

Jaakko Palomäki

Internet Protocol Security (IPsec) -automaation kehittäminen

Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2021



**KAMK • University
of Applied Sciences**

Tiivistelmä

Tekijä: Palomäki Jaakko

Työn nimi: Internet Protocol Security (IPsec) -automaation kehittäminen

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: IPsec, Robot Framework, tiedonsiirtoprotokollat, automaatio, Linux, Python

Opinnäytetyön toimeksiantajana toimi Bittium Wireless Oy, jonka tuotteisiin kuuluu mm. taktisen kommunikoinnin ja suojattujen yhteyksien tuotteita. Työn tavoitteena oli laajentaa useamman projektin käytössä olevaa automaatiotestauspaikan kapasiteettia sekä kehittää testausta kattavammaksi.

Automaatio on kehitetty käyttäen Robot Frameworkia, joka pohjautuu vahvasti Python-ohjelmointikieleen. Toteutus kehitettiin erityisesti testaamaan IPseciä, jonka toiminnan varmistamiseksi on olennaista OLSR- ja OSPF-reititysprotokollien toiminta. Koska aiemmin toteutettu automaatio vaati täydennyksiä tämän opinnäytetyön puitteissa, oli tarpeellista toteuttaa laajennettu Python-kirjasto, jota on mahdollista hyödyntää jatkokehitystyössä.

Robot-testit suunniteltiin toisistaan riippumattomiksi, jotta voitiin varmistaa mahdollisimman hyvä regressiotestaamisen valmiusaste automatisoitavan ominaisuuden arkaluontoisuuden vuoksi.

Abstract

Author: Palomäki Jaakko

Title of the Publication: Development of Internet Protocol Security (IPsec) automation

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: IPsec, Robot Framework, networking protocols, automation, Linux, Python

The commissioner of this Bachelor's thesis was Bittium Wireless Oy who offers a variety of products in different areas like Tactical Communications and communication security. The goal of this thesis was to extend the testing capacity of an automation test platform that was already utilized by different projects and to further develop the automation test suite used by said projects.

This automation task used Robot Framework which is developed in Python programming language. This implementation was explicitly made to test IPsec, which as a feature requires a working status for networking protocols, such as OLSR and OSPF. Because the previous automation suite required additions to support this thesis, it was necessary to implement an extended Python library to support this automation set which, in turn, can be used in the future development work.

The actual tests were designed to work independently of one another specifically to ensure the best possible basis for regression testing. This choice was deliberately made because of the vulnerability of the feature in question.

Sisällys

| | | |
|-------|---|----|
| 1 | Johdanto | 1 |
| 2 | Testaaminen | 2 |
| 2.1 | Testaamisen tasot | 2 |
| 2.2 | Käsin testaaminen | 3 |
| 2.3 | Automaatio..... | 4 |
| 3 | Reititys | 5 |
| 3.1 | Staatinen ja dynaaminen reititys | 5 |
| 3.2 | Reititysprotokollat..... | 6 |
| 3.3 | OSI-malli | 6 |
| 4 | Internet Protocol Security (IPsec)..... | 9 |
| 4.1 | Yleistä | 9 |
| 4.2 | Kuljetus- ja tunnelikäyttötapa | 10 |
| 5 | Robot Framework | 15 |
| 5.1 | Robot Framework automaatiassa | 16 |
| 5.1.1 | Työkalun käyttöönotto..... | 16 |
| 5.1.2 | Testien suunnittelu..... | 16 |
| 5.1.3 | Testien toteutus | 17 |
| 5.1.4 | Testien ajaminen..... | 19 |
| 5.1.5 | Testien tulokset..... | 19 |
| 5.1.6 | Tulosten hyödyntäminen | 21 |
| 6 | Yhteenveto ja pohdinta | 22 |
| | Lähteet | 23 |

Käsiteluettelo

| | |
|------------|---|
| API | Application programming interface eli ohjelmointirajapinta määrittelee miten ohjelmat voivat keskustella keskenään. |
| Decryption | Salauksen purku. Salauksen käänteinen operaatio, jossa tieto saatetaan takaisin luettavaan muotoon. |
| Encryption | Salaus. Tiedon pakkaamine muotoon, jossa se on turvallista kuljettaa eri laitteiden välillä. |
| Header | Tunnistetietoa paketin alussa. Käytetään välittämään dataa, jota tarvitaan paketin kuljetuksessa haluttuun osoitteeseen. |
| ICMP | Internet Control Message Protocol. Verkon virheisiin ja diagnostiikkaan käytetty protokolla. Käytetään yleisesti mm. Ping-työkalussa. |
| IPsec | Internet Protocol Security. Turvallisen tiedonsiirron mahdollistava protokolla, jota hyödynnetään mm. virtual private networkien toteutuksessa (VPN). |
| MANET | Mobile ad-hoc network eli liikkuva langaton verkko. Ominaisuuksiin luokituu itsekonfiguroituvuus ja -kontrollointi ilman erillistä tukiasemaa. |
| OLSR | Optimized Link State Routing Protocol. Reititysprotokolla, jota käytetään MANET-verkoissa. |
| OSPF | Open Shortest Path First. Reititysprotokolla, joka perustuu lyhimmän reitin laskevaan algoritmiin. |
| RFC | Request for Comments. RFC:t ovat muuttumattomia dokumentteja, joihin kirjataan internetin ominaisuuksia ylös. |
| Root cause | Juurisyys. Alkuperäinen syy sille, miksi jokin asia tai vaikutus tapahtuu. Tärkeä selvittää virheitä korjatessa. |
| TCP | Transmission Control Protocol. Yleinen tiedonsiirtoprotokolla. TCP:n tärkeimpiä ominaisuuksia ovat tiedonsiirron luotettavuus ja virheiden tunnistus. |

UDP

User Datagram Protocol. Yhteydetön tiedonsiirtoprotokolla. UDP tarjoaa yksinkertaisen protokollan laitteiden väliseen kommunikointiin, tehden siitä hyvän valinnan, jos virheiden tunnistus ei ole tarpeen tai se suoritetaan muilla tavoin kuin tiedonsiirtoprotokollalla.

1 Johdanto

Testaaminen on olennainen osa ohjelmistonkehitysprosessia, jonka aikana varmistetaan toiminnallisuuden täyttävän kaikki ne asetetut kriteerit, joita ohjelmiston tulee täyttää [1]. Järjestelmien kasvaessa ja uusien ominaisuuksien lisääntyessä on hyvin todennäköistä, että valmiissa ohjelmiston osissa tulee esiin vikoja. Jotta voidaan olla varmoja toiminnan eheydestä, tulisi testaus suorittaa uudelleen aina, kun uusia muutoksia tuodaan järjestelmään. Ohjelmistojen kasvaessa tästä muodostuu mahdoton haaste käsin suoritettavaksi, minkä vuoksi kehitetään testiautomaatiota, joka varmentaa ohjelmiston toiminnan jokaisen muutoksen yhteydessä. Tämä tekee toistuvasta ja virheille herkästä toiminnasta luotettavaa ja verrattain nopeaa. Erityisesti nyt automatisoitavaksi kohteeksi valittu IPsec on käsin testattaessa erittäin työläs ja herkkä virheille, jotka pahimmillaan voivat estää koko ohjelmiston toiminnan.

Opinnäytetyö suoritetaan osana projektityöskentelyä Bittium Wireless Oy:llä, missä jo käytössä olevaa testiautomaatioympäristöä laajennetaan kattamaan suurempaa laitekokonaisuutta sekä testattavien ohjelmisto-osuuksien määrää. Toteutus lisää merkittävästi useiden eri projektien mahdollisuutta varmentaa omien muutoksiensa toiminnallisuutta ennaltaehkäisevästi, mikä parantaa ohjelmiston laatua.

Työn aikana tärkeää on skaalautuvuus. Kaikki se, mitä lisätään järjestelmään, tulee toteuttaa tavalla, joka ei aseta uusia riippuvuuksia tulevaisuudessa. On myös tärkeää huomioda, että mikäli jatkossa tehtävälle automaatiolle on tarve tehdä kehitystyötä, se on tehtävä mahdollisimman yksinkertaiseksi. Tähän haasteeseen löytyy avaimet yrityksen ohjelmistonkehityskäytänteistä.

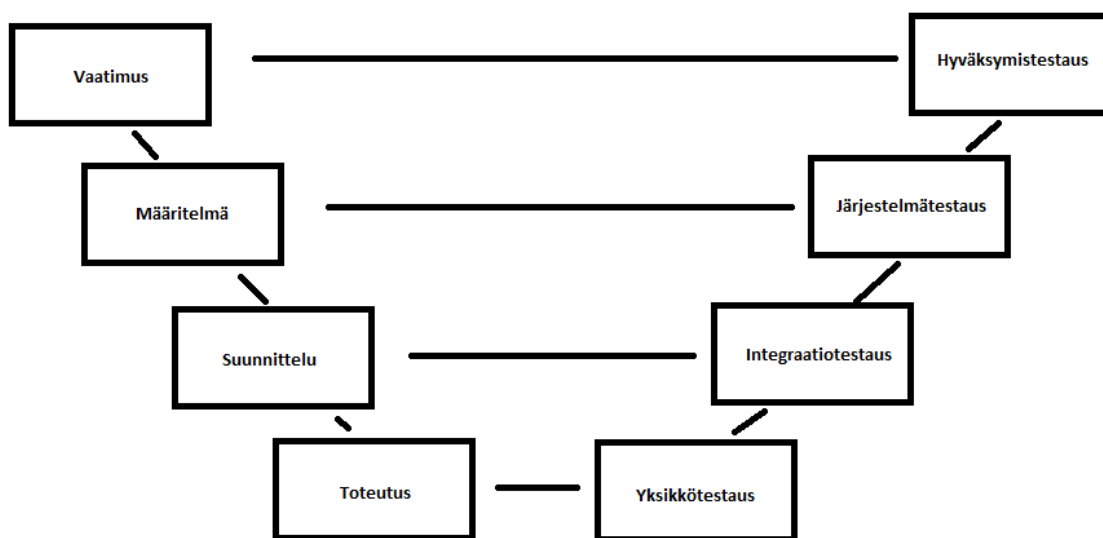
Käytännön työ automaatiolle tapahtuu käyttäen Robot Frameworkia, joka on yleisesti käytetty, geneerinen, avoimen lähdekoodin automaation toimintaympäristö [2]. Tähän aiheeseen liittyen on yrityksessä jo tehty opinnäytetyö Robot Framework reitityksen automaattisen testaamisen toteutuksessa [3], jota tässä opinnäytetyössä laajennetaan käytännön tasolle.

2 Testaaminen

Testaaminen tulisi aloittaa ajatuksella, että ohjelmisto sisältää virheitä. Jotta testaaminen voitaisiin katsoa hyödylliseksi, sen tulisi pystyä lisäämään työhön arvoa. Jos testaamista suoritetaan vain ajatuksella, että testaus osoittaa ohjelmiston toimivaksi, jätämme helposti huomiotta tätä päämäärää risteävät ongelmakohdat. On myös ongelmallista määritellä testaaminen vain demonstroimaan ohjelman toimintaa sellaisena kuin sen pitäisi olla, koska ohjelma voi toimia halutulla tavalla virheistä huolimatta. [4.]

2.1 Testaamisen tasot

Jotta mahdollisimman paljon virheitä olisi mahdollista havaita erinäisissä vaiheissa ohjelmiston testaamista, testaaminen jaetaan tyypillisesti useisiin eri testaamisen tasoihin V-mallin mukaisesti (Kuva 1).



Kuva 1. Testaustasot

Yksikkötestaus (Unit testing) on tyypillisin testaamisen muoto, jossa varmennetaan ohjelmiston yksittäisen osion toiminta oman toteutuksensa yhteydessä [1]. Tämän ensimmäisen testaamisen tason suorittaa yleensä ohjelmoija itse. Onnistunut yksikkötestaus ei vielä ota kantaa muuhun

kuin siihen, toimiiko ominaisuus tai ohjelmiston osuus muuten kuin eristettynä yksikkönä kaikesta muusta toiminnallisuudesta. Tyypillisesti yksikkötestausta voidaan suorittaa kirjoittamalla esimerkiksi erilaisia komentoja, joihin ohjelmistomoduulin täytyy reagoida halutulla tavalla.

Integraatiotestauksessa (integration testing) puolestaan kaikki yksikkötestauksessa erillisinä palikoina varmennetut ohjelmiston palaset sovitetaan yhdeksi ohjelmistokokonaisuudeksi, jota testataan yhtenäisenä järjestelmänä. Integraatiotestauksessa on tyypillistä hyödyntää niin sanottuja tynkäosioita (stub) järjestelmässä, jotka on lisätty tilkitsemään järjestelmän osien vaatimuksia, jotka eivät ole vielä valmiita.

Järjestelmätestauksessa (system testing) testataan kokonaista järjestelmää, kun kaikki palaset on saatettu valmiiksi niin, ettei integraatiotestauksessa olevia tynkiä enää hyödynnetä. Järjestelmätestaukselle ominaista on se, että vaikka testattava järjestelmä voikin olla jo lopullinen, testaaminen suoritetaan vielä testiympäristössä.

Hyväksymistestauksessa (acceptance testing) siirretään testaamisessa hyväksi todettu järjestelmä lopulliseen ympäristöönsä. Tässä vaiheessa asiakas ottaa tuotteen käyttöön ja toteaa ohjelmiston joko täyttävän asetetut vaatimukset tai olevan puutteellinen, jolloin palataan aikaisempiin testaamisen vaiheisiin riippuen puutteiden vakavuudesta.

Testaaminen on mahdollista suorittaa joko käsin (manual testing) tai automatisoidusti (automated testing). Käsin testatessa testaaja suorittaa toiminnan oikeellisuuden varmentavat toimenpiteet itse, kirjaten ylös tulokset ml. mahdolliset löytyneet virheet. Automaattisessa testauksessa testaaja laatii erinäisten työkalujen avulla tietokoneohjelman, joka suorittaa testin vaiheet (test steps) ilman ulkoisia vaikutuksia.

Edellä mainituista testitasoista kolme ensimmäistä on mahdollista ja tulisikin jossain määrin automatisoida, jotta välttyttäisiin liialliselta työltä ja kuluilta [1].

2.2 Käsin testaaminen

Käsin testaaminen tarkoittaa kaikkea sitä testaustyötä, joka tapahtuu testaajan itse tekemänä. Käsin testaamista voi, ja joskus täytyykin, tehostaa hyödyntämällä skriptejä, joilla toistuva työtehtävä voidaan hoitaa tehokkaasti, vaikka pääpaino testaamisessa olisikin vielä itse testaajalla. Tällaisia tehtäviä voivat olla mm. laitteen päivittäminen ja konfigurointi eri testien välissä tai erilaisten työkalujen käyttöönotto laitteen päivittämisen yhteydessä.

Käsin testaaminen on verrattain työlästä ja aikaa vievää, joskin sen aikana on herkemmin havaittavissa virheet, jotka helposti jäävät testiautomaation kattavuuden ulkopuolelle. Käsin testaaminen simuloi myös huomattavasti paremmin sitä, kuinka asiakas käyttää lopullista tuotetta, mikä auttaa havaitsemaan myös puutokset ja epäjohtonmukaisuudet, joita kehitystyön varrella on päätyntä ohjelmistoon, joita ei välttämättä ole osattu ottaa huomioon automaatiotestejä laatiessa.

2.3 Automaatio

Testiautomaatio kattaa kaiken sellaisen testaamisen, jossa rakennetaan työvälineitä, jotka toimivat ilman testaajan suoraa vuorovaikutusta. Automaation piiriin on hyvä saattaa kaikki testitapaukset, jotka vaativat toistuvaa testaamista, kuten esimerkiksi päivittäinen käänös (daily build). Automaation perimmäinen tarkoitus täten onkin vapauttaa testaaja tekemään muita töitä toistuvien työtehtävien sijaan [1].

Tyypillisiä automatisoitavia kohteita ovat erilaiset rajapinnat (API) ja käyttöliittymät, jotta voidaan varmistaa toiminnan oikeellisuus ennen ohjelmiston julkaisua. Kohteeksi tulisi kuitenkin rajata toiminallisuuksia ja ominaisuuksia, joiden on syytä olettaa pysyä muuttumattomina pitkälläkin aikavälillä, sillä testiautomaation jatkuva muuttaminen käy projektille kalliiksi. Tähän perustuu tarve myös automaation skaalautuvuuden tarpeesta, sillä myöhempi korjaaminen ja uudelleen suunnittelu käyttää valtavasti resursseja.

Tyypillinen automaation käyttökohde on järjestelmän stressitestaus, jossa pyritään rasittamaan järjestelmää virheiden löytämiseksi [5].

3 Reititys

Reititys (routing) on erityisen herkkä virheille ja työläs varmistaa toimivaksi käsin, mikä lisää merkittävästi automaation rakentamisen tarvetta. Reititys tapahtuu reititysalgoritmien (routing algorithm) pohjalta [6], joiden teoreettinen pätevyys ja toiminta on laajuudeltaan jo yhden kokonaisen opinnäytetyön mittainen ja jätetään näin ollen tarkemman tarkastelun ulkopuolelle tämän työn puitteissa.

3.1 Staattinen ja dynaaminen reititys

Staattisella reitityksellä tarkoitetaan reititystaulujen manuaalista konfiguroimista, eli reititykseen liittyviä tietoja ei ole saatu minkään reititysalgoritmin avulla. Tämä vaatii laajaa tuntemusta ope- roitavasta verkosta. Koska laitteet eivät vaihda tietoja verkon tilasta, ne eivät myöskään pysty havaitsemaan muutoksia verkossa. [7.]

Koska staattinen reititys konfiguroidaan käsin, on käyttäjän helppo ymmärtää, mitä kautta paketit kulkevat. Kuitenkin erilaiset virheet asetusten asettamisessa ovat yleisiä ja yleensä aiheuttavat ainakin osittaisen verkon toimimattomuuden. [7.]

Koska staattinen reititys ei pysty havaitsemaan muutoksia verkossa eikä se sisällä mitään vian tunnistusta, se ei sovellu hyvin suurien tai jatkuvasti muuttuvien verkkojen konfiguroimiseen vaan lähinnä pienten, muuttumattomien verkkojen yhteydessä käytettäväksi.

Dynaamisessa reitityksessä yhdistyy reititysprotokollan ja reititysalgoritmin käyttö. Reititysproto- kollien tehtävä on jakaa informaatiota verkon tilasta laitteiden välillä [7]. Tässä työssä käytettyjä reititysprotokollia tarkastellaan enemmän luvussa 3.2.

Reititysalgoritmi laskee lyhimmän reitin laitteiden välillä, antaen reititysalgoritmeille mahdolli- suuden tehdä ajonaikaisia päätöksiä siitä, mikä reitti on paras kullakin hetkellä [7].

3.2 Reititysprotokollat

Koska reititysprotokollat ovat aiheena erittäin laajoja, rajataan tämän työn puitteissa käsittely vain koskemaan käytössä olleita OLSR- ja OSPF-reititysprotokollaa.

OLSR (Optimized Link State Routing) on liikkuvien langattomien verkkojen (MANET) reitityksessä käytetty, ennakoiva, taulukoihin perustuva protokolla [8]. OLSR:n toiminta perustuu multipoint relay (MPR) solmuihin, joilla pyritään optimoimaan reititystä. Kaikki solmista yhden hypyn päässä olevat solmut oletetaan pystyvän lähettämään dataa molempiin suuntiin ja ovat ainoita solmuja, joille lähetetään paketteja lähetysvaiheessa. Nämä vuorostaan välittävät viestit omille MPR solmuille. Tällä pyritään verkon kuormituksen vähentämiseen. [8.] OLSR:stä on tehty RFC 3626, jossa sen toimintaperiaatteet ovat dokumentoituina [9].

OSPF (Open Shortest Path First) on yleisesti käytetty reititysprotokolla. Se lukeutuu niin sanottuihin interior gateway protokoliin (IGP), joita käytetään reitin löytämiseen autonomisessa järjestelmässä [10]. IGP:t voidaan jakaa kahteen eri kategoriaan: etäisyysvektori-protokoliin (distance-vector protocol) ja linkkitilaprotokoliin (link-state protocol). OSPF lukeutuu näistä jälkimmäiseen ja on Internet Engineering Task Forcen (IETF) standardoima kansainvälinen Open Systems Interconnection (OSI) referenssi [10]. Näin ollen lähes jokaisesta nykypäivän reitittimestä löytyy tuki OSPF:lle.

Linkkitilaprotokollissa mainostetaan jokaisen solmun tilaa lähettämällä paketti kaikkiin verkossa oleviin reitittimiin, jotka muodostavat saatujen pakettien avulla topologiakartan, niiden väliset linkit ja välimatkojen hinnat. Jokainen reitin laskee saamiensa tietojen perusteella välimatkojen hinnan suhteessa viereisiin solmuihin ja muodostavat sen perusteella reititystaulun. [10]. Vaikka kaikki solmut näkevät koko verkon samassa topologiassa, näkevät ne verkon kokonaisuutena, jossa ne ovat itse koko puurakenteen alakerros [11].

3.3 OSI-malli

Open Systems Interconnection model (OSI model) on havainnollistava tapa esittää seitsemän kerroksen avulla tiedonsiirtoprotokollien toimintaa. Nämä seitsemän kerrosta ovat sovelluskerros (application), esitystapakerros (presentation), istuntokerros (session), kuljetuskerros (transport),

verkkokerros (network), siirtokerros (data link) sekä fyysinen kerros (physical layer). Näistä koostuen on mahdollista esittää, kuinka informaatio saadaan muotoon, jossa näemme sen erinäisten sovellusten parissa [12]. OSI-mallin kerrokset havainnollistetaan kerroksittain (Kuva 2).



Kuva 2. OSI-mallin kerrokset

Jokainen OSI-mallin kerros on erilainen toiminnoiltaan, mutta kaikki vastaavat informaation välitymisestä sekä mallin ylä- että alapuolella olevalle seuraavalle kerrokselle. Täten ilman, että datan olisi mahdollista kulkea molempiin suuntiin OSI-mallin kerroksien välillä, kahden laitteen välinen kommunikointi ei olisi mahdollista.

TCP/IP protokollakokoelma (TCP/IP protocol suite) kehitettiin ennen kuin OSI-mallia oli keksitty, joten vaikka periaatteet ovat samanlaisia, kaikkia kerroksia ei ole käytössä sen toteutuksessa. Sen sijaan TCP/IP käyttää DoD-mallia (Department of Defense reference model), jonka kehitti Yhdys-

valtojen puolustusvoimat. DoD-mallissa on käytössä vain neljä kerrosta: sovelluskerros (application), kuljetuskerros (transport), internetkerros (internet), sekä verkkorajapintakerros (network interface layer). [12.] DoD-mallin kerrokset ovat sisällytettynä OSI-malliin ja näin ollen voidaan havainnollistaa vierekkäin (Kuva 3).

| | |
|---------------------------|--------------------------------|
| 7 Sovelluskerros | 4 Sovelluskerros |
| 6 Esitystapakerros | |
| 5 Istuntokerros | |
| 4 Kuljetuskerros | 3 Kuljetuskerros |
| 3 Verkkokerros | 2 Internetkerros |
| 2 Siirtokerros | 1 Verkkorajapintakerros |
| 1 Fyysinen kerros | |

Kuva 3. OSI-mallin kerrosten vastaavuus DoD-mallissa

4 Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec) on verkkoviestinnässä käytetty työkalu, joka suojaa kahden laitteen välisen kommunikaation hyödyntäen salausta (encryption).

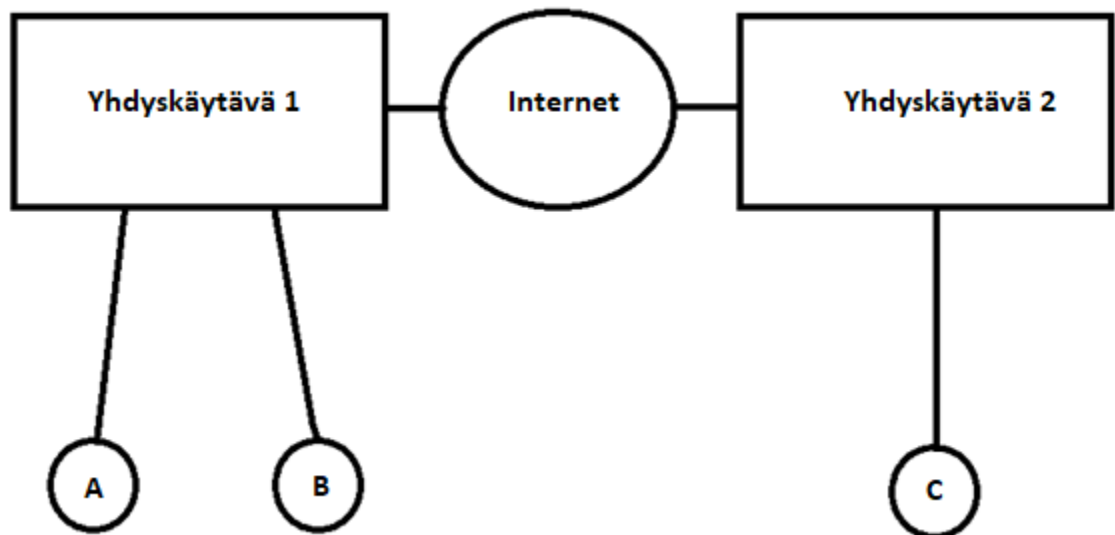
James Tiller kirjassaan *A Technical Guide to IPSec Virtual Private Networks* kuvaa IPseciä kaikenkattavaksi termiksi, joka pitää sisällään monia erilaisia kommunikoinnin hallitsemiseen, käyttöön-ottoon, laitteiden väliseen kommunikointiin ja neuvotteluun, varmennukseen, liikenteen kontrollointiin sekä toimintaperiaatteiden implementointiin käytettäviä komponentteja [14].

4.1 Yleistä

IPsecillä on mahdollista tukea kaikkia IP-protokollia kuten esimerkiksi TCP-protokollaa (transmission control protocol), UDP-protokollaa (user datagram protocol) ja ICMP-protokollaa (internet control message protocol). Sillä on mahdollista suojata kahden tai useamman laitteen, eli hostin, välinen liikenne. [15.] Salaus on mahdollista toteuttaa esimerkiksi ennalta jaetulla avaimella (pre-shared key, PSK) tai julkisen avaimen salausalgoritmillä (RSA).

Koska eri hostien välisen kommunikaation suojauksessa on usein tarve myös liikenteelle eri verkkojen välillä, toteutettiin IPseciin kaksi erilaista tuettua loogista rakennetta eli topologiaa: laitteiden välistä (host-to-host) sekä verkkojen välistä (gateway-to-gateway) topologiaa. Laitteiden välisessä topologiassa suojattu yhteys muodostetaan suoraan kahden päätelaitteen välille, kun taas verkkojen välisessä topologiassa on kaksi yhdyskäytävää (gateway), jotka toimivat verkkojen tunnettuna välittäjinä (proxy) oman verkkonsa puolesta. [15.]

Kuvassa 4 on havainnollistettu IPsec topologia.



Kuva 4. IPsec-topologia

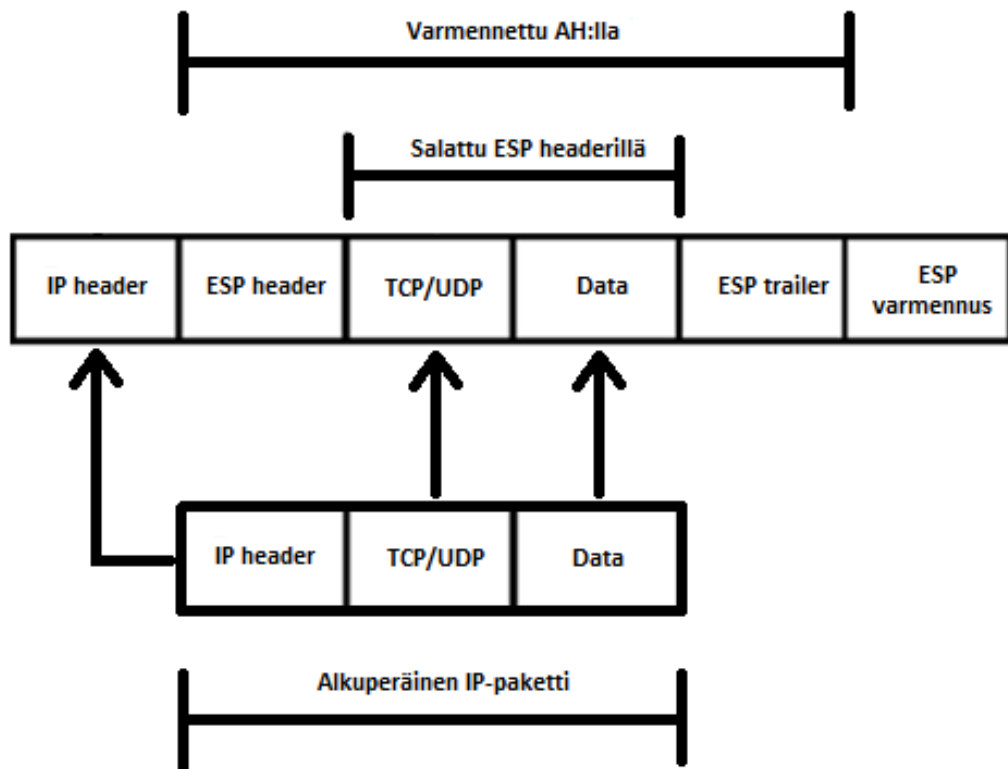
Laitteiden välisessä mallissa kuvan solmun A on mahdollista muodostaa IPsec-tunneli solmulle C yhdyskäytävien 1 ja 2 kautta. Vaihtoehtoisesti voidaan rakentaa verkkojen välinen topologia, jossa A on tunnettu solmu yhdyskäytävälle 1, joka muodostaa IPsec-tunnelin yhdyskäytävään 2, joka vuorostaan tuntee solmun C. Laitteiden välisessä topologiassa yhteys on suojattu koko matkan A:lta C:lle, mutta verkkojen välisessä topologiassa vain yhdyskäytävien väli on suojattu, näin ollen jättäen laitteen ja yhdyskäytävän välisen liikenteen suojaamattomaksi. Tätä suojauksen puutosta on korjattu mm. sallimalla solmujen keskustella suoraan yhdyskäytävien kanssa. [15.]

4.2 Kuljetus- ja tunnelikäyttötapa

IPsecin suojaus perustuu kahteen suojausprotokollaan: varmennuspäätteeseen (authentication header, AH) ja tiivistettyyn suojaussisältöön (encapsulated security payload, ESP). Näistä protokollista molemmat tarjoavat eheyden ja varmennuksen välitettävälle datalle, mutta vain ESP tukee datan eheyttä suojauksen yhteydessä. [15.] Näiden lisäksi on kehitetty internetavaimen vaihdos (inter key exchange, IKE), joka mahdollistaa ESP:n ja AH:n toiminnan [14]. IETF on laatinut IKE:stä vuonna 1998 kolme eri RFC:tä (RFC 2407-2409). Sittemmin IKE:n RFC:itä on päivitetty useaan eri otteeseen ja vuoteen 2014 mennessä IKE:een liittyviä RFC:itä on tehty yhteensä seitsemän kappaletta. Näistä viimeisin, RFC 7296, teki IKE:stä esitetyn standardin sijasta internet standardin [16].

IPsec tarjoaa kaksi erilaista käyttötapaa käytössä olevan järjestelmän vaatimuksien asettaman tarpeiden mukaan. Hyödyntäen kuvan 4 havainnollistusta: kuljetuskäyttötavassa (transport mode) alkuperäinen keskusteluun käytetty protokolla, kuten esimerkiksi ICMP, kuljettaa koko matkan datan solmulta A solmulle B, mutta IPseciä käytetään vain datan salaamiseen lähettäjällä ja viestin varmennukseen vastaanottajalla. Tunnelikäyttötavassa (tunnel mode) data kapseloidaan (encapsulation) salausta ja varmennusta myöten, jonka jälkeen data siirretään kuljetuskäyttötavan mukaisesti A:lta B:lle. [14.]

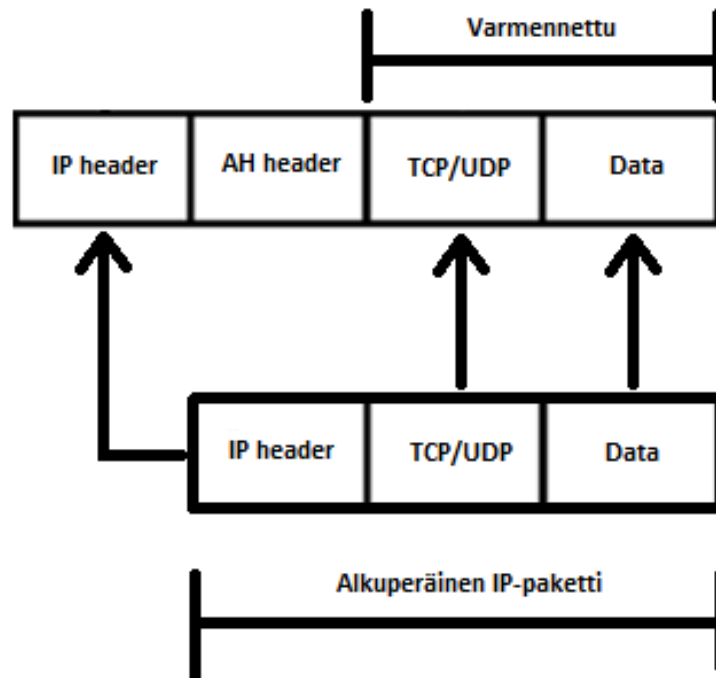
Kuljetuskäyttötavassa kuljetettavan paketin IP-header muutetaan käytetyn AH tai ESP-headerin myötä protokollan ID:n 50 (ESP) tai 51 (AH) mukaiseksi. Alkuperäisen protokollan ID:n arvo säilytetään IPsec traileriin, josta se palautetaan paketin salauksen purkamisen yhteydessä. [17.] Kuljetuskäyttötapaa ESP-headereiden avulla havainnollistaa kuva 5.



Kuva 5. Kuljetuskäyttötapaa ESP headerillä.

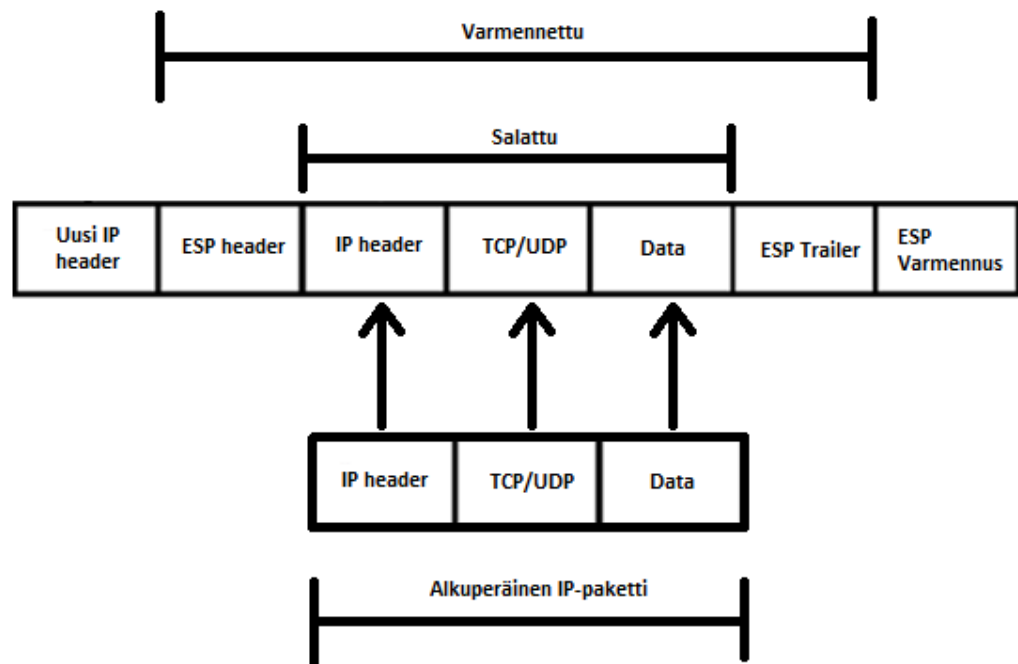
Kuljetuskäyttötavassa IP-header siirretään paketin alkuun, mikä todistaa kuljetuskäyttötavan sisältämän suojauksen puutteen. Siitä syystä sitä käytetään usein jonkin toisen tunnelointiprotokollan kanssa yhteisesti.

Kuljetuskäyttötavassa AH:ta voidaan käyttää joko yksin tai ESP:n kanssa yhdessä. Kuten kuvasta 6 alla havaitaan, AH ei myöskään suojaa IP-headeriä, vaan siirtää siitä kopion paketin eteen, joten sama ongelma IP-headerin sisältämän informaation suojaamattomuudesta säilyy.



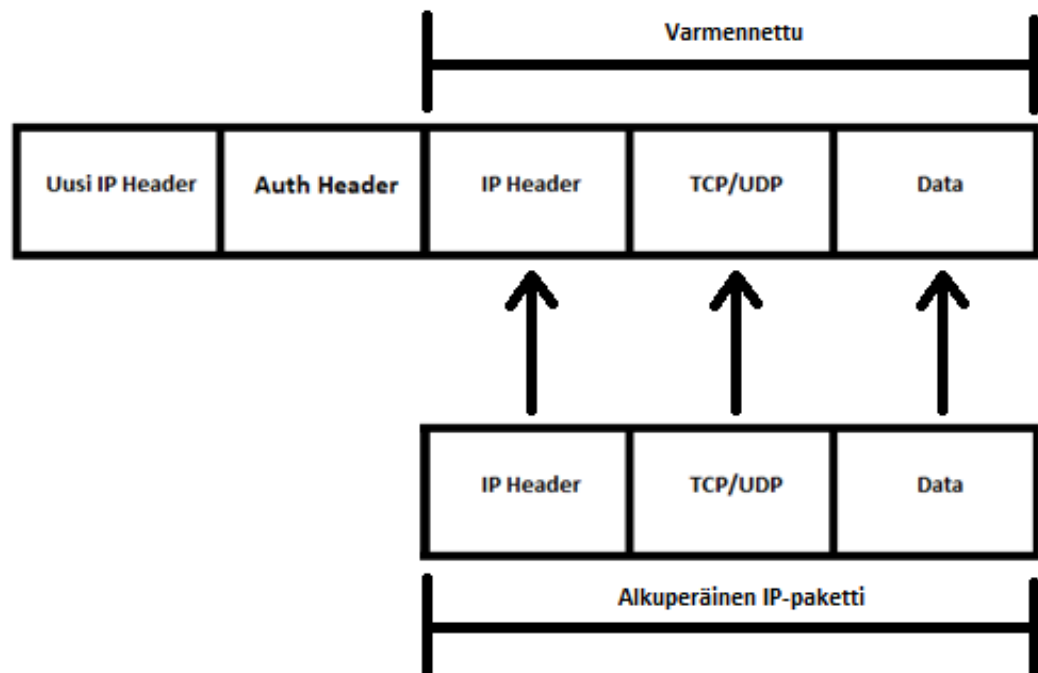
Kuva 6. Kuljetuskäyttötapa AH headerillä.

Tunnelikäyttötapa on IPsecin oletuskäyttötapa. IP header merkitään IP-protokollan ID:llä 50 (ESP) tai 51 (AH). Paketti salataan ja siihen lisätään uusi IP header. Tämän jälkeen paketti voidaan lähettää vastaanottajalle. [17]. Tunnelikäyttötapaa havainnollistetaan kuvassa 7.



Kuva 7. Tunnelikäyttötapa ESP headerillä.

Kuten kuljetuskäyttötavassakin, AH:ta on mahdollista käyttää joko ESP:n kanssa tai ilman. Kuva 8 havainnollistaa AH headerin toimintaa tunnelikäyttötavassa.



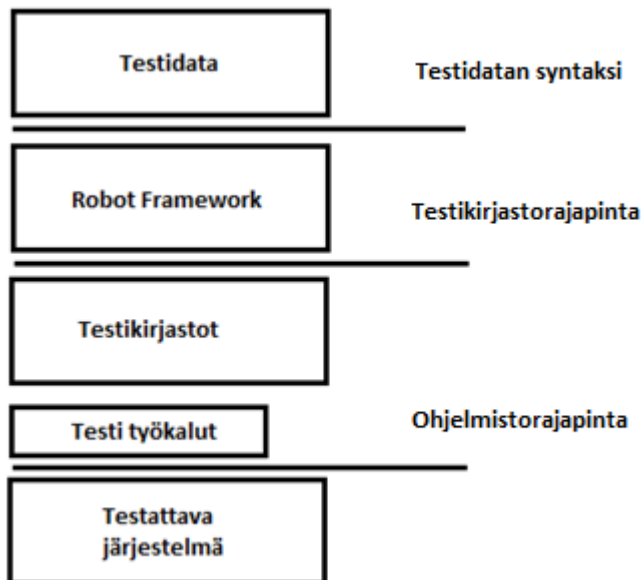
Kuva 8. Tunneli käyttötapa AH headerillä.

Solmun B saatua paketin solmulta A voidaan sen salaus purkaa käyttäen hyväksi joko yhteneväisiä PSK-avaimia tai solmujen sertifikaatteja.

5 Robot Framework

Työssä käytetään Robot Frameworkia (myöhemmin Robot), joka on tyypillinen työkalu testiautomaation laatimisessa. Robot on avoimen lähdekoodin automaatiotyökalu, joka on julkaistu Apache 2.0 -lisenssin alaisena, mikä antaa varsin laajat oikeudet sen hyödyntämiseen ohjelmistoprojekteissa. Robot on toteutettu hyödyntäen Python-ohjelmointikieltä, ja sitä on mahdollista käyttää joko Pythonilla, Jythonilla tai IronPythonilla. Robot kehitettiin alun perin Nokia Networkilla Pekka Klärckin toimesta mutta se avattiin avoimen lähdekoodin projektiksi vuonna 2008. [2.]

Robot Frameworkin korkean tason arkkitehtuuri on kuvattu sen dokumentaatiossa kuvan 9 mukaisesti:



Kuva 9. Robotin korkean tason arkkitehtuuri [18]

Testidatan editointi on vaivatonta ja helppolukuista. Itse automaation toimintaympäristö ei tiedä varsinaisesti mitään testattavasta järjestelmästä, vaan kaikki testaaminen tapahtuu ja käsitellään kirjastojen avulla. Kirjastojen on mahdollista hyödyntää erilaisia alemman tason työkaluja ajureina tai hyödyntää rajapintoja suoraan. [18]

5.1 Robot Framework automaatiiossa

Robot on modulaarisuutensa vuoksi erinomainen työkalu käytettäväksi missä tahansa automaatiota vaativassa ohjelmistoprojektissa. Sen toiminta on kehitetty avainsanojen (keyword) ympärille, jotka kerätään kirjastoihin. Juuri kirjastojen tuominen osaksi omaa testiympäristöä on yksi modulaarisuuden peruspilareista. Oikein toteutettuna testiautomaatiokehys antaa mahdollisuudet hyödyntää omassa testikokoelmassa (test suite) muiden samassa projektissa olevien automaatiotestien avainsanoja, jolloin samaa toiminnallisuutta ei tarvitse toteuttaa jatkuvasti uudelleen tai kopioida eri paikkojen välillä.

5.1.1 Työkalun käyttöönotto

Robotin Jython vaatii java-alustan ja IronPython .NET-alustan toimiakseen. Opinnäytetyössä hyödynnettiin Python versiota Robotista, jonka asennus on mahdollista suorittaa python komennolla *"python setup.py install"* Robot Frameworkin lähdekoodin hakemistossa, tai käyttämällä komentoa *"pip install robotframework"* mikäli tietokoneella on Python ja pip asennettuna. Robot Frameworkin uusin versio 4.0 tukee Python 2.7, 3.5 sekä sitä uudempia Python versioita. Testiautomaation tuen jatkuvuuden kannalta suositeltavaa on kuitenkin käyttää vähintään Python 3.6 versiota, sillä vanhempien versioiden tuki on loppunut, joten uusien projektien aloittaminen vanhalla versiolla saattaa aiheuttaa turhan pullonkaulan testaamiseen. [18.]

Windows pohjaisilla tietokoneilla, joilla Python on tarvinnut asentaa itse, on hyvä lisätä Windowsin PATH muuttuja listaan Robotin runner-skriptien sijainti, jotta Robotin käyttäminen komentoriviltä on vaivattomampaa. Unix pohjaisissa käyttöjärjestelmissä täytyy muokata yleisemmän tason konfiguraatio tiedostoja, joten niitä varten on hyvä tutustua käyttöjärjestelmän omiin dokumentaatioihin voidakseen muokata oikeita asetuksia. [18.]

5.1.2 Testien suunnittelu

Testien suunnittelemisen kannalta on olennaista kartoittaa käytettävissä olevan laitteiston määrä ja ominaisuudet. Testit tulisi suunnitella luettavuudeltaan sellaiseen muotoon, että testien kulun voi ymmärtää myös ohjelmointiin täysin perehtymätön henkilö. Avainsanojen on hyvä olla lyhyitä

ja kuvaavia testivaiheessa suoritettuun operaatioon nähden ilman, että otetaan kantaa siihen, miten välivaihe toteutetaan. Esimerkiksi avainsanana voisi toimia:

Ping from PC1 to PC2

Tämän avainsanan perusteella on helppo ymmärtää, että aloitetaan ping-operaatio, joka lähettää ICMP pulssin ensimmäiseltä testitietokoneelta toiselle. Liiallisen tiedon, kuten esimerkiksi testitietokoneiden IP-osoitteiden lisääminen, heikentäisi avainsanan toteutuksen muokattavuutta ja olisi muutenkin itse testioperaation kuvaamiselle tarpeetonta informaatiota; kun avainsanat pysyvät tarpeeksi korkean tason kuvauksina toiminnasta, ei niitä ole tarpeen muokata joka kerralla, kun toteutusta hieman muutetaan. Avainsanat toteutettavat peräkkäisinä välivaiheina yhden testitapausten (test case), joka on toiminnan varmistamisen perusyksikkö automaatiossa.

Mikäli testiautomaatiokehys on toteutettu oikein, on mahdollista hyödyntää muiden kehittämiä avainsanoja omien testitapausten suunnittelussa ja toteutuksessa. Parhaassa tapauksessa automaatiota kehittäessä Python -toteutusta ei tarvitse kirjoittaa kuin erittäin vähän tai ei ollenkaan, vaan testitapausten toteutus tapahtuu puhtaasti järjestelemällä jo olemassa olevia avainsanoja haluttuun järjestykseen.

5.1.3 Testien toteutus

Niputtamalla samaan aihepiiriin kuuluvat testitapaukset yhden Robot-tiedoston alle saadaan luotua testikokoelma (test suite). Testikokoelman sisältämien tapausten on mahdollista hyödyntää kaikkia niitä metodeja, joita siihen sisällytetyt kirjastot sisältävät. Näin ollen työn määrä vähenee, kun ei tarvitse jokaiselle tapaukselle tehdä toteutuksia useassa eri tiedostossa. Tämä ominaisuus parantaa mm. testien luettavuutta ja uudelleenkäytettävyyttä.

Testikokoelman olisi hyvä koostua enimmillään kymmenestä eri testitapauksesta, ellei hyödynnetä datalähtöistä lähestymistapaa (data-driven approach), jossa testitapaus koostuu vain yhdestä korkean tason avainsanasta [18].

Testikokoelmalle tulee määritellä alustus- (setup) ja lopetustoimet (teardown). Alustuksessa on hyvä tehdä kokoelmalle tyypilliset valmistelutoimenpiteet, jotka ovat samat kaikissa testitapauksissa kuten laitteelle sisäänkirjautuminen tai pohjakonfiguraation lataaminen testattavalle laitteelle. Alustustoimenpiteet määritellään aivan tiedoston alussa (Kuva 10).

```

*** Settings ***
Documentation      Test suite for Isec tests.

Library           IsecLib.py

Suite Setup       Suite Setup
Suite Teardown    Suite Teardown

```

Kuva 10. Esimerkki testikokoelman alustamisesta.

Alustus- ja lopetustoimien varsinainen toiminta määritellään erillisessä avainsanat-osiossa myöhemmin tiedostossa esimerkiksi kuvan 11 osoittamalla tavalla:

```

*** Keywords ***

Suite Setup
    Open session
    Mount testtools
    Set base configuration          ${BASE}

Suite Teardown
    Close session

```

Kuva 11. Alustus- ja lopetustoimien määrittely.

Varsinainen Python-toteutus, joka oikeasti määrittelee mitä mikäkin avainsana näiden operaatioiden takana tekee, on erillisessä Python-kirjastossa (library), johon metodit toteutetaan. Tässä esimerkissä kirjastona toimii kuvan 10 IsecLib.py-tiedosto.

Alustukseen liittyvien asetusten jälkeen toteutetaan muuttujien alustaminen. Muuttujiin tulisi sijoittaa kaikki sellainen data, jota testitapausten ajamisessa hyödynnetään. Esimerkiksi on hyvä tapa alustaa \${HOST} muuttuja kovakoodatun IP-osoitteen sijaan, jolloin muuttuja voidaan antaa ajon aloittamisen yhteydessä komentorivillä. [18.]

Joskus kuitenkin muuttujiin on perusteltua käyttää staattisia arvoja, kuten laitekonfiguraation polkuja, tällöin muuttujien alustaminen näyttää kuvan 12 mukaiselta:

```

*** Variables ***

${CONFIG}          config/test_host_config.py
${BASE}            ${CURDIR}/confs/basel.json
${TC18044_CONF1}   ${CURDIR}/confs/olsr_psk.json
${TC18044_CONF2}   ${CURDIR}/confs/olsr_psk.json
${TC18045_CONF1}   ${CURDIR}/confs/olsr_rsa.json
${TC18045_CONF2}   ${CURDIR}/confs/olsr_rsa.json

```

Kuva 12. Muuttujien alustaminen robot-tiedostossa.

5.1.4 Testien ajaminen

Testien ajaminen tapahtuu tyypillisesti komentorivillä käyttämällä *robot* -komentoa:

```
robot [options] IpsecTests.robot
```

Aikaisemmissa versioissa oli käytössä *pybot*, *jybot* ja *ipybot* komennot, joilla testit oli mahdollista käynnistää, mutta robotin version 3.1 jälkeen kaikki kääntäjät (interpreter) käyttävät uutta menetelmää vanhojen sijaan.

Robot valitsee jäsentäjän (parser) sen mukaan, mikä tiedostopääte on käytössä testidatassa, joista tyypillisimpinä ovat txt- tai robot-tiedostot. Näiden lisäksi tuettuina ovat HTML- ja reStructuredText-tiedostot [18].

5.1.5 Testien tulokset

Ajon lopuksi tulokset kerääntyvät robotin ajamaan lokihakemistoon (logs folder), jonne automaattisesti luodaan kolme eri tiedostoa: log.html, output.xml sekä report.html.

Log.html sisältää graafisen näkymän testien tuloksista vaiheittain merkiten jokaisen eri automaattiotestaamisen välivaiheen joko vihreällä testin onnistuessa tai punaisella epäonnistumisen kohdalla. Jos välivaihe epäonnistuu, lokitiedostoon kirjautuu saatu virheviesti, joka helpottaa ongelmakohdan selvityksessä ja korjaamisessa. Kuvassa 13 on kuvattu esimerkinäkymä lokitiedostosta, kuten se on esitetty robotin omilla kotisivuilla:

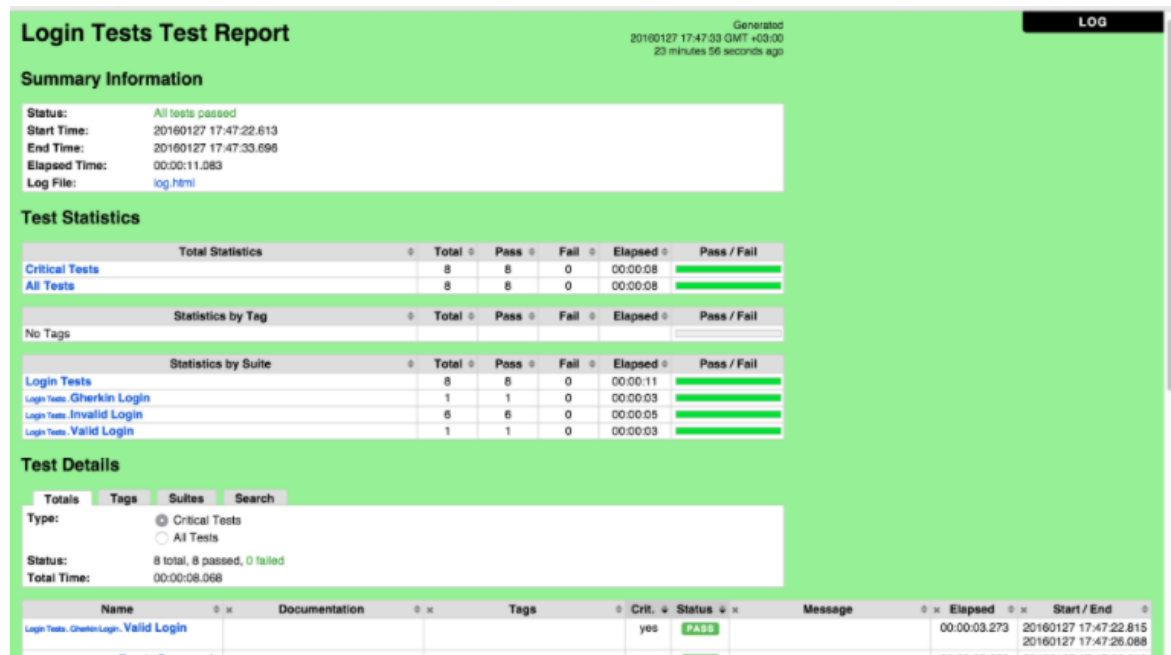


Kuva 13. Lokitiedosto Robotissa [19].

Lokitiedosto näyttää havainnollisesti, kuinka ajossa olleet testit ovat sujuneet, niihin käytetyn ajan, sekä välivaiheiden kulun. Mahdolliset ajossa tapahtuneet virheet tulee merkityksi sekä lokitiedoston alkuun sijoitettuun koosteeseen, että välivaiheisiin alla mahdollisten virheilmoitusten kanssa, mikä helpottaa ajossa ilmenneen ongelman tutkimista ja korjaamista.

Output.xml -tiedostoon kerääntyy tarkemmat välivaiheiden antamat tiedot teknisine yksityiskohdineen. Tiedon runsauden vuoksi tästä tiedostosta on haastavaa löytää haluamansa informaatio, ellei tiedä jo suunnilleen mitä on etsimässä. Tunnetun ongelman juurisyiden (root cause) etsimiseen tämä on kuitenkin hyvä työkalu.

Kolmannessa, report.html, tiedostossa kerätään ajoista korkeamman tason raportti, jolla on helpompi arvioida kokonaisuutta (Kuva 14). Kun kaikki palautetiedostot luodaan ajon lopuksi, raporttiin tulee linkit luotuun lokitiedostoon, jolloin korkeamman tason raportista on helppo mennä lokiin tarkastelemaan tarkempia yksityiskohtia [18].



Kuva 14. Raporttitiedosto Robotissa [19].

5.1.6 Tulosten hyödyntäminen

Saatujen tulosten pohjalta on mahdollista tehdä kattavaa arviota ohjelmiston toiminnan oikeellisuudesta sekä siitä, täyttääkö se kaikki asiakkaan asettamat vaatimukset toiminnalle. Sumit Bisht kirjassaan *Robot Framework Test Automation* asettaa raporteille neljä syytä miksi niitä tulisi pitää välttämättöminä: nopea käsitys testiajosta, vertailtavuus testiajojen välillä, yksittäisten tulosten yksityiskohtien löytäminen sekä älykäs arviointi. Erityisesti raporttien ja lokitiedostojen avulla on mahdollista lyhentää työmäärää, jota kuluu testiajon ymmärtämiseen pelkän tietokoneen konsolin antaman palautteen avulla. [20.]

Mikäli monia virheitä alkaa ilmetä automaatioajojen aikana on syytä arvioida kriittisesti, onko virhe luoduissa testitapauksissa vaiko testattavassa ohjelmistossa. Maksimaalisen hyödyn saavuttamiseksi yksikään punaisella merkitty välivaihe lokitiedostossa ei jää ilman virhetiketin avaamista ja tarkkaa tutkimusta, sillä erinäisten pienien virheiden kasaantuminen rappauttaa automaation luotettavuutta pitkällä aikavälillä.

6 Yhteenveto ja pohdinta

Opinnäytetyön tavoitteena oli laajentaa toimeksiantajana toimineen Bittium Wireless Oy:n Kaajan toimiyksikön automaatiotestauksen kapasiteettia ja kattavuutta. Toteutetun opinnäytetyön käytännön osuuden kannalta suurimmaksi haasteeksi muodostui kesken työn muuttunut laitevaatimusten määritelmä, joka hidasti toteutettavan automaation rakentamista merkittävästi. Laitepuutteen, sekä projektin sisäisten resursointien vuoksi, varsinainen automaation toteutus IPsecille jäi keskeneräiseksi tämän työn puitteissa. Automaatiopaikan suunnittelu- ja laite-laajennustyö saatettiin kuitenkin loppuun asti. Automaatiotestitapausten kirjallinen suunnittelu toteutettiin loppuun prosessin mukaisesti. Tehty työ antoi perustan, jonka päälle jatkokehitys on mahdollista ja yksinkertaista toteuttaa. Työn suunniteltujen testitapausten mukaisesti IPsec automaation kehitystä on tarkoitus jatkaa osana projektin työmäärää myöhempänä ajankohtana. Tämän opinnäytetyön haasteiden myötä todella huomasin myös, kuinka tärkeää on kartoittaa myös sellaisia riskitekijöitä, jotka eivät ole kovin todennäköisiä. Jos mahdollisuutta laitemäärän kasvamiselle olisi pidetty realistisena jo työn alkuvaiheessa, olisi testien suunnittelussa voitu paremmin huomioida reititykset poikkeavassa ympäristössä.

Opin valtavasti Robot Frameworkista tätä automaatiota suunnitellessa ja toteuttaessa. Sen yleis-pätevyys lisää merkittävää arvoa testaamiseen, ja aloittaminen jo olemassa olevaan toimintaympäristöön on mielestäni vaivatonta. Robotilla automaation kehittäminen vahvensi myös Python osaamistani, sillä automaation toteutus perustui siihen.

Tekemisen aikana opin valtavan määrän testiautomaation koko elinkaaresta vaatimusmäärittelyjen laatimisesta ylläpitoon. Kaikki työssä saatu kokemus nitoutui hyvin aikaisemmissa työtehtävissäni toteuttamaani käsin testaamiseen. Automaatioon liittyvä osaaminen tulee jatkossa olemaan merkittävä lisä osaamiseeni testaajana.

Lähteet

- 1 Kasurinen, J. (2015). *Ohjelmistotestauksen käsikirja*. Saatavilla <http://www.kamk.fi/kirjasto>, Ellibslibrary
- 2 Robot Framework Introduction. Saatavilla 3.2.2021: <http://robotframework.org/#introduction>
- 3 Karjalainen, H. (2016). *Robot Framework reitityksen automaattisen testaamisen toteutuksessa*. AMK-opinnäytetyö. Tampereen ammattikorkeakoulu. <http://urn.fi/URN:NBN:fi:amk-2016120919674>
- 4 Myers, G., Sandler, C. & Badgett, T. (2012). *The art of software testing 3rd edition*. Saatavilla <http://www.kamk.fi/kirjasto>, Ebook Central
- 5 Chorafas, D. (2007). *Stress Testing for Risk Control Under Basel II*. <http://www.kamk.fi/kirjasto>, Ebook Central
- 6 Medhi, D. & Ramasamy, K. (2007). *Network Routing*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central
- 7 Misra, S. & Goswami, S. (2017). *Network Routing*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central
- 8 Sarkar, S. (2012). *Wireless Sensor and Ad Hoc Networks Under Diversified Network Scenarios*. Saatavilla: <http://kamk.fi/kirjasto>, Ebook Central
- 9 RFC 3626. Saatavilla 27.4.2021: <https://tools.ietf.org/html/rfc3626>
- 10 Eiji, O. & Vogt, C. (2012). *Advanced Internet Protocols, Services, and Applications*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook central
- 11 Davies, G. (2004). *Designing and Developing Scalable IP Networks*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central
- 12 Ciccarelli, P. & Faulkner, C. (2004). *Networking Foundations: Technology Fundamentals for IT Success*. Saatavilla 16.4.2021: <http://www.kamk.fi/kirjasto>, Ebook Central

- 13 Blank, A. (2002). *TCP/IP JumpStart: Internet Protocol Basics*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central
- 14 Tiller, J. (2017). *A Technical Guide to IPsec Virtual Private Networks*. Saatavilla <http://www.kamk.fi/kirjasto>, Ebook Central
- 15 Held, G. (2004). *Virtual Private Networking*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central
- 16 RFC 7296. Saatavilla 27.4.2021: <https://tools.ietf.org/html/rfc7296>
- 17 Firewall. *Understanding VPN IPsec Tunnel Mode and IPsec Transport Mode - What's the Difference?* Saatavilla 27.4.2021: <http://www.firewall.cx/networking-topics/protocols/870-ipsec-modes.html>
- 18 Robot Framework User Guide version 4.0.1. Saatavilla 16.4.2021: <http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>
- 19 Robot Framework Examples. Saatavilla 21.4.2021: <http://robotframework.org/#examples>
- 20 Bisht, S. (2013). *Robot Framework Test Automation*. Saatavilla: <http://www.kamk.fi/kirjasto>, Ebook Central