisto käyttöjärjestelmä ja lähes aina joukko pieniä apuohjelmia. Tässä luvussa käydään läpi, miten ympäristö valmistellaan kaappausta varten. Lisäksi sivutaan mahdollisia ongelmia, joita valmistelun aikana voi ilmetä.

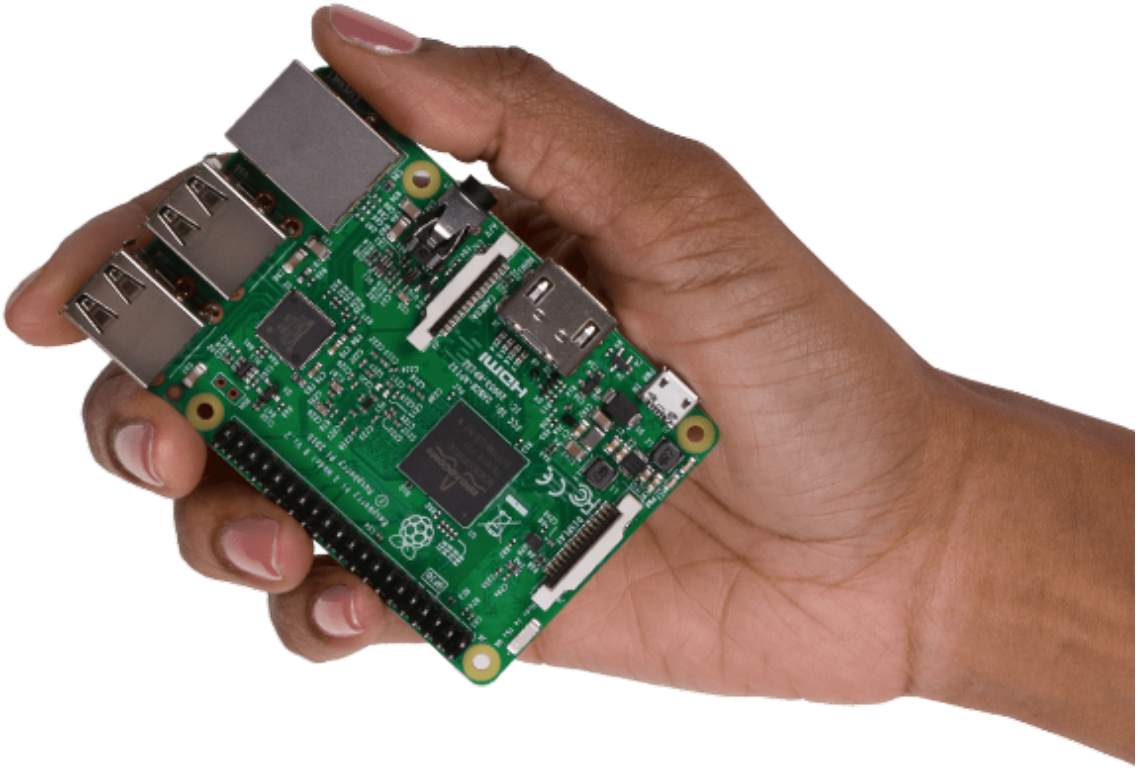
2.1 Tarvittava laitteisto

Kaappaamisen liittyy paljon erilaista laitteistoa, jossa jokaisella laitteella on oma tehtävänsä. Kamkar [16] listasi ohjeistuksessaan laitteiston, jota itse käytti kaappauksen suorittamiseen. Tätä listaa hyödyntämällä voidaan yleistää, millaisella kokonaisuudella kaappaus voidaan suorittaa. Seuraava lista sisältää laitteiston, mikä kaappaamiseen vaaditaan:

- tietokone
- WLAN-adapteri
- Parrot AR Drone
- älypuhelin tai tabletti
- virtalähde.

Vaatimuksena tietokoneelle on se, että siihen on pystyttävä ohjelmoimaan. Suositeltavaa on kuitenkin, että siihen pystytään asentamaan jokin Linux-käyttöjärjestelmä. Tämä sen takia, että Linuxille on ohjelmoitu valmiiksi kaikki tarpeellinen, joten sen kautta toimiminen säästää aikaa ohjelmoinnilta. Lisäksi Linuxilla on hyvät edellytykset jatkokehitystä varten edellä mainitun syyn takia. Lisäksi tietokoneen täytyy olla kevyt. Kokeellisessa tutkimuksessa todettiin, että AR Drone, ilman styrox-runkoa, pystyy kantamaan korkeintaan noin puolen kilon verran ulkoista kuormaa [2]. Tämän tiedon tukena on S. Kamkarin toteamus, jossa hän väittää, että AR Drone pystyy kantamaan noin 400 g painon [16]. Käytännössä siis sopivia tietokoneita ovat pienet, yhden piirilevyn tietokoneet, kuten Raspberry Pi tai Arduino. Tällaiset tietokoneet ovat pieniä ja kevyitä, eivätkä siten ole lennokin kannalta liian painavia

kuljettaa mukana. Seuraavassa kuvassa työssä tietokoneena käytetty Raspberry Pi 2.1.



Kuva 2.1 *Raspberry Pi [6]*

Tietokoneessa täytyy olla joko integroitu WLAN-adapteri tai USB-portti WLAN-adapteria varten. WLAN-adaptereita täytyy olla vähintään yksi, mutta kahden WLAN-adapterin omistaminen mahdollistaa kaapatun lennokin kamerakuvan seuraamisen [5]. Mikä tahansa WLAN-adapteri ei kuitenkaan toimi, vaan sen täytyy tukea pakettiliikenteen seuraamista (Monitor mode). Tämän lisäksi sen pitää tukea paketti-injektointia, eli WLANin avulla täytyy pystyä lähettämään itseluotuja paketteja, jotka näyttävät kuuluvan jonkin jo luodun yhteyden liikenteeseen [8].

Parrot AR Drone eli itse lennokki on tarpeellinen kaappausta varten. Toisaalta, jos tavoitteena on kaapata yhdellä lennokilla toinen lennokki, niin on syytä omistaa kaksi lennokkia. Kaapattavan lennokin tulee olla Parrot AR Drone versio 1 tai 2 [16]. Seuraavassa kuvassa työssä käytettävä lennokki AR Drone V2 2.2.

Kuva 2.2 *Parro AR Drone V2 [15].*

Älypuhelin tai tabletti toimii käyttöliittymänä lennokin lentämistä varten. Älylaitteen tulisi olla ainakin sen verran moderni, että sille pystyy lataamaan ja sillä pystyy käyttämään lennokin lentämiseen suunniteltua ohjelmaa AR.Freeflight. Lennokin ohjaaminen on mahdollista myös Windows-tietokoneelta [14].

Virtalähteen tehtävä on tuottaa virtaa tietokoneelle. Virtalähteen tulisi tuottaa tarpeeksi virtaa, että tietokone pystyy sen avulla toimimaan normaalisti. Lisäksi jos virtalähteen on tarkoitus kulkea lennokin mukana, se ei saa olla liian painava. Kamkar [16] suositteli omassa demonstraatiossaan käytettäväksi 1 000 mAh:n akkua, sillä siinä on tarpeeksi virtaa ylläpitämään ainakin Raspberry Pitä [16]. Tätä työtä tehtäessä virtalähteenä käytettiin verkkovirtaa.

2.2 Ympäristö tietokoneella

Kaappaukseen käytettävä ympäristö voi vaihdella huomattavasti. Seuraavaksi käydään läpi työhön käytetty laitteisto ja sen ympäristön alustus tietokoneelle.

Työssä tietokoneena käytettiin Raspberry Pi 2 Model B:tä, jota ohjattiin Windows-tietokoneelta SSH-yhteyden avulla. Yhteys kulki langallisen lähiverkon kautta, joten sillä ei ollut mitään vaikutusta työn tuloksiin. Työhön tarkoitetulla Raspberry Pillä oli valmiiksi asennettuna Raspbian Linux -käyttöjärjestelmä, joten käyttöjärjestelmän asentamiseen ei perehdytä tarkemmin. Kaappaamista varten käytetty WLAN-adapteri oli Ralink RT5370. Seurantaan käytettävää erillistä WLAN-adapteria ei ollut.

Ympäristön asentaminen tehtiin pääosin paketinhallintaohjelmilla. Paketinhallinnan tarkoitus on antaa käyttäjälle yksinkertainen käyttöliittymä, jonka avulla käyttäjä voi ladata ja asentaa itselleen työkaluja ja apuohjelmia. Ne myös tarvitsevat usein toimiakseen joitain muita työkaluja ja apuohjelmia. Yksi paketinhallinnan tehtävistä onkin huolehtia siitä, että myös ohjelman vaatimat muut apuohjelmat tulevat ladattua. Yleensä Linux-käyttöjärjestelmän mukana tulee valmiina jokin paketinhallinta, jolla ympäristön asennuksen voi suorittaa. Tässä työssä käytössä olevassa Raspbian Linuxissa mukana tullut paketinhallinta on apt.

Työtä tehtäessä kaikkia ohjelmia ei kuitenkaan pystytä asentamaan paketinhallinnan avulla, vaan jotkut ohjelmat joudutaan hakemaan muista lähteistä. Muista lähteistä haetut ohjelmat ovat lähes väistämättä paketoituna jotenkin. Ohjelman paketointi tarkoittaa sitä, että alkuperäisistä tiedostoista koostetaan helposti siirrettävissä oleva tiedosto. Tiedosto voi paketointiohjelmasta riippuen olla myös pakattuna pienempään tilaan. Tällä tavoin paketoitu data ei siis ole sellaisenaan luettava, vaan paketointi pitää purkaa ennen kuin sitä voi käyttää. Tiedoston paketoinnin ja paketin purkamisen voi kuitenkin suorittaa valmiiksi asennetuilla ohjelmilla. Työssä tällaiset toimenpiteet tehtiin tar-nimisellä ohjelmalla.

Linuxilla lennokin ohjaamiseen suunniteltu ja myöskin kaappaamisessa tukeuduttu node-ar-drone on kirjoitettu Node.js-ympäristöön. Node.js on palvelimen puolella toimiva Javascript-ympäristö. Se on pääosin C:llä ja C++:lla toteutettu ympäristö, joka on suunniteltu olemaan suorituksen ja muistin käytön kannalta tehokas ympäristö. Node on kehitetty tukemaan kauan suorituksessa olevia palvelinprosesseja. [20, s. 1] Node-ympäristöön suunnitelluille ohjelmille ja apuohjelmille on käytössä oma paketinhallinta nimeltä npm (Node package manager). Npm:n voi asentaa Linuxin mukana tulleella paketinhallinnalla, mutta työtä suoritettaessa tuli tällä lähestymistavalla ongelmia. Apt:n asentaman npm:n ja Node.js:n versiot ovat liian vanhoja. Apt:n käyttämä oletuslähde npm:lle ja Node.js:lle on vanhentunut. Tämän seurauksena npm:llä ei käytännössä voi ladata mitään paketteja, sillä minkä tahansa paketin lataamisyrityksen yhteydessä npm yrittää asentaa simple-debug-nimisen työkalun. Tämän vuoksi paketin latausyrityksessä tulee aina seuraava virheilmoitus:

```
1 npm http GET https://registry.npmjs.org/simple-debug
2
3 npm ERR! Error: failed to fetch from registry: simple-debug
4 npm ERR!     at /usr/share/npm/lib/utils/npm-registry-client/get.
      js:139:12
5 npm ERR!     at cb (/usr/share/npm/lib/utils/npm-registry-client/
      request.js:31:9)
6 ...
```

Ohjelma 2.1 *Npm:n vanhentuneella versiolla latausyrityksessä tuleva virheilmoitus.*

Ohjelmassa 2.1 rivillä 1 olevan URL:n viimeinen osa "simple-debug"on sen apuohjelman nimi, jota yritettiin asentaa. Tässä demonstroinnin vuoksi yritettiin asentaa sitä kyseistä pakettia, minkä latauksessa tulee ongelmia. Kyseessä voisi siis olla mikä tahansa paketti, jota yritetään asentaa, ja tuloksena tulisi sama virheilmoitus, mikä alkaa riviltä 4. Virheilmoituksen kokonaispituus on 28 riviä, mutta sen sisältö ei ole oleellista, sillä virheen syy on jo tiedossa. Virheilmoituksen lopussa käydään läpi pinoselvitys (stack trace). Siitä selviää, miten ohjelma etenee, ennen kuin se joutuu kyseiseen virhetilanteeseen. Pinoselvityksen alun näkee ohjelmasta 2.1 riviltä 5 eteenpäin. Pinoselvityksen jälkeen ohjelma kertoo käytettävän järjestelmän olennaisten osien versionumerot, kuten Linuxin, npm:n ja Noden.

Ongelman ratkaisuun on useita vaihtoehtoja, kuten ympäristön kääntäminen suoraan lähdekoodista tai toisen lähteen antaminen apt:lle, josta se lataisi npm:n. Työtä suoritettaessa ongelma ratkaistiin lataamalla valmiiksi käännetty versio seuraavan komentosarjan mukaisesti:

```
1 wget https://nodejs.org/dist/v4.2.4/node-v4.2.4-linux-armv6l.tar.
      gz
2 sudo mv node-v4.2.4-linux-armv6l.tar.gz /opt
3 cd /opt
4 sudo tar -xzf node-v4.2.4-linux-armv6l.tar.gz
5 sudo mv node-v4.2.4-linux-armv6l nodejs
6 sudo rm node-v4.2.4-linux-armv6l.tar.gz
7 sudo ln -s /opt/nodejs/bin/node /usr/bin/node
8 sudo ln -s /opt/nodejs/bin/npm /usr/bin/npm
```

Ohjelma 2.2 *Node-ympäristön asentaminen [9].*

Ohjelmassa 2.2 ensimmäisellä rivillä ladataan ohjelma pakattuna Node.js:n virallisesta jakeluosoitteesta. Seuraavaksi ladattu paketti siirretään toiseen hakemistoon, minne se puretaan rivin 4 komennolla. Tämän jälkeen purettu ohjelma siirretään uuteen hakemistoon, johon se jätetään ja pakkaustiedosto poistetaan. Viimeisillä kahdella rivillä lisätään symbolinen linkki Noden ja npm:n binäärikansiosta käyttä-

jän binääritiedostojen kansioihin. Hakemisto */usr/bin* on sellainen, mistä Linux etsii käyttäjän ajettavia ohjelmia. Symbolinen linkki on käytännössä viite jostain paikasta johonkin toiseen paikkaan. Nyt kun Linux saa esimerkiksi komennon *npm*, se etsii käyttäjän ajettavista ohjelmista npm-nimistä kansiota. Seuraavaksi se löytää symbolisen linkin, eli Linuxin näkökulmasta viitteen, joka vain osoittaa toiseen kohtaan järjestelmässä. Nyt viitettä seuraamalla järjestelmä löytää ajettavat tiedostot.

Kun paketinhallintaohjelmat on asennettu ja ne toimivat, itse ohjelman ja sen ympäristön asennus voi alkaa. Kaappausohjelman saa haettua Kamkarin [16] versionhallinnasta. Yleensä Linuxissa on valmiina jokin versionhallinta ja ainakin työssä käytetyssä Raspbianissa oli Git-versionhallinta jo valmiina. Versionhallinta on työkalu, jonka tarkoitus on helpottaa projektin versioiden yhdistämistä ja ylläpitämistä. Monesti versionhallintaa käytetään myös avoimen lähdekoodin jakelupaikkana. Jos tietokoneella ei ole valmiiksi versionhallintaa, niin sellaisen saa asennettua paketinhallinnasta.

```
1 git clone https://github.com/felixge/node-ar-drone.git
2 npm install ar-drone
3 wget http://download.aircrack-ng.org/aircrack-ng-1.2-rc4.tar.gz
4 tar -zxvf aircrack-ng-1.2-rc4.tar.gz
5 cd aircrack-ng-1.2-rc4
6 make
7 make install
```

Ohjelma 2.3 *Tarvittavien ohjelmien ja apuohjelmien lataaminen ja asentaminen.*

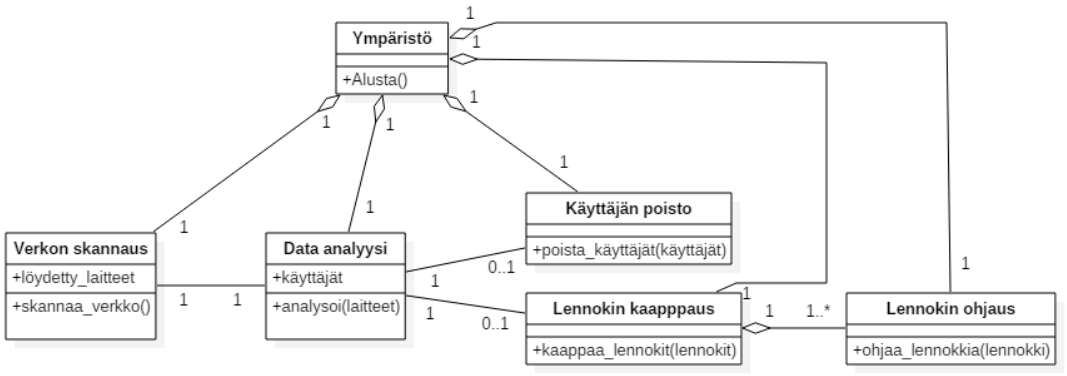
Ohjelman 2.3 komentosarjassa on listattu tarpeellisten ohjelmien ja apuohjelmien lataamiseen ja asentamiseen käytetyt komennot. Siinä ensimmäisenä haetaan kaappaamiseen käytettävä ohjelma versionhallinnasta rivillä 1. Rivillä 2 ladataan ja asennetaan kaapatun lennokin ohjaamiseen käytettävä apuohjelma. Kolmannesta rivistä eteenpäin suoritetaan kaapattavan lennokin ohjaajan poistamiseen käytettävän apuohjelman lataus ja asennus. Tämän ohjelman asentamisessa erikoista on se, että siinä ohjelma käännetään suoraan lähdekoodista. Riviltä 3 lähtien ensin ohjelma ladataan. Ladattu ohjelma tulee paketoituna, joka rivillä 4 puretaan. Tämän jälkeen *make* ja *make install* -komennoilla käsketään käyttöjärjestelmää kääntämään ohjelma ja suorittamaan ohjelman asentamiselle määritelty ohjeistus.

3. KAAPPAAVA OHJELMA

Ohjelman toiminta on keskeinen osa kaappaamista ja sen ymmärtämistä vaaditaan kaappauksen toiminnan ymmärtämistä varten. Seuraavaksi tutustutaan ohjelman rakenteeseen, joka on hyvä ymmärtää kun tutkitaan ohjelman ajamiseen liittyviä asioita, sekä ohjelman toimintaa. Tämän jälkeen tarkastellaan niitä muutoksia, joita jouduttiin tekemään, että ohjelma saatiin toimimaan työympäristössä oikein. Tämän jälkeen perehdytään ohjelman toimintaan ja lopuksi käydään läpi muita ongelmatilanteita, joita työssä tuli vastaan ohjelman johdosta.

3.1 Ohjelman rakenne

Ohjelman toiminta voidaan jakaa viidestä luokasta koostuvaksi kokonaisuudeksi kuvan 3.1 mukaisesti. Näistä luokista verkon skannaus, data-analyysi, käyttäjän poisto ja lennokin kaappaus kuuluvat yhteen komentosarjaan, joka löytyy versionhallinnasta tiedostosta *skyjack.pl* [17]. Tämän lisäksi lennokin ohjaus on täysin oma kokonaisuutensa ja se löytyy versionhallinnasta tiedostosta *drone_pwn.js* [16]. Ohjelman ympäristöllä tarkoitetaan sitä ympäristöä, jolle komentorivi komennot annetaan ja mihin ohjelman tarvitsemat muuttujat tallennetaan. On huomattavaa, että ohjelmassa ei ole olemassa oikeita luokkia, jonka vuoksi kuvan 3.1 luokkakaavion, sekä myöhemmin kuvan 3.2 sekvenssikaavion luokat ja metodit ovat ainoastaan havainnollistamista varten.



Kuva 3.1 *Komentosarjan luokkakaavio.*

Ohjelman ensimmäisen komentosarjan [17] toiminnot jakautuvat neljälle luokalle. Verkon skannaukselle, jonka tehtävä on seurata langattoman verkon liikennettä. Data-analyysille, joka verkon skannauksen saamasta datasta lennokkeja ja niiden käyttäjiä. Käyttäjän poisto, jonka tarkoitus on poistaa annetut käyttäjät tukiasemasta. Sekä viimeisenä lennokin kaappaus, jonka tarkoitus on yhdistää annettuihin lennokkeihin. Lennokin ohjaus on luokka, jonka tarkoitus on ohjata sitä lennokkia, minkä tiedot lennokin kaappaus sille antaa. Kuvassa 3.1 on lisäksi määritetty luokkien yhteydet toisiinsa, sekä yhteyksien määrät. Esimerkiksi data-analyysi kommunikoi korkeintaan yhden käyttäjän poiston kanssa ja käyttäjän poisto on aina yhteydessä yhteen data-analyysiin. Lisäksi kuvassa 3.1 lennokin kaappaus koostu yhdestä tai useammasta lennokin ohjauksesta.

3.2 Ohjelman ajaminen

Vaikka ohjelma onkin valmiiksi kirjoitettu, se ei tarkoita, että ohjelmaa voisi vain ajaa ja kaikki toimisi välttämättä halutulla tavalla. Ohjelma on versionhallinnasta haettuna syntaksisesti oikea, eli sitä voi ajaa ongelmitta, mutta toiminnallisesti se ei ainakaan työtä tehtäessä ollut sellainen kuin haluttiin. Ohjelmaan jouduttiinkin tekemään muutamia muutoksia, jotta kaappaus saatiin onnistuneesti ja turvallisesti tehtyä.

Yleensä ohjelma on suunniteltu niin, että siinä käytetään kahta WLAN-laitetta, joista toinen suorittaa kaappauksen ja toisella voidaan seurata kaapattuja laitteita [16].

Työtä tehtäessä käytössä oli vain yksi WLAN-laite, joten ohjelma ei toiminut alunperin halutulla tavalla. Versionhallinnasta löytyvän tiedoston *skyjack.pl*-koodissa rivillä 21 ja 22 määritellään WLAN-rajapinnat, joita käytetään kaappauksessa edellä mainittuihin rooleihin [17]. Nyt kun WLAN-laitteita on vain yksi, koodissa piti muuttaa kummatkin rajapinnat käyttämään samaa WLAN-laitetta.

WLAN-laitteen nimi on oletuksena *wlan0*, jossa viimeinen numero kasvaa yhdellä aina, jos edellinen numero on jo käytetty. Tämän tarkoitus on pitää nimet uniikkeina. Käyttäjä voi kuitenkin mielivaltaisesti vaihtaa nimen johonkin muuhun uniikkiin nimeen, joten laitteen nimi ei välttämättä ole yksiselitteinen. Oman laitteen nimen voi tarkistaa esimerkiksi ajamalla Linuxilla komennon *ifconfig*, joka listaa kaikkien rajapintalaitteiden perustiedot. Työtä tehtäessä WLAN-laitteen nimi oli *wlan0* ja koodiin tehtiin seuraavat muutokset:

```
1 -my $interface  = shift || "wlan1";
2 +my $interface  = shift || "wlan0";
3 my $interface2 = shift || "wlan0";
```

Ohjelma 3.1 *Ohjelman käyttämien rajapintojen muutokset.*

Ohjelmassa 3.1 miinusmerkillä on merkitty muuttuneet rivit alkuperäisessä muodossa ja plusmerkillä on muuttuneen rivin muuttunut muoto. Ohjelmassa siis muutettiin *interface* käyttämään *wlan0*-rajapintaa kaappaamiseen ja seuraamiseen.

Rajapinnalla tarkoitetaan metodeja, jotka ohjelma tai laite tarjoaa ulkopuoliseen käyttöön. Tällaisista metodeista kerrotaan alkuvaatimukset ja loppuvaatimukset. Alkuvaatimuksilla tarkoitetaan sitä, millaisesta lähtökohdasta metodia voidaan kutsua, ja vastaavasti loppuvaatimuksilla tarkoitetaan, millaiseen lopputulokseen metodin kutsuja päätyy. Rajapinnan metodien toimintaa ei kuitenkaan määritellä tarkemmin, vaan ainoastaan luvataan saavutettava lopputulos, mikäli alkuvaatimukset täyttyvät. Näin toimitaan, koska käyttäjän ei tarvitse tietää toiminnan yksityiskohtia. Lisäksi käyttäjän ei haluta optimoivan omaa toimintaansa sen perusteella, miten rajapinnan metodi sillä hetkellä toimii. Tämä mahdollistaa metodin toiminnan täydellisen muuttamisen, kunhan alku- ja loppuehdot pysyvät samana. Tekstin kannalta pitää ymmärtää, että adapterilla tarkoitetaan fyysistä laitetta ja rajapinnalla tarkoitetaan laitteen tuomaa toiminnallisuutta.

Asiaa dokumentoimatta, ohjelma olettaa kaappaukseen käytettävän rajapinnan olevan tarkkailevassa tilassa (Monitor mode). Tarkkailutila on tila, jossa WLAN-laite kuuntelee passiivisesti liikennettä jollain kanavalla [21]. Työssä ongelmana oli se, että WLAN-rajapinta on oletuksena hallitussa tilassa (Managed mode). Hallitussa

tilassa oleva WLAN-rajapinta on valmiudessa liittyä tukiasemaan (access point) ja antamaan tarvittavat tiedot ja tunnukset yhteyden muodostamiseen [21]. Tukiasema on laite, johon WLAN-yhteys muodostetaan. Tukiasemana voi toimia esimerkiksi modeemi, tai tukiasemaksi suunniteltu erillinen laite. Työssä kaapattava lennokki toimii tukiasemana.

Verkonhallinta (Network manager) on ohjelma, jonka tarkoitus on helpottaa ja automatisoida verkkoyhteyksien hallintaa. Verkonhallinta pyrkii käyttämään langallista verkkoa, jos mahdollista, tai langatonta verkkoa, jos langallista ei ole saatavilla. Tämän lisäksi verkonhallinta tulee valmiiksi asennettuna Ubuntu-pohjaisissa Linux-käyttöjärjestelmissä. [7] Verkonhallinnan automatisointi pyrkii pitämään verkkorajapinnat hallitussa tilassa, joka oli työtä tehtäessä ongelmallinen tilanne. Hallitussa tilassa oleva WLAN-rajapinta ei kykene seuraamaan kulkevaa verkkoliikennettä. Tämän lisäksi verkonhallinta estää rajapinnan tilan muuttamista tarkkailevaan tilaan ylläpitääkseen oman toimintansa. Aircrackin mukana tullut komentosarja Airmon-ng on ohjelma, joka on suunniteltu ratkaisemaan kyseiset ongelmat. Airmon-ng toimii siten, että sille annetaan verkkorajapinta, joka halutaan tarkkailevaan tilaan. Tämän jälkeen se etsii prosesseja, jotka mahdollisesti varaavat rajapinnan käyttöönsä ja tarvittaessa lopettaa ne. [12]

```
1 # put device into monitor mode
2 +sudo($airmon,  "check", "kill");
3 sudo($ifconfig, $interface, "down");
```

Ohjelma 3.2 *Seurantatilaan vaihtaminen.*

Ohjelman 3.2 rivillä 2 oleva komento suorittaa tarkkailutilaan vaihtamisen. Komennossa oleva *check*-parametri käskee Airmon-ng:tä etsimään rajapintaa hallitsevia prosesseja ja *kill*-parametrin käskee lopettamaan kyseiset prosessit. Komento lisättiin alkuperäiseen lähdekoodiin rivin 38 jälkeen [17].

Kirjoittaessaan komentosarjaa, Kamkar [16] teki toiminnan kannalta kriittisen virheen ja oletti kaikkien lennokkien käyttävän tukiaseman oletusnimeä *ardrone* koodissa rivillä 85 [17]. Jo työtä tehtäessä tämä suunnitteluvirhe tuotti ongelmia lennokin kaappaamisessa, sillä työhön käytetyn lennokin tukiaseman nimi oli *blue*, eikä *ardrone*.

```
1 - if (/^($dev:[\w:]+),\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+(\d+),.*(ardrone
    \S+),/)
2 + if (/^($dev:[\w:]+),\s+\S+\s+\S+\s+\S+\s+\S+\s+\S+\s+(\d+),.*\s(\S+)
    ,/)
```

Ohjelma 3.3 *Muutos regexiin.*

Ohjelmassa 3.3 on kirjattu regexiin tehdyt muutokset, jolla ohjelma etsii niiden tukiasemien nimiä, jotka se tunnistaa tulevan AR Droneista. Regex on kirjasto, jonka tarkoitus on etsiä merkkijonohahmoja annetusta merkkijonosta. Tässä tapauksessa annettu data, jonka rivejä regexillä tutkitaan, on seuraavanlaista:

```
1
2 BSSID, Firsttimeseen, Lasttimeseen, channel, Speed, Privacy,
    Cipher, Authentication, Power, #beacons, #IV, LANIP, ID-
    length, ESSID, Key
3 90:03:B7:38:E7:3B, 2017-11-1309:02:37, 2017-11-1309:02:41, 1, 54,
     OPN, , , -66, 2, 66, 192.168.1.2, 4, Blue,
4 4E:9E:FF:14:A2:24, 2017-11-1309:02:38, 2017-11-1309:02:41, 4, 54,
     WPA2, CCMPTKIP, PSK, -31, 6, 0, 0.0.0.0, 9, WLANkoti,
5
6 StationMAC, Firsttimeseen, Lasttimeseen, Power, #packets, BSSID,
    ProbedESSIDs
7 C0:EE:FB:92:B7:B5, 2017-11-1309:02:37, 2017-11-1309:02:41, -78,
    66, 90:03:B7:38:E7:3B,
```

Ohjelma 3.4 *Airodumpin antama data.*

Ohjelman 3.4 tyyppisestä datasta regexillä etsitään ensin lennokille tyypillisiä MAC-osoitesta. Nämä MAC-osoitteet käydään for-silmukalla läpi ja yksittäiset MAC-osoitteet tallennetaan ohjelmassa 3.3 näkyvään muuttujaan dev. Dev sisältää vain tunnistamiseen tarvittavan osan lennokin MAC-osoitteesta ja MAC-osoitteen loppu käydään läpi \w+ osuudella, joka etsii yhtä tai useampaa kirjainta. Suuri osa merkkijonon tiedosta on epäoleellista ja ne osat käydään läpi etsimällä ensin valkoisia merkkejä \s ja tämän jälkeen ei-valkoisia merkkejä \S. Lopussa ohjelma etsii kirjaimellisesti merkkijonoa ardrone, jonka sitten tallentaa muuttujaan. Tässä lähestymistavassa ohjelma ei pysty kaappaamaan minkään muun nimistä lennokkia kuin ardrone. Tämä ongelma korjattiin tallentamalla ardronen sijaan mikä tahansa sarja ei-valkoisia merkkejä kyseisessä kohdassa. Tässäkin lähestymisessä tulee ongelmia, jos tukiaseman nimessä on valkoisia merkkejä, kuten välilyöntejä. Tämä ei kuitenkaan ole todennäköinen ongelma, sillä lennokin tukiaseman nimeä ei saa helposti muutettua sellaiseksi, että siinä olisi valkoisia merkkejä. Tämä johtuu siitä, että lennokin ohjaamiseen käytettävä AR.freeflight ei anna muuttaa nimeä sellai-

seen muotoon jossa olisi välilyöntejä. Käytännössä käyttäjä joutuisi siis käyttämään mahdollisesti paljonkin aikaa valkoisten merkkien lisäämiseksi tukiaseman nimeen.

Viimeisenä muutoksena komentosarjaan tehtiin tarkkailutilan palauttaminen hallittuun tilaan. Ohjelma oli alun perin suunniteltu siten, että hallinta lennokista otetaan toiseen WLAN-rajapintaan, mutta koska työssä oli käytössä vain yksi rajapinta, tämä ei ollut mahdollista. Nyt kun lennokista pitäisi ottaa hallinta, tulee vastaan ongelma. Tarkkailevassa tilassa oleva laite ei kykene muodostamaan yhteyttä tukiasemaan, joten laite pitää palauttaa hallittuun tilaan. Tämä toimenpide suoritetaan siten, että pysäytetään tarkkailutila, jonka jälkeen käynnistetään rajapinta uudelleen hallittuun tilaan. Kyseiset muutokset lisätään alkuperäiseen koodiin rivin 135 jälkeen [17]. seuraavasti:

```
1    print "\n\nConnecting to drone $chans{$drone}[1] ($drone)\n";
2  + sudo($airmon, "stop", $interface);
3  + sudo($ifconfig, $interface, "up");
4    sudo($iwconfig, $interface2, "essid", $chans{$drone}[1]);
```

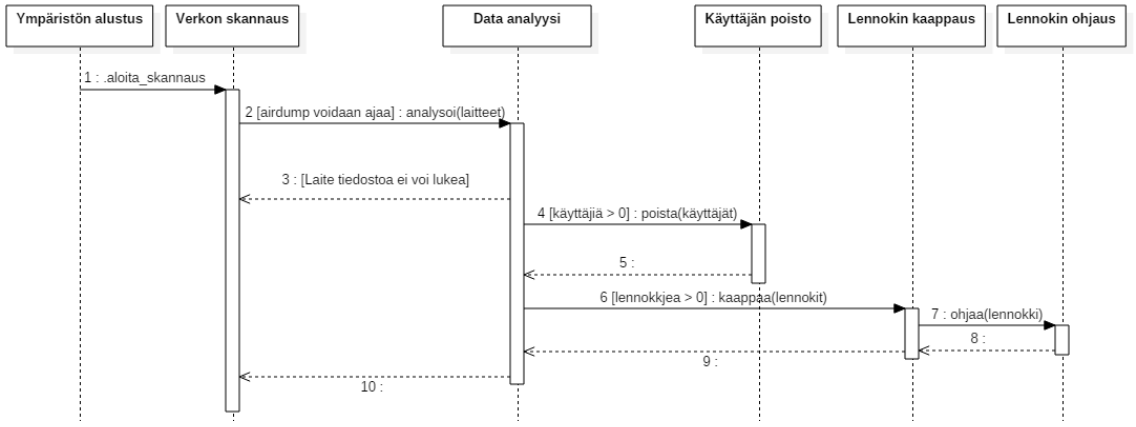
Ohjelma 3.5 *Hallintatilan palauttaminen.*

Tarkkailutilan pysäytys luotetaan sille suunnitellulle ohjelmalle airmonille ohjelman 3.5 rivin 2 mukaisesti, sekä hallintatilan käynnistys suoritetaan ifconfigilla rivin 3 mukaisesti.

Lopuksi työntekijän ja ympäristön suojaamiseksi muutettiin komentosarjaa, joka ohjasi lennokkia yhteyden muodostamisen jälkeen. Ohjelmasta poistettiin komennot, jotka ohjeistivat lennokkia lähtemään lentoon ja tekemään lennossa erilaisia asioita. Nämä komennot korvattiin erilaisilla ledien välkyttelyillä, koska niiden komentojen toimiminen demonstroi kaappaamisen onnistumista yhtä hyvin kuin lentoon lähteminen. Lisäksi niitä komentoja voi turvallisesti ajaa sisätiloissa. Tämän jälkeen poistettiin koodi, joka pyrki lähettämään videokuvaa tarkkailevalle WLAN-rajapinnalle. Tämä tehtiin sen takia, että työtä tehtäessä kyseinen koodi ajautui virhetilanteeseen, jossa se yritti ajaa Node-ympäristön metodia jota ei ollut olemassa työssä tehtävällä Node versiolla. Node-versiota yritettiin päivittää ajantasaisempaan versioon, mutta lukuisten epäonnistuneiden yritysten jälkeen työtä tehtäessä todettiin, että kuvan katsominen ei ollut osa työn suunnitelmaa, joten sen lähettäminen tai vastaanottaminen oli epäoleellista työn kannalta. Huomioitavaa on, että kaappaus ja ohjaus voitiin suorittaa onnistuneesti myös virheeseen ajautuvan koodin kanssa. Ohjelman muokkaus oli siis lähinnä ohjelman ulostulon siistimisen vuoksi. Tämän kappaleen koodi muutokset ovat dokumentoitu liitteeseen 1.

3.3 Ohjelman toiminta

Komentosarjan toiminta jakautuu neljään osaan: Ympäristön alustukseen, käyttäjän etsimiseen ja poistamiseen, lennokin etsimiseen ja kaappaamiseen, sekä kaappaamisen jälkeen suoritettavaan lennokin ohjaamiseen. Viimeiset kaksi osuutta ovat ohjelmassa toteutettu jokseenkin päällekkäin. Ohjelman toiminnan etenemistä on hahmoteltu seuraavalla sekvenssikaaviolla:



Kuva 3.2 *Komentosarjan sekvenssikaavio.*

Kuvan 3.2 sekvenssikaavio on jaettu kuudelle osakokonaisuudelle. Ympäristön alustus, verkon skannaus, data-analyysi, käyttäjän poisto, lennokin kaappaus ja lennokin ohjaus. Perinteisesti osakokonaisuudet ovat omia olioita, mutta komentosarjassa ei ole olioita, joten osakokonaisuudet ovat nimetty kuvaamaan ohjelman osakokonaisuuksien toimintaa.

Ympäristön alustuksessa ohjelma määrittelee käytettävien käyttöjärjestelmän komentojen nimet muuttujiin. Tällaisia komentoja on muun muassa dhclient, ifconfig ja airmon-ng. Tämän lisäksi komentosarja tallentaa lennokkien MAC-osoitteet muuttujaan. MAC-osoitteet ovat tunnettuja, sillä lennokin valmistaja on ostanut lennokkien käyttöön sarjan uniikkeja MAC-osoitteita, jotka ovat standardoitu olevan Parrot S.A -laitteilla. Ohjelmassa tallennetaan muuttujaan kyseisen standardin mukaiset MAC-osoitteet. Niiden avulla voidaan tunnistaa, onko jokin havaittu laite sellainen, jonka ohjelma voisi kaapata. Muuttujiin tallennetaan myös tarvittavat WLAN-rajapinnat, lennokkia ohjaavan komentosarjan ja väliaikaistiedoston polku, jonne skannattujen laitteiden tiedot sijoitetaan. Lopuksi alustetaan ympäristö tarkkailevaan tilaan ja jatketaan seuraavaan osakokonaisuuteen sekvenssikaavion kuvan 3.2 metodikutsun 1 mukaisesti.

Verkon skannaus on silmukka, joka ensin skannaa verkkoa ja lähettää saamansa informaation eteenpäin lopun ohjelman hoidettavaksi. Kun loput ohjelmasta on tehnyt omien vastuualueiden mukaiset tehtävänsä, ohjelma palaa verkon skannaus silmukkaan ja aloittaa uudestaan. Verkon skannaus yrittää skannata verkkoa jonkin aikaa ja tallentaa saamansa datan tiedostoon. Tämän jälkeen se antaa saamansa datan eteenpäin data-analyysiä varten. Edellä mainittu toiminto on sekvenssikaavion metodikutsu 2. Metodikutsun edessä hakasulkeissa oleva teksti kuvaa ehtoa, jonka pitää täyttyä, että metodia kutsutaan. Kyseinen ehto on kirjoitettu ohjelmaan varmistamaan, että ohjelmaa ajavalla käyttäjällä on pääkäyttäjän oikeudet. Tarkastus on hyvä tehdä aikaisessa vaiheessa, jotta ohjelma ajautuu mahdolliseen ongelmatilanteeseen mahdollisimman nopeasti ajamisen aloittamisesta ja ohjelman toiminta virhetilanteessa on ennustettavaa. Lisäksi tämän jälkeen kyseistä testiä ei tarvitse tehdä uudestaan, sillä loppu ajon aikana ohjelman oikeudet eivät muutu. [Kuva 3.2]

Data-analyysi on ohjelman seuraava osakokonaisuus. Tämä osakokonaisuus on toteutettu silmukka rakenteella, jonka tarkoitus on käydä läpi skannauksen tuottama data ja etsiä sieltä ensin lennokkeja ja tämän jälkeen lennokkiin yhdistäneitä laitteita. Tämä toimii siten, että ohjelma etsii regexillä sellaisia rivejä, joiden alussa on jokin lennokin MAC-osoite. Tunnistuksen jälkeen ohjelma tallentaa lennokin eli tukiaseman nimen, sekä lähetyskanavan löydettyjen lennokkien listaan. Tämän jälkeen ohjelma etsii saman rivin lopusta lennokkien tunnettuja MAC-osoitteita. Jos ohjelma löytää tällä tavalla MAC-osoitteen, niin se tallentaa rivin alussa olevan MAC-osoitteen, sekä rivin lopussa olevan MAC-osoitteen löydettyjen käyttäjien listaan. Menettely on omituinen, sillä tiedosto rakentuu ohjelman 3.4 mukaisesti siten, että yhdellä rivillä on joko tukiasema, tai johonkin yhdistetty laite. Ei kumpikin yhtä aikaa. [Kuva 3.2]

Aloittaessaan toimintansa, data-analyysi yrittää avata skannauksesta saamansa tiedoston. Jos tiedostoa ei jostain syystä pysty avaamaan, esimerkiksi siksi, että tiedostoa ei ole olemassa tai tiedostoon ei ole oikeuksia, niin ohjelma palaa verkon skannaukseen paluuviestin 3 mukaisesti. Jos data-analyysi onnistuu analysoimaan tiedoston sisällön, niin se antaa käyttäjien tiedot eteenpäin käyttäjän poistoa varten metodikustun 4 mukaisesti. käyttäjien poiston jälkeen data-analyysi antaa löydettyjen lennokkien tiedot lennokkien kaappausta varten metodikutsun 6 mukaisesti. Näiden operaatioiden jälkeen ohjelma palaa paluuviestin 10 mukaisesti. [Kuva 3.2]

Käyttäjien poisto saa löydettyjen käyttäjien MAC-osoitteet listana, jonka se käy silmukalla läpi. Aluksi käyttäjien poistossa vaihdetaan käytettävä WLAN-rajapinta samalle kanavalle missä poistettava käyttäjä on. Kanavatieto löytyy löydettyjen lennokkien listalta. Tämän jälkeen suoritetaan poiskirjaus (deauthentication) aireplayl-

la. Kyseinen operaatio vaatii WLAN-rajapinnalta tuen paketti-injektioon. Poiskirjauksessa käyttäjälle ja tukiasemalle lähetetään eroamispaketteja (dissociate), joiden jälkeen käyttäjä ja tukiasema luulevat toisen osapuolen purkaneen yhteytensä [18]. [Kuva 3.2]

Lennokin kaappaus on silmukka, joka käy löydettyjen lennokkien listaa läpi. Se tarkistaa ensin, onko kyseessä oleva lennokki kaapattu jo aiemmassa vaiheessa ja lisää lennokin sille listalle, jos ei ole. Seuraavaksi se valitsee seuraamiseen tarkoitetun WLAN-rajapinnan, jolla yrittää yhdistää laitteeseen. Tarkistamatta onnistumista ohjelma yrittää tämän jälkeen ohjata lennokkia erillisen metodikutsun 7 mukaisesti. [Kuva 3.2]

Lennokin ohjaus toimii node-ar-drone -työkalun avulla, joka on lennokin ohjaukseen kehitetty valmis rajapinta. Node-ar-drone tukee lukuisia staattisia komentoja, joiden perusteella se ohjaa lennokin toimintaa. Tällaisia komentoja on esimerkiksi ledien välkyttely, korkeuden nostaminen, tai oikealle kääntäminen. Näitä komentoja voi antaa lennokille esimerkiksi ajamalla valmiiksi kirjoitettuja komentosarjoja, samalla tavalla kuin työn ohjelmassa. Lisäksi node-ar-drone tukee REPLin avulla komentojen ajamista. REPLin (read evaluate print loop) tarkoitus on reaaliaikaisesti ottaa vastaan käyttäjän syötteitä, kääntää ja ajaa syötetty komento, tulostaa komennon ulostulo ja jäädä odottamaan seuraavaa syötettä. Tällä tavoin lennokin ohjaaminen on hyödyllistä, kun käyttäjä ei osaa ennakoida, mitä hän milloinkin haluaisi lennokin tekevän.

Lennokin ohjaus on komentosarja, jonka tarkoitus on lähettää kaapatun lennokin videokuvaa lennokkia seuraavalle WLAN-rajapinnalle. Tämän lisäksi lennokin ohjaukseen on ohjelmoitu on valmis komentosarja liikkeille, mitä se aikoo komentaa lennokin suorittamaan. Aluksi se yrittää muodostaa yhteyden lennokkiin, tämän jälkeen se poistaa mahdollisesta käyttäjän poistamisesta johtuvan hätäilmoitus viestin. Tämän jälkeen lennokin ohjaus yrittää ottaa videokuvan haltuunsa ja lähettää sen WLAN-rajapinnalle. Kuvan lähetyksen jälkeen alkaa suoritettavien liikkeiden komentosarja. Nämä komennot suoritettuaan ohjelma palaa paluuviestin 9 mukaisesti. [Kuva 3.2]

3.4 Ongelmatilanteet

Kaappauksen toteuttaminen on monimutkainen toimenpide ja siinä voi ympäristöstä riippuen olla enemmän tai vähemmän muuttujia, jotka vaikuttavat toiminnan kulkuun. Tästä johtuen ohjelman alustamisessa, ajamisessa, tai suorituksessa voi ilmetä erilaisia ongelmia. Työtä tehtäessä ympäristön erilaisuus vaikutti monilta

osin kaappauksen suoritukseen. Näistä esimerkkinä komentosarjan toiminnan muutokset. Siinä missä Kamkarin [16] demonstraatiovideo ohjelman toiminnasta sujui ilman minkäänlaisia ongelmia, työtä tehtäessä ongelmat eivät rajoittuneet ainoastaan koodin muokkaamiseen. Työssä ongelmia löytyi tämän lisäksi älylaitteen johdosta. Tässä alaluvussa käydään läpi sitä ongelmaa ja sen syytä, joka ilmeni vielä koodimuutoksien jälkeen.

Työtä tehtäessä kaappauksen suorituksen osana huomattiin ongelma, missä joskus lennokkia käyttävän laitteen poistamisen jälkeen ei onnistuttu ottamaan haltuun lennokkia. Ongelma ilmaantui kahdella eri tavalla, joista kumpaakaan tapaa ei pystynyt toistamaan jokaisella yrityksellä. Ensimmäinen ja samalla yleisempi tapa tällä virhetilanteella ilmetä, oli komennon *dhclient* käytön yhteydessä. Tässä tapauksessa, kun dhclient yritti yhdistää lennokkiin, se ei onnistunut tekemään yhteyttä, koska se ei saanut tukiasemalta eli lennokilta verkko-osoitetta (IP address). Tästä syystä *dhclient* heitti virheen: *NO DHCPOFFERS received.*. Toinen tapa tällä virhetilanteella ilmetä oli sellainen, että vaikka dhclient väitti saaneensa yhteyden lennokkiin, tämän jälkeen ajettavat ohjauskomennot eivät menneet perille. Eli työtä tehtäessä lennokki ei alkanut välkyttää ledejään. Ongelmatilanteeseen päädyttiin, kun älylaite oli yhdistettynä lennokkiin ja kaappaus yritettiin suorittaa. Työtä tehtäessä aina tähän ongelmaan ei kuitenkaan päädytty.

Kaappaamisen epäonnistumisen syynä on se, että älylaite yhdistää uudelleen lennokkiin nopeammin, kuin kaappaavan laitteen seurantaan tarkoitettu rajapinta yhdistää lennokkiin. Tämä ilmiö on selitettävissä sillä, että laitteelle tehtävän poiskirjauksen tarkoitus on pakottaa laite katkaisemaan yhteys tukiasemaan. Se ei kuitenkaan estä laitetta liittymästä tukiasemaan välittömästi uudelleen ja itseasiassa jotkin hyökkäykset luottavat siihen, että laite yrittää liittyä tukiasemaan välittömästi yhteyden katkeamisen jälkeen. [4] Teoriaa vahvisti tarkastelu, jossa seurattiin älypuhelimen yhteyden tilaa lennokkiin. Tällä tavalla tutkiessa huomattiin, että kaappauksessa älypuhelin menetti yhteytensä lennokkiin, jonka jälkeen älylaite muodosti saman tien uuden yhteyden lennokkiin. Teoriaa olisi voitu vahvistaa myös ottamalla yhteyden uudelleen muodostaminen pois päältä, mutta työtä käytettävässä laitteessa kyseistä ominaisuutta ei voitu ottaa pois päältä. Lisäksi ei ollut saatavissa laitetta, missä kyseinen toiminto olisi ollut mahdollista.

4. LENNOKIN TIETOTURVA

Lennokin tietoturva, tai lähinnä sen puute, on olennainen osa kaappauksen suoritusta. Tämän luvun tarkoitus on tarkastella, mikä tietoturvassa oli ongelmallista lennokin kannalta. Sen lisäksi käydä läpi mitä tietoturvaongelmalle voitaisiin tehdä, että sellaista ei pääsisi tapahtumaan. Lennokkiin hyökätessä hyväksikäytettiin kahta haavoittuvuutta, joita tässä luvussa analysoidaan. Toinen oli todennuksen puute ja toinen oli poiskirjaus hyökkäykselle haavoittuvuus.

4.1 Todennuksen puute

Suurin ongelma lennokin tietoturvassa oli se, että yhteyden muodostaminen lennokkiin toimi ilman minkäänlaista todentamista tai suojausta. Tämä aiheutti sen, että käyttäjän poiston jälkeen kaappaajan ei tarvitse tehdä käytännössä mitään lennokin hallinnan ottamiseksi. Kaappaajan täytyi ainoastaan yhdistää lennokkiin ja tämän jälkeen hänellä oli koko lennokin toiminnallisuus käytettävissään omaa tarkoitustaan varten. Tällaisen ongelman olisi voinut korjata yksinkertaisesti lisäämällä todennuksen lennokkiin yhteyden muodostamiseen. Suositeltava käytettävä salausprotokolla olisi WPA2, mutta esimerkiksi WPA-protokolla vahvalla salasanalla tarjoaisi tarpeeksi suojausta yhteydelle, että hyökkääjä ei pystyisi järkevässä ajassa murtamaan salausta [11].

Työssä käytetty lennokki ei tarjoa ainakaan perusominaisuuksiltaan tukea minkäänlaisen suojauksen määrittämiseksi, joten jos tällaista turvallisuutta lennokille haluaisi saada, niin sille pitäisi löytää kolmannen osapuolen toteutus, tai se pitäisi tehdä itse. Laitteen turvallisuutta lisätessä on huomattavaa, että tukiaseman SSID:n piilottaminen ei tuo lisää turvallisuutta laitteiden väliselle liikenteelle. Tämä johtuu siitä, että SSID:tä ei ole suunniteltu suojattavaksi ominaisuudeksi ja siten sen selvittäminen ei ole hankala operaatio [19]. Esimerkkinä tässä työssä poiskirjaus hyökkäyksessä käyttäjälle lähetettävä paketti kirjaa käyttäjän pois laitteesta. Tähän reagoidessa käyttäjän laite lähettää automaattisesti uuden kirjautumispyynnön tukiasemalle ja samalla paljastaa tukiaseman nimen [13].

4.2 Poiskirjaus

Toinen ongelma laitteen tietoturvassa on poiskirjauspaketit. Poiskirjauksen johdosta käyttäjälle tulee vähintään väliaikainen lennokin hallinnan menetys. Tänä aikana käyttäjä ei kykene ohjaamaan lennokkia ja hyökkääjä pystyy yhdistämään lennokkiin, kaapaten ohjauksen kokonaan. Lisäksi lennokin menetettyä yhteyden hallitsevaan käyttäjään, se asettaa itsensä hätätilaan ja lopettaa kaiken toiminnan propelleillaan tiputtaen lennokin maahan. Maahan tippumisen seurauksena voi tulla laskeutumiskulmasta riippuen materiaalisia vahinkoja.

Tällaiselta hyökkäykseltä turvautuminen on jälleen mahdollista, mutta se vaatisi edistyneempiä muutoksia tukiaseman ja käyttäjän verkonhallinnan toimintatapaan. Turvautuminen tällaisilta hyökkäyksiltä voitaisiin tehdä esimerkiksi siten, että tukiaseman tai käyttäjän laitteen saadessa poiskirjaukseen liittyviä paketteja, se jäisi odottamaan jonkin ennalta määritellyn ajan ja kuuntelisi uusien pakettien saapumista samalta laitteelta. Jos laite saisi tällä tavalla uusia paketteja poiskirjauksen jälkeen, se tietäisi kyseessä olevan poiskirjaus hyökkäyksen. Tämä johtuu siitä, että oikeassa laitteen poiskirjauksessa paketteja ei ikinä saapuisi poiskirjauksen jälkeen. Kyseistä toimenpidettä tutkivat Bellardo ja Savage testasivat kyseistä metodia, ja tutkimuksessaan totesivat sen suojaavaan hyvin verkkoyhteyttä poiskirjaus hyökkäyksiltä. He kuitenkin huomasivat tämän metodin synnyttävän uusia haavoittuvuuksia. Esimerkiksi sellaisen missä hyökkääjä voisi estää käyttäjän aidon poiskirjautumisen lähettämällä muita paketteja poiskirjautumis yrityksen jälkeen. He olivat kuitenkin sitä mieltä, että kyseiset haavoittuvuudet eivät käytännössä aiheuttaisi oikeita uhkia verkossa. [3]

Työssä käytetyllä lennokilla ei ollut mitään tämänkaltaista turvallisuus asetusta, joten sen saaminen olisi jälleen vaatinut kolmannen osapuolen toteutuksen. Tämän lisäksi työssä käytetystä älylaitteesta ei löytynyt vastaavanlaista toiminnallisuutta, joten turvallisuuden toiminnallisuutta ei pystytty työssä testaamaan kummallakaan laitteella.

5. YHTEENVETO

Työn tavoitteena oli demonstroida ja luoda demo-ympäristö siitä, kuinka yksinkertaisesta tietoturvariskistä voi seurata täydellinen ohjauksen menetys. Työn tavoitteissa onnistuttiin, sillä työssä luotiin ja dokumentoitiin demo-ympäristön, sekä sen toiminnan edellytykset. Lisäksi ympäristöllä pystyttiin kaappaamaan haavoittuva lennokki ja samalla demonstroimaan yksinkertaisesta tietoturvariskistä johtuva ohjauksen menetys.

Lennokin kaappauksen tutkiminen aloitettiin tarkastelemalla, millaisella ympäristöllä kaappaus on mahdollista suorittaa. Tässä päädyttiin siihen tulokseen, että kunhan kaappaava laite on ohjelmoitavissa ja siinä on WLAN-rajapinta, joka tukee paketti-injektiota, niin kaappaus on suoritettavissa. Lisäksi todettiin, että jos tavoitteena on yhdellä lennokilla kaapata toinen lennokki, niin tietokoneen, sekä sen virtalähteen tarvitsee olla tarpeeksi kevyitä kannettavaksi. Tämän lisäksi työssä käytiin läpi, miksi kaappaus on mahdollista suorittaa ja miten kaappaava ohjelma toimii. Lyhyesti kaappaus toimii siksi, että lennokilla ei ole minkäänlaista käyttäjä todennusta. Lisäksi hyökkäävä ohjelma hyödyntää kaappauksen suorittamiseen langattoman verkon poiskirjaus pakettien heikkoa suojausta.

Nyt kun lennokissa olevan tietoturvaongelma avulla on pystytty kaappaamaan toisen käyttäjän lennokki, niin mitä tilanteessa voi tapahtua seuraavaksi? Kaappaaja varmasti haluaa pystyä ohjaamaan lennokkia, mutta koska kaappaaja ei voi tietää lennokin tarkkaa asentoa, ei hän voi kirjoittaa staattista komentosarjaa ohjaamaan lennokkia pois paikan päältä. Tähän ongelmaan löytyy vastaus node-ar-dronessa mukana tulevasta REPL-tuesta. Sen avulla lennokille voi antaa reaaliaikaisesti komennot liikkua haluttuun suuntaa. Jos kaappaaja suorittaa kaappauksen omalla lennokillaan, niin REPLin avulla on myös helppo välittää kaappaavan lennokin liikekomennot kaapatulle lennokille.

Käytännössä kuitenkin, koska kokeellisten tutkimusten mukaan lennokin WLAN-signaali ei kanna kuin noin 15 metriä, niin työssä tehdyllä tavalla lennokin kaappaaminen ei ole mitenkään tehokasta. Työssä tutkittu kaappaus toimii paremmin opettavana esimerkkinä tietoturvan puutteen seurauksista pienessä mittakaavassa.

Tällaisen esimerkin avulla nähdään, miten suunnitteluvirheestä voi seurata katastrofaalinen hallinnan menetys ja miksi tietoturva on tärkeä huomioida suunnittelussa.

LÄHTEET

[1] BBC, Janyary 2016, Viitattu 21.10.2017, Saatavissa: http://www.bbc.com/news/technology-35297464.

[2] ARDroneShow, "Ar drone 2.0 ultraflight upgrade propeller mod - max payload & flight test," Youtube, September 2013, Viitattu 17.10.2017, Saatavissa: https://www.youtube.com/watch?v=z9zJQNpBC6Q.

[3] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions." in *USENIX security symposium*, vol. 12. Washington DC, 2003, p. 21.

[4] D.W., August 2015, Viitattu 21.10.2017,Saatavissa: https://www.aircrack-ng.org/doku.php?id=airmon-ng.

[5] Felixge, "node-ar-dron," Github, July 2012, Viitattu 21.9.2017, Saatavissa: https://github.com/felixge/node-ar-drone.

[6] R. P. Foundation, "Raspberry pi," Viitattu 10.03.2018, Saatavissa: https://www.raspberrypi.org/app/themes/mind-control/images/home-products-cta__image.png/.

[7] glen q, "Network manager," October 2016, Viitattu 21.10.2017, Saatavissa: https://help.ubuntu.com/community/NetworkManager.

[8] Q. Gu, P. Liu, S. Zhu, and C.-H. Chu, "Defending against packet injection attacks unreliable ad hoc networks," in *GLOBECOM '05. IEEE Global Telecommunications Conference, 2005.*, vol. 3, Nov 2005, pp. 5 pp.–.

[9] Y. Khachlek, Stackoverflow, October 2015, Viitattu 21.10.2017, Saatavissa: https://raspberrypi.stackexchange.com/questions/4194/getting-npm-installed-on-raspberry-pi-wheezy-image.

[10] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, no. 3, p. 49, May 2011.

[11] A. H. Lashkari, M. M. S. Danesh, and B. Samadi, "A survey on wireless security protocols (wep, wpa and wpa2/802.11i)," in *2009 2nd IEEE International Conference on Computer Science and Information Technology*, Aug 2009, pp. 48–52.

[12] mister_x, "Airmon-ng," August 2015, Viitattu 12.12.2017, Saatavissa: https://www.aircrack-ng.org/doku.php?id=airmon-ng.

[13] R. Moskowitz, "Wlan testing reports," *PSK as the Key Establishment Method, ICSA Labs*, December 2003.

[14] P. SA, "Ar.freeflight," 2014, Viitattu 10.10.2017, Saatavissa: https://www.microsoft.com/en-us/store/p/arfreeflight/9wzdncrdjq4g.

[15] ——, "Largo parrot," Viitattu 07.03.2018, Saatavissa: https://www.parrot.com/global/sites/default/files/ps/3023-large-parrot-3023jpg.jpg/.

[16] K. Samy, "Skyjack," Github, December 2013, Viitattu 28.10.2017, Saatavissa: https://github.com/samyk/skyjack.

[17] ——, "Skyjack.pl," Github, December 2013, Viitattu 1.10.2017, Saatavissa: https://github.com/samyk/skyjack/blob/master/skyjack.pl.

[18] sleek, "Deauthentication," November 2010, Viitattu 13.11.2017, Saatavissa: https://www.aircrack-ng.org/doku.php?id=deauthentication.

[19] R. Steve, "Myth vs. reality: Wireless ssids," October 2007, Viitattu 27.11.2017, Saatavissa: https://blogs.technet.microsoft.com/steriley/2007/10/16/myth-vs-reality-wireless-ssids/.

[20] S. Tilkov and S. Vinoski, "Node.js: Using javascript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80–83, Nov 2010.

[21] vias.org, "Ieee 802.11 wireless networks," January 2007, Viitattu 21.10.2017, Saatavissa: http://www.vias.org/wirelessnetw/wndw_05_04.html.

LIITE 1. DRONE_PWN.JS MUUTOKSET

```
@@ -6,14 +6,14 @@ var client = arDrone.createClient();
 client.disableEmergency();

 console.log('Connecting png stream ...');
-var pngStream = client.getPngStream();
+//var pngStream = client.getPngStream();

 var lastPng;
-pngStream
-  .on('error', console.log)
-  .on('data', function(pngBuffer) {
-    lastPng = pngBuffer;
-  });
+//pngStream
+//  .on('error', console.log)
+//  .on('data', function(pngBuffer) {
+//    lastPng = pngBuffer;
+//  });

 var server = http.createServer(function(req, res) {
   if (!lastPng) {
@@ -27,49 +27,9 @@ var server = http.createServer(function(req,
     res) {
 });

 server.listen(8080, function() {
-  console.log('Serving latest png on port 8080 ...');
-  client.takeoff();
-
-  client
-    .after(5000, function() {
-      this.clockwise(0.5);
-    })
-    .after(5000, function() {
-      this.stop();
-    })
-    .after(5000, function() {
-      this.clockwise(0.5);
-    })
-    .after(5000, function() {
-      this.stop();
-    })
-    .after(5000, function() {
-      this.clockwise(0.5);
```

```
44 -    })
45 -    .after(5000, function() {
46 -      this.stop();
47 -    })
48 -    .after(5000, function() {
49 -      this.clockwise(-0.5);
50 -    })
51 -    .after(5000, function() {
52 -      this.stop();
53 -    })
54 -    .after(5000, function() {
55 -      this.clockwise(-0.5);
56 -    })
57 -    .after(5000, function() {
58 -      this.stop();
59 -    })
60 -    .after(5000, function() {
61 -      this.clockwise(-0.5);
62 -    })
63 -    .after(5000, function() {
64 -      this.stop();
65 -    })
66 -    .after(1000, function() {
67 -      this.stop();
68 -      this.land();
69 -    });
70 -
71 +       console.log('Serving latest png on port 8080 ...');
72 +       console.log('red snake!');
73 +       client.animateLeds('redSnake', 5, 2);
74 +       console.log('Blink orange!');
75 +       client.animateLeds('blinkOrange', 10000);
76  });
```