jamk.fi

Verkkoprotokollia toteuttavien ohjelmistojen automaattinen tietoturvatestaus

Matti Ruusupiha

Opinnäytetyö
Joulukuu 2018
Tieto- ja viestintätekniikka
Insinööri (AMK), tieto- ja viestintätekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Tekijä(t)	Julkaisun laji	Päivämäärä
Ruusupiha, Matti	Opinnäytetyö, AMK	Joulukuu 2018
	Sivumäärä 41	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Verkkoprotokollia toteuttavien ohjelmistojen automaattinen tietoturvatestaus

Tutkinto-ohjelma
Tieto- ja viestintätekniikka

Työn ohjaaja(t)
Marko Rintamäki ja Juha Saarisilta

Toimeksiantaja(t)
Insta DefSec OY

Tiivistelmä

Opinnäytetyön tarkoituksena oli kehittää ratkaisu, joka mahdollistaisi multicast-verkkoprotokollien ohjelmallisten toteutusten automatisoidun tietoturvatestauksen. Projektin pää-paino oli rikkovissa testeissä. Ratkaisussa toteutettiin testitapauksia multicast -lähetystekniikkaan liittyville IGMP ja PIM -protokollille.

Multicast-lähetystekniikka mahdollistaa useammalle vastaanottajalle tarkoitetun datan lähettämisen yksittäisenä virtana, joka monistetaan tarvittaessa pakettien liikkuessa kohti vastaanottajia. Tällainen tiedonlähetys vähentää verkon kuormaa verrattuna unicast -lähetystekniikkaan ja on omiaan esimerkiksi multimediasisällön jakamiseen useille vastaanottajille.

Ratkaisu koostui kahdesta Robot Framework -kirjastosta, jotka luotiin testauksessa tarvittavia avainsanoja varten, sekä yhdestä Scapy -kirjastosta, jossa implementoitiin PIM -protokollan keskeiset pakettityypit. Lisäksi valittujen protokollien testausta varten luotiin Robot Framework -testitapaukset. Testit ajettiin Jenkins -automatisointiserverin avulla, mikä mahdollisti testien automaattisen ajon ja tulosten raportoinnin. Jenkinsillä automatisoitujen ajojen parametrisointia varten luotiin shell skriptit.

Projektin lopputulos vastasi onnistuneesti tehtävänannossa asetettuja vaatimuksia. Ratkaisu vaatii kuitenkin vielä jatkokehitystä käytettävyyden osalta. Kehitetyillä menetelmillä kohdetta testattaessa ei löydetty vakavia ohjelmistovirheitä. Muutama erikoisuus havaittiin.

Avainsanat (asiasanat)
tietoturva, tietoturvatestaus, robot, scapy, python, verkkoprotokollat, ohjelmointi

Muut tiedot (salassa pidettävät liitteet)

jamk.fi

Author(s)	Type of publication	Date
Ruusupiha, Matti	Bachelor's thesis	December 2018
		Language of publication: Finnish
	Number of pages: 41	Permission for web publication: x

Title of publication

Automated security testing for applications implementing network protocols

Degree programme

Information and Communications Technology

Supervisor(s)

Rintamäki, Marko; Saarisilta, Juha

Assigned by

Insta DefSec OY

Abstract

The goal was to develop a solution for automated security testing for applications implementing network protocols. The focus of the project was on destructive testing. The test cases were created for multicast related protocols IGMP and PIM.

Multicast technology enables one to send a single data stream intended for multiple recipients, which is then duplicated when necessary. This sort of data transmission is less taxing for network links than unicast and works well for broadcasting multimedia content for multiple recipients.

The solution consists of two Robot Framework libraries containing the keywords needed for building the automated test cases and a Scapy library, which implements the essential packet types for PIM protocol. Robot test suites were also created for the chosen protocols. The suites were executed using a Jenkins automation server, which enabled the interpretation and archiving of the tests results. To help parametrization, shell scripts were created to be used by Jenkins during automated runs.

The finished project was in accordance with the requirements set for the project. However, the solution still requires further development regarding general usability. When testing the target with the developed solution, no critical software vulnerabilities were found; however, some peculiarities were found.

Keywords/tags (subjects)

security, security testing, robot, Scapy, Python, network protocols, programming

Miscellaneous (Confidential information)

Sisältö

Kuviot

Taulukot

Sanasto

ATDD	Acceptance Test-Driven Development
BSR	Bootstrap Router
CI	Continuous Integration
DR	Designated Router
DSL	Domain Specific Language
HTML	Hyper Text Markup Language
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
MAC	Media Access Control
PIM	Protocol-Independent Multicast
PIM-SM	PIM Sparse Mode
PIM-DM	PIM Dense Mode
PIM-SSM	PIM Source Specific Multicast
RP	Rendezvous Point
RPT	Rendezvous Point Tree
SPT	Shortest Path Tree
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege

1 Johdanto

Opinnäytetyön tavoitteena oli kehittää ratkaisu, joka mahdollistaisi eri verkkoprotokollien ohjelmallisten toteutusten automaattisen tietoturvatestauksen ja soveltuisi toimeksiantajan testausympäristöön. Testauskohteena oli eräs IGMP- ja PIM -verkkoprotokollien implementaatio. Projektin fokus oli destruktiivisten testien kehittämisessä. Työn toimeksiantajana toimi Insta DefSec OY (iDS).

Insta DefSec OY toimii osana Insta konsernia, joka on noin tuhat henkilöä työllistävä suomalainen perheyritys (Insta on turvallisen ja kilpailukykyisen yhteiskunnan rakentaja ja kehittäjä 2018). Yhtiö tuottaa muun muassa hätä- ja hälytyskeskusratkaisuja, kyber- ja tietoturvaratkaisuaja sekä puolustusratkaisuja. Asiakkaina sillä on erinäisiä viranomaisia sekä korkeaa turvallisuustasoa vaativia yrityksiä. Insta DefSec on myös Suomen puolustusvoimien strateginen kumppani. (Insta DefSec – Keitä me olemme? 2018)

Käytännössä työ toteutettiin ketteränä ohjelmistokehityksenä yhteistyössä iDS:n kanssa. Kehityksen aikana tutustuttiin testattaviin verkkoprotokolliin ja valittuihin kirjastoihin sekä luotiin ratkaisu, joka mahdollisti automatisoitujen testitapausten kirjoittamisen, ajamisen ja tulosten raportoinnin. Työn tutkimusmenetelmänä käytettiin toimintatutkimusta.

2 Ohjelmistotestauksen teoriaa

2.1 Testausajattelu

Täydellisen ohjelmiston kirjoittamista ei yleisesti pidetä mahdollisena. Tästä syystä ohjelmistoa tulee testata ennen sen siirtymistä tuotantoon. Tuotannossa olevassa ohjelmistossa esiintyvä virhe saattaa vaikuttaa negatiivisesti esimerkiksi ohjelmistoa käyttävän yhtiön maineeseen, yleiseen turvallisuuteen sekä käyttäjien turvallisuuteen. Testauksella voidaan löytää ja eliminoida ohjelmistossa olevia virheitä, ennen kuin niillä on mahdollisuus vaikuttaa negatiivisesti ohjelmiston käyttäjiin. (ISO 29119-1:2013, 13.)

Jos testattavaa ohjelmistoa tai sen komponentteja suoritetaan testauksen yhteydessä, testausta kutsutaan dynaamiseksi testaukseksi. Staattiseksi testaukseksi kutsutaan testausta, jossa testattavaa ohjelmistoa arvostellaan joitain laatuvaatimuksia vastaan ohjelmistoa tai sen komponentteja suorittamatta. Esimerkiksi lähdekoodin katselmointi on staattista testausta. (Foundation Level Syllabus 2018, 13.)

Testauksella voi olla monia päämääriä. Testauksella voidaan rakentaa luottamusta projektin laatuun, ehkäistä vikoja, löytää virheitä sekä täyttää laillisia, sopimuksellisia tai sääntelyn mukaisia vaatimuksia. Testin päämäärä riippuu testauskohteesta, testaustasosta sekä ohjelmistokehityksen elinkaarimallista. (mts. 13.)

2.2 Toiminnallinen testaus

Toiminnallisessa testauksessa arvioidaan sellaisia toimintoja, joita testattavan kohteen tulisi voida suorittaa. Toiminnot voivat olla erikseen määriteltyjä, käyttötapausten mukaisia tai joskus jopa dokumentoimattomia. Kyseiset toiminnot määrittelevät, mihin ohjelmistoa voi käyttää.

Toiminnallista testausta tulisi tehdä jokaisella testaustasolla. Testauksen tasoa voidaan mitata toiminnallisten testien kattavuudella. Testikattavuus ilmaistaan testattujen elementtien määränä prosentteina verrattuna elementtien kokonaismäärään. Kun testit ovat jäljitettävissä, voidaan testikattavuutta käyttää toimintojen testaukseen liittyvien vajausten havaitsemiseen. Toiminnallisten testien suunnitteluun saatetaan tarvita erityisosaamista tai testattavan järjestelmän erityistä tuntemusta. Esimerkiksi testattaessa tietokonepelin interaktiivisia elementtejä testaajan on oltava selvillä komponentin roolista ohjelmiston osana. (Foundation Level Syllabus 2018, 39-40.)

2.3 Ei-toiminnallinen testaus

Ei-toiminnallisessa testauksessa arvoidaan testattavan kohteen erinäisiä ominaisuuksia, kuten käytettävyyttä, suorituskykyä tai tietoturvallisuutta. Ei-toiminnallisessa testauksessa otetaan selville, miten ”hyvin” ohjelmisto toimii.

Ei-toiminnallista testausta tulisi suorittaa jokaisella testaustasolla ja se pitäisi aloittaa mahdollisimman aikaisin. Ei-toiminnallisten vikojen löytyminen ohjelmistosta kehityksen loppuvaiheilla saattaa olla erittäin vaarallista projektin onnistumisen kannalta.

Testikattavuutta voidaan mitata samaan tapaan kuin toiminnallisessa testauksessa. Testikattavuus ilmaistaan prosentuaalisena arvona testattujen elementtien määrästä verrattuna elementtien kokonaismäärään. Ei-toiminnallisia testejä suunniteltaessa tulee ottaa huomioon erityisesti testattavaa ohjelmistoa koskettavat seikat, kuten yleiset haavoittuvuudet ohjelmistossa käytettävässä ohjelmointikielessä tai ohjelmiston käyttäjäkunnan erityistarpeet. (Foundation Level Syllabus 2018, 40.)

2.4 Testaustasot

2.4.1 Yksikkötestaus

Yksikkötestauksessa keskitytään sellaisenaan testattavissa olevien komponenttien testaamiseen. Yksikkötestaus suoritetaan yleensä erillisenä itse järjestelmästä. Tästä syystä yksikkötestausta varten saatetaan tarvita virtualisoituja ympäristöjä tai testausta varten erikseen generoitua dataa. Testauksessa keskitytään esimerkiksi ohjelmiston luokkiin ja tietorakenteisiin tai järjestelmän komponentteihin ja moduuleihin.

Yksikkötestaus voi olla toiminnallista tai ei-toiminnallista. Esimerkiksi algoritmin suorittamien laskelmien tulosten oikeellisuuden testaaminen on toiminnallista yksikkötestausta, kun taas muistivuotoja etsivä yksikkötesti on esimerkki ei-toiminnallisesta yksikkötestauksesta. Yksikkötesteissä pyritään löytämään ja korjaamaan yksittäisissä komponenteissa esiintyviä virheitä sekä estää niiden karkaaminen korkeammille testitasoille. Virheet korjataan heti niiden löytyessä, monesti ilman erillistä virheenkäsittelyprosessia.

Automatisoidut yksikkötestit voivat olla osa ohjelmiston regressiotestausta. Regressiotestausta tehdään, kun ohjelmistossa on tapahtunut muutoksia ja halutaan varmistaan, etteivät muutokset ole rikkoneet aikaisemmin toimineita komponentteja tai toiminnallisuuksia.

Yksikkötestit suorittaa yleensä testattavan koodin kirjoittannut kehittäjä. Yksikkötesteissä löydettyjen vikojen tulisi vaikuttaa ohjelmiston jatkokehitykseen. Testit ajetaan usein ohjelmistokoodin kirjoittamisen jälkeen. Ketterässä kehityksessä yksikkötestit saatetaan kuitenkin kirjoittaa jo ennen testattavaa koodia. (Foundation Level Syllabus 2018, 31-32.)

2.4.2 Integraatiotestaus

Integraatiotestauksessa keskitytään komponenttien tai järjestelmien vuorovaikutukseen. Integraatiotestaus voidaan luokitella komponenttien integraatiotestaukseen, jossa testataan eri komponenttien vuorovaikutusta, sekä järjestelmäintegraatiotestaukseen, jossa testataan eri järjestelmien yhteistoimintaa. Komponenttien integraatiotestaus suoritetaan yksikkötestien jälkeen. Tämä testausprosessi on usein automatisoitu. Järjestelmäintegraatiotestauksen haasteisiin kuuluu sellaisten järjestelmien yhteensopivuuden testaaminen, joita ei kehitetä samassa organisaatiossa, kuten esimerkiksi kahden eri verkkopalvelun rajapinnat. Tällöin testien epäonnistumisen aiheuttava virhe saattaa sijaita toisen organisaation ylläpitämässä järjestelmässä ja virheenkorjaus voi olla hankalaa. (Foundation Level Syllabus 2018, 32.)

Integraatiotestauksessa tulisi keskittyä nimenomaan komponenttien yhteistoimintaan ja jättää yksittäisten komponenttien toiminnan testaaminen yksikkötestauksen tasolle. Integraatiotestaus voi olla toiminnalista tai ei-toiminnallista. Integraatiotestit voivat myös olla osana järjestelmän tai komponenttien regressiotestausta. (mts. 32.)

Integraatiotestauksessa tulee ottaa huomioon testattavan järjestelmän koko ja monimutkaisuus. Mikäli järjestelmä on monimutkainen ja sisältää useita yhdessä toimivia komponentteja, integraatiotestien laajuutta tulee kontrolloida. Liian laajat integraatiotestit tekevät virheiden löytämisestä ja korjauksesta vaikeaa. Tästä syystä jatkuva integraatio (Continuous Integration tai CI) on yleistynyt. Jatkuvassa integraatiossa ohjelmistoa integraatiotestataan joka kerta, kun ohjelmistoon lisätään uutta koodia tai komponentteja. Komponenttien yhteistoimintaa regressiotestataan useilla eri testaustasoilla, monesti automatisoidusti. (mts. 33.)

2.4.3 Järjestelmätestaus

Järjestelmätestauksessa testataan koko järjestelmän toiminnallisia ja ei-toiminnallisia ominaisuuksia. Monesti järjestelmätestauksessa keskitytään sellaisiin tehtäviin, joita järjestelmän on tarkoitus suorittaa. Testejä ajaessa tutkitaan näiden tehtävien toiminnallista suorittamista sekä järjestelmän ei-toiminnallista käyttäytymistä tehtävien suorittamisen aikana. Järjestelmätestausympäristön tulisi vastata mahdollisimman läheisesti järjestelmän lopullista käyttöympäristöä.

Järjestelmätestausta suorittaa järjestelmätestaukseen perehtynyt testaaja, joka ei välttämättä ole osallistunut järjestelmän kehitykseen. Tästä syystä järjestelmän vaatimusten ja toiminnallisuuksien tarkka dokumentointi on tärkeää. Jos vaatimukset ovat epäselviä tai niitä ei ole dokumentoitu, järjestelmätestauksessa saattaa ilmetä virheitä, jotka myöhemmin todetaan halutuksi toiminnallisuudeksi. Testaaja saattaa myös jättää raportoimatta testauksessa ilmenneitä virheitä, jos testaaja olettaa niiden olevan osa haluttua toiminnallisuutta. Testauksen ja testaajien sisällyttäminen ohjelmistokehitykseen mahdollisimman aikaisin vähentää tällaisten tapahtumien todennäköisyyttä. (Foundation Level Syllabus 2018, 34-36.)

2.4.4 Hyväksyntätestaus

Hyväksyntätestauksessa varmistetaan, että järjestelmä tai tuote toimii täysin odotetulla tavalla ja on valmis siirrettäväksi tuotantoon. Hyväksyntätestauksessa keskitytään enemmän järjestelmän käyttövalmiuden arviointiin kuin virheiden löytämiseen. Useiden virheiden löytymistä hyväksyntätestausvaiheessa voidaan jopa pitää joissain tapauksissa suurena riskinä projektin onnistumisen kannalta. Hyväksyntätestaus jaotellaan yleisesti nejään eri kategoriaan. (Foundation Level Syllabus 2018, 36.)

Käyttäjän hyväksyntätestauksessa varmistetaan, että järjestelmä toimii odotetulla tavalla myös käyttäjän näkökulmasta. Testaus suoritetaan todellisessa tai todellista simuloivassa toimintaympäristössä. Testauksen päämääränä on rakentaa luottamusta järjestelmän kykyyn vastata käyttäjien tarpeisiin sekä täyttää järjestelmälle määritellyt vaatimukset. (mts. 36.)

Käyttöön soveltuvuuden hyväksyntätestauksen suorittaa järjestelmänvalvoja tai muu järjestelmää ylläpitävä henkilö tai henkilöt. Testausympäristö on todellinen tai todellisuutta läheisesti simuloiva. Käyttöön soveltuvuuden testauksessa voidaan testata esimerkiksi järjestelmän päivittämistä tai järjestelmän palauttamista toimintaan varmuuskopioiden avulla. Testauksen tarkoituksena on arvioida järjestelmän ylläpidettävyyttä ja toimintaa sekä normaaleissa että poikkeuksellisissa olosuhteissa. (mts. 36-37.)

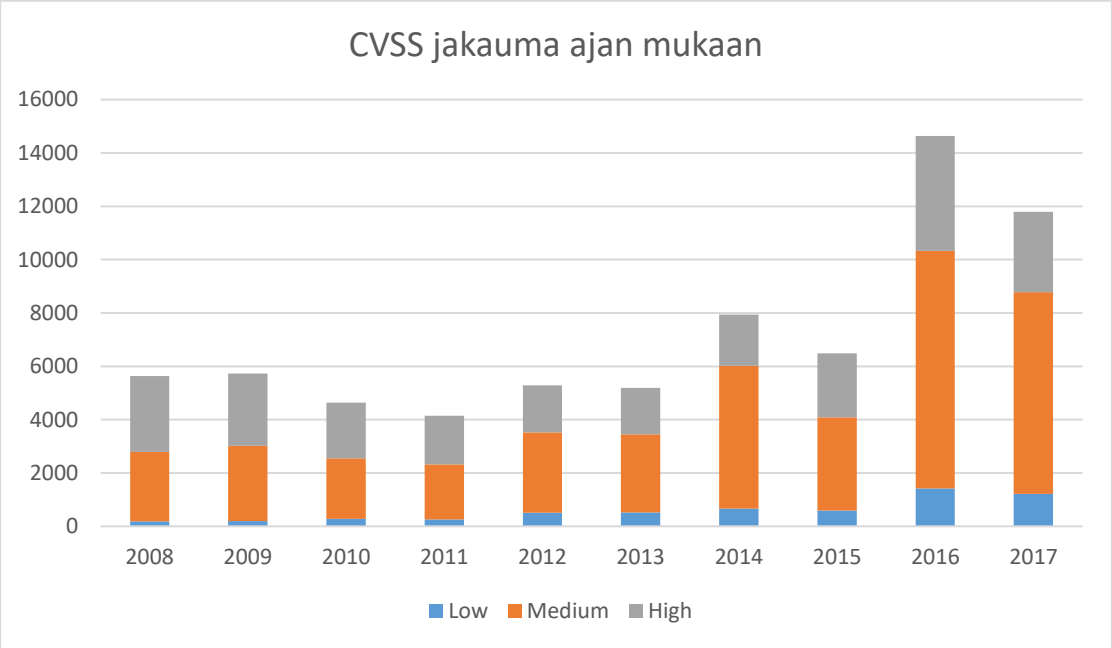
Sopimuksiin tai säännöksiin perustuvan hyväksyntätestauksen tarkoituksena on varmistaa, että järjestelmä tai tuote vastaa sopimuksissa tai säännöksissä määriteltyjä vaatimuksia. Säännöksiin perustuvassa testauksessa testit saatetaan suorittaa jonkin sääntelyviraston edustajan valvonnan alaisuudessa. (mts. 37.)

Alpha- ja betatestausta käytetään yleensä sellaisissa ohjelmistokehitysprojekteissa, joiden kehittäjät haluavat palautetta tuotteen käyttäjiltä jo ennen tuotteen siirtämistä tuotantoon. Alphatestausta tehdään ohjelmistoa kehittävän organisaation tiloissa, kun taas betatestaus suoritetaan testaajien omissa tiloissa. Testaajina toimivat ohjelmiston käyttäjät tai ylläpitäjät. Alpha- ja betatestauksen tarkoituksena on testata tuotteen käytettävyyttä käyttäjien näkökulmasta sekä löytää tuotteessa olevia virheitä. (mts. 37-38.)

Hyväksyntätestausta pidetään yleensä ohjelmistokehityksen elinkaaren viimeisenä vaiheena. Hyväksyntätestausta voidaan kuitenkin suorittaa esimerkiksi uuden toiminnallisuuden lisäämisen jälkeen ennen järjestelmätestausta. (mts. 39.)

2.5 Ohjelmistotietoturvatestauksesta

Ohjelmistotietoturvatestaus on välttämätön osa ohjelmistokehityksen elinkaarta. Ohjelmistovirheiden ja haavoittuvuuksien esiintyminen on hyväksytty luonnolliseksi osaksi ohjelmistokehitystä. Ohjelmista löytyvien haavoittuvuuksien määrä on kaksinkertaistunut kahden viimeisen vuoden aikana verrattuna edellisvuosiin (ks. kuvio 1). On myös havaittu, että haavoittuvuuksien korjaaminen jälkituotannossa on usein kallista ja aikaavievää. Ohjelmiston tietoturvan jatkuvan parantamisen katsotaan parhaiten varmistavan, että ohjelmisto on vastustuskykyinen hyökkäyksille heti sen tullessa tuotantoon. (Dutta 2015)



Kuvio 1. CVSS jakauma ajan mukaan

Tietoturvallinen ohjelmisto käyttäytyy odotetusti silloinkin, kun sitä yritetään tarkoituksenmukaisesti rikkoa tai saada käyttäytymään odottamattomalla tavalla. Ohjelmistotietoturvatestauksessa pyritään toimimaan kuten potentiaalinen hyökkääjä saattaisi toimia pyrkiessään rikkomaan tai käyttämään hyväkseen ohjelmiston toiminnallisuuksia. (McGraw & Potter 2004, 81.)

2.6 Tietoturvatestausmenetelmiä

2.6.1 Black, white ja gray box -testaus

Khanin ja Khanin mukaan (2012, 12) tärkeimmät menetelmät, joita käytetään ohjelmistovirheiden löytämiseen ovat white box, black box ja gray box -testaustekniikat. White box -testauksessa tutkitaan perin pohjin testattavan kohteen lähdekoodia ja sisäistä logiikkaa. Testaajalla on käytettävissään kohteen koko lähdekoodi. Black box -testauksessa sen sijaan testattavaa kohdetta lähestytään ilman mitään tietoa sen sisäisestä toiminnasta. Testaajalla ei ole käytettävissä kohteen lähdekoodia lainkaan. Gray box -testauksessa on tarkoitus yhdistää white box ja black box testaustekniikoiden ominaisuuksia. Testaaja tuntee järjestelmän ja voi tutkia sen lähdekoodia suunnitellakseen testitapauksia. Testit kuitenkin suoritetaan kohteen ulospäin osoittavia rajapintoja vastaan, kuten black box -testauksessa tehtäisiin. (mts. 12-14.)

2.7 Tietoturvatestaustekniikoita

2.7.1 Käsin tehty tarkastus

Käsin tehdyssä tarkastuksessa tutkitaan ohjelmiston rakennetta, teknologisia ratkaisuja sekä menettelytapoja manuaalisesti. Tarkastuksessa saatetaan käyttää ohjemiston dokumentaatiota ja haastatella ohjelman toteutukseen osallistuneita henkilöitä. Tällä tavalla tehty testaus on konseptina yksinkertainen, mutta tehokas. Monesti tieto siitä, miksi jokin ohjelmiston ominaisuus on implementoitu juuri tietyllä tavalla paljastaa siinä mahdollisesti piileviä haavoittuvuuksia. Käsin tehdyssä tarkastuksessa voidaan myös varmistaa, että ohjelmisto on koodattu nuodattaen turvallisia ohjelmointikäytäntöjä. (OWASP Testing Guide 2015.)

2.7.2 Staattinen analyysi

Staattisessa analyysissä ohjelmiston lähdekoodia tutkitaan manuaalisesti. Monet haavoittuvuudet ovat löydettävissä ainoastaan tällä tavalla. Staattista analyysiä pidetään tarkimpana testaustekniikkana, mutta se vaatii erittäin taitavia testaajia, jotka kykenevät havaitsemaan koodissa esiintyviä virheitä ja ymmärtävät miten ne vaikuttavat ohjelmiston turvallisuuteen. Lähdekoodin tutkiminen on erityisen tehokas tapa löytää ohjelmistosta syötteiden tarkistukseen liittyviä virheitä. Analyysiä suorittaessa testaajan tulee kuitenkin pitää mielessä, että tutkittava koodi ei välttämättä mene sellaisenaan tuotantoon. (OWASP Testing Guide 2015.)

2.7.3 Penetraatiotestaus

Penetraatiotestaus tai eettinen hakkerointi on kohteen testaamista hyökkääjän näkökulmasta. Testaaja saa kohteeseen samanlaisen pääsyn kuin tavallinen käyttäjä ja yrittää hyökkääjän tavoin löytää siitä haavoittuvuuksia. Penetraatiotestausta pidetään tehokkaana menetelmänä testattaessa verkkoturvallisuutta, mutta sen ei katsota soveltuvan samalla tavalla ohjelmistotestaukseen. Verkkoturvallisuutta testattaessa tavoitteena on tunnettujen haavoittuvuuksien löytyminen ja hyväksikäyttö. Ohjelmistoa testattauessa pyritään yleensä löytämään uusia haavoittuvuuksia. Keskitettyä penetraatiotestausta voidaan kuitenkin käyttää

esimerkiksi löydetyn haavoittuvuuden korjauksen toimivuuden tarkastamiseen korjauksen jälkeen. (OWASP Testing Guide 2015.)

2.7.4 Uhkamallinnukseen perustuva testaus

Uhkamallinnus auttaa ohjelmistokehittäjiä ennakoimaan kohdetta mahdollisesti tulevaisuudessa uhkaavat tietoturvaongelmat. Uhkamallinnus helpottaa resurssien suuntaamista ohjelmiston todennäköisimmin haavoittuvaisiin osa-alueisiin. Uhkamallin rakentamista ja dokumentointia suositellaan kaikille kehitettäville ohjelmistoille ja niitä tulisi päivittää projektin edetessä. Uhkamallinnusta voi tehdä useammalla eri tavalla. (OWASP Testing Guide 2015.)

Esimerkiksi Microsoft on luonut oman metodinsa uhkien luokitteluun uhkamallinnuksen yhteydessä. Tälle metodille annettu nimi STRIDE tulee sanoista Spoofing, Tampering, Repudiation, Information disclosure, Denial of service ja Elevation of privilege. STRIDE-kategoriat eivät kuitenkaan ole täsmällisiä, ja eri uhkat saattavat kuulua useaankin STRIDE-kategoriaan. (Osterman 2007)

2.7.5 Fuzzaus

Fuzzauksessa testattavalle kohteelle annetaan odottamattomia tai rikkinäisiä syötteitä sen ulkoisia rajapintoja käyttäen. Fuzzauksen tarkoitus on löytää kohteesta tietoturvahaavoittuvuuksia, palvelunestohyökkäysmenetelmiä tai muita ei-toivottuja ominaisuuksia. Fuzzauksessa käytettävät syötteet ovat satunnaisia tai lähes satunnaisia. Näin vältytään tekemästä oletuksia kohteen toiminnasta ja keskitytään ainoastaan siihen, miten kohde reagoi annettuihin syötteisiin. Ohjelmia ja ohjelmistokehyksiä, joilla suoritetaan fuzzausta, kutsutaan fuzzereiksi. (Takanen, DeMott & Miller 2008, 24-25.)

Eräs tapa luokitella fuzzereita on jaotella ne testitapausten monimutkaisuuden mukaan. Näin syntyy neljä eri fuzzeriluokkaa. Staattinen sapluunapohjainen fuzzeri ei ole tietoinen testattavan kohteen rakenteesta tai ominaisuuksista. Sitä käytetään pääasiassa tiedostoformaattien ja erittäin yksinkertaisten protokollien testaamiseen. Lohkoihin perustuva fuzzeri implementoi testattavan protokollan tai järjestelmän perusrakenteen. Se saattaa myös sisältää alkeellisia dynaamisia toiminnallisuuksia, kuten esimerkiksi tarkistussummien automaattisen laskemisen. Dynaamiseen

syötteiden kehittämiseen pohjautuva fuzzeri ei välttämättä aluksi tunne testattavaa kohdetta tai protokollaa, mutta oppii luomaan syötteitä perustuen kohteen palautteen perusteella. Malli- tai simulaatiopohjainen fuzzeri toteuttaa testattavan kohteen tai protokollan täysin tai simuloi hyvin tarkasti sen toimintaa. Tällainen fuzzeri voi muutella syötteiden rakennetta sekä lähettää odottamattomia viestiketjuja. (mts. 27.)

2.8 Ratkaisun toteutukseen valitut menetelmät ja tekniikat

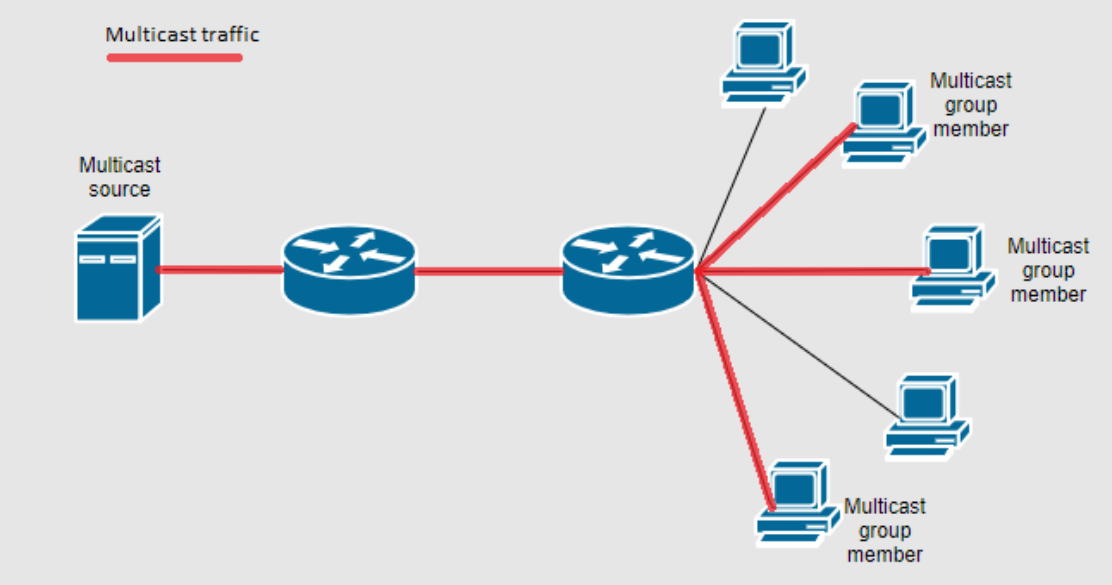
Ratkaisun kehittämisessä käytettiin gray box -lähestymistapaa. Testattava kohde oli täysin testaajan tutkittavissa, mutta testitapaukset suunniteltiin koettelemaan ainoastaan testattavan kohteen ulospäin näkyviä rajapintoja. Rikkovissa ei-toiminnallisissa testeissä asetettiin testikohteelle lähetettävien pakettien kenttiin odottamattomia tai keskenään ristiriidassa olevia arvoja. Vaihtoehtoisesti kaikki tai osa pakettien kentistä fuzzattiin täyttämällä kenttien arvot satunnaislukugeneraattorilla. Ratkaisu voidaan luokitella malli- tai simulaatiopohjaisten fuzzereiden kategoriaan.

3 Testattavat protokollat

Toimeksiannossa testattaviksi protokolliksi oli määritelty IGMP ja PIM, tarkemmin IGMP versiot 2 ja 3 sekä PIM-SM. IGMP ja PIM ovat ryhmälähetykseen liittyviä protokollia. Niiden toiminnan ymmärtäminen edesauttoi ratkaisun kehittämisessä.

3.1 Multicast

Multicast tarkoittaa sitä, että mahdollisesti useammalle vastaanottajalle tarkoitetut datapaketit lähetetään vain kerran ja ne monistetaan tarvittaessa (Kuvio 2). Jos päätelaite haluaa vastaanottaa multicast-liikennettä, se ilmoittaa reitittimelle kuuluvansa kyseisen lähetyksen vastaanottajiin. Yleisemmässä yksittäislähetyksessä eli unicastissa paketit lähetetään erikseen jokaiselle vastaanottajalle. Multicastilla voidaan vähentää verkon linkkien kuormitusta, ja se soveltuu hyvin esimerkiksi multimedian jakamiseen useille eri vastaanottajille. (Noppari 2008, 8.)



Kuvio 2. Multicast-lähetys

IANA on määritellyt IPv4 multicast -paketeille oman osoiteavaruutensa, 224.0.0.0-239.255.255.255. Tämä avaruus on taas jaettu lohkoihin osoitteiden käyttötarkoituksen mukaan. Toimeksiannon kannalta relevantti osoitelohko on Local Network Control Block tai lähiverkkolohko 224.0.0.0/24. Taulukossa 1 luetellaan muutamia IGMP- ja PIM-protokollien kannalta olennaisia multicast-osoitteita. Lähiverkkolohko on sellaista multicast-ohjausliikennettä varten, jonka on tarkoitus pysyä paikallisessa verkossa. (RFC 5771:2010, 3.)

Taulukko 1. Multicast-osoitteita

Osoite	Kuvaus
224.0.0.1	Kaikki aliverkon laitteet
224.0.0.2	Kaikki aliverkon reitittimet
224.0.0.13	Kaikki PIM reitittimet
224.0.0.22	IGMPv3 Membership Report -osoite

Multicast -liikenteen reitittämistä varten OSI-mallin tasolla 2 IANA on määritellyt multicast -liikenteelle oman MAC-osoitelohkonsa. Tämän osoitelohkon osoitteet alkavat aina IANA OUI:lla 00-00-5E (hexadesimaalinen merkintätapa) ja päättyvät multicast osoitteen 23:een vähiten merkitsevään bittiin. Koska IP multicast -

osoitteilla on 28 merkitsevää bittiä, useampi kuin yksi osoite saattaa saada saman multicast MAC-osoitteen. (RFC 1112:1989, 5.)

3.2 IGMP

Päätelaitteet käyttävät Internet Group Management Protocol (IGMP) -protokollaa kertoakseen läheisille verkkolaitteille, mihin multicast-ryhmiin kyseiset päätelaitteet kuuluvat. Verkkolaitteet ottavat selvää, mihin niiden fyysisiin rajapintoihin on kytköksissä päätelaitteita, jotka kuuluvat johonkin multicast-ryhmään. Verkkolaitteet ylläpitävät listaa ryhmien jäsenyyksistä.

3.2.1 Verkkolaitteiden roolit

Verkkolaitteilla voi IGMP-protokollan mukaan olla kaksi roolia: tiedusteleva ja ei-tiedusteleva. Liittyessään uuteen verkkoon kaikki verkkolaitteet aloittavat kyselijöinä. Jos ne kuulevat samassa verkossa olevan matalamman IP-osoitteen omaavan verkkolaitteen lähettävän Membership Query -viestin, ne luopuvat rooleistaan ja muuttuvat ei-tiedusteleviksi. (RFC 2236:1997, 3.)

Päätelaitteet vastaavat tiedusteluihin raportoimalla verkkolaitteille ne ryhmät, joiden jäseniä ne ovat tai joihin ne haluavat liittyä lähettämällä Membership Report -viestin. Poistuessaan ryhmästä päätelaitteet ilmoittavat asiasta verkkolaitteille lähettämällä Leave Group -viestin.

3.2.2 IGMP viestien rakenne

IGMP viestit enkapsuloidaan IP-pakettien sisään. IGMPv2 viesteissä on neljä kenttää: Type, Max Resp Time, Checksum ja Group Address (Kuvio 3). IGMP-version 1 viestit ovat muuten samanlaisia, mutta Max Resp Time -kenttää ei käytetä mihinkään ja se jätetään nollaksi. Type-kenttä kertoo, onko kyseessä tiedustelu, raportti vai ryhmästä poistumisilmoitus. Max Resp Time-kenttä määrittelee sen ajan pituuden, jonka aikana päätelaitteen tulee ilmoittaa kuulumisestaan johonkin ryhmään reitittimien asiaa tiedustellessa. Checksum-kenttä on tarkistussumma, joka lasketaan ennen viestin lähettämistä ja tarkistetaan viestin saapuessa. Group Address -kenttä saa eri merkityksen riippuen siitä, minkä tyyppisessä viestissä sitä käytetään. Jos viesti on

tyyppiä Membership Query ja kentän arvo on nolla, tulkitaan viesti yleiseksi tiedusteluksi. Jos kentän arvo on jonkin tietyn ryhmän osoite, tulkitaan viesti tälle ryhmälle tarkoitetuksi. Viestin ollessa tyyppiä Membership Report, Group Address -kentässä on sen ryhmän osoite, johon päätelaite ilmoittaa kuuluvansa. (RFC 2236:1997, 1.)

tavu	1	2	3	4
paketti	Type	Max Resp Time	Checksum	
	Group Address			

Kuvio 3. IGMPv2 paketti

Membership Report -viestejä on kolmea eri tyyppiä, yksi jokaiselle IGMP:n versiolle. Näin reititin tietää, mitä IGMP:n versiota päätelaite käyttää.

IGMP:N versiossa 3 laajennettiin Membership Query -viestin toiminnallisuutta kattamaan myös lähde- ja ryhmäkohtaiset tiedustelut. IGMPv3 Membership Query -viesteissä on Number of Sources -kenttä, joka kertoo, montako lähdeosoitetta viesti sisältää. Mikäli lähteiden määrä on nolla, tiedustelu voi olla yleinen tai tietylle ryhmälle tarkoitettu. Mikäli lähteitä on yksi tai enemmän, on kyseessä lähde- ja ryhmäkohtainen tiedustelu. (RFC 3376:2002, 9.)

3.3 PIM

3.3.1 Yleistä

Protocol Independent Multicast (PIM) -protokolla käyttää jo olemassa olevaa reititysinformaatiota hyväkseen reitittääkseen multicast-liikenettä lähteistä päätelaitteille. PIM voi toimia kahdessa toimintatilassa: dense tai sparse. Dense-tilassa protokolla uskoo jokaisen päätelaitteen haluavan kuulla multicast-liikenettä ja karsii niitä vastaanottajien listalta sitä mukaan, kun päätelaitteet ilmaisevat, että ne eivät halua vastaanottaa kyseistä liikennettä. Sparse-tilassa päätelaitteille ei lähetetä multicast -liikennettä, elleivät ne pyydä sitä erikseen. (IP Multicast Technology

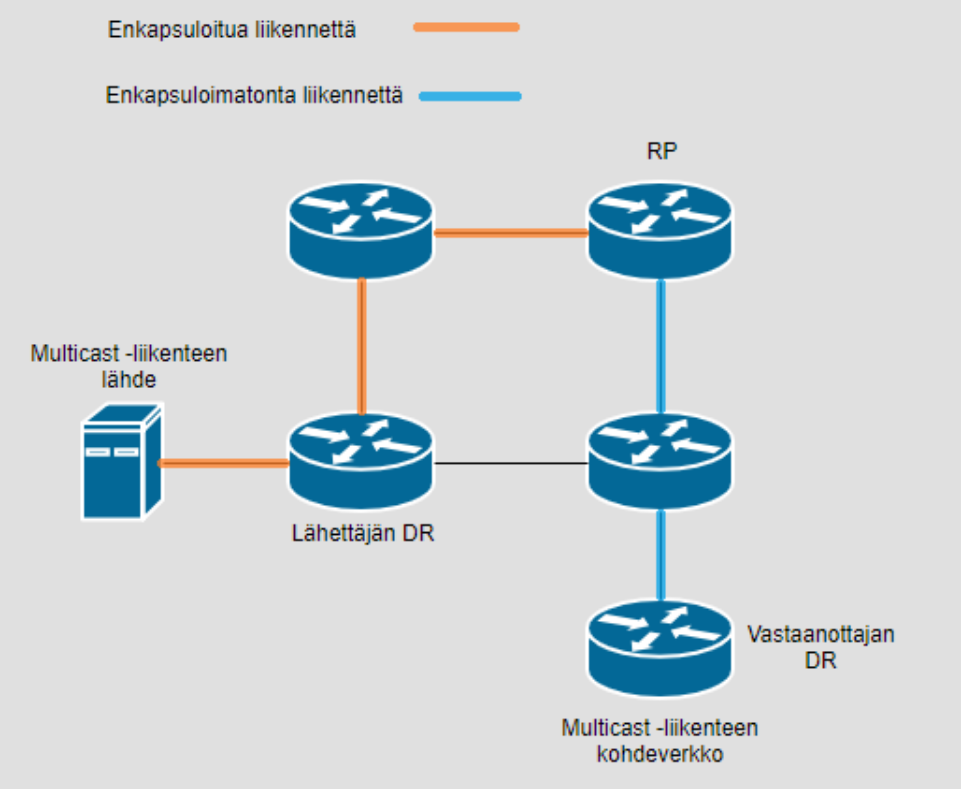
Overview 2009.) Toimeksiannon mukaisesti tässä opinnäytetyössä keskitytään PIM Sparse Mode (PIM-SM) -toimintatilaan.

PIM-SM-protokollan toiminta alkaen päätelaitteen halukkuudesta vastaanottaa tietylle ryhmälle suunnattua multicast-liikennettä voidaan jakaa kolmeen vaiheeseen. Nämä vaiheet on kuvattu luvuissa 3.3.2-3.3.4.

3.3.2 Reitityspuu

Ensiksi päätelaite ilmoittaa haluavansa vastaanottaa multicast-liikennettä, yleensä lähettämällä IGMP Membership Report -viestin naapurireitittimelleen. Yksi päätelaitteen aliverkossa olevista reitittimistä valitaan Designated Router (DR) -reitittimeksi. Kuullessaan päätelaitteen pyynnön vastaanottaa multicast-liikennettä DR lähettää Join-viestin kyseisen multicast-ryhmän Rendevous Point (RP) -reitittimelle. Tämä viesti tunnetaan myös (*, G) -viestinä, sillä liikenteen lähde (*) ei ole vielä tiedossa. Viesti leviää reitittimeltä reitittimelle kohti RP:tä alustaen muut reitityslaitteet osaksi RP -reitityspuuta eli RPT:tä. Tämä RPT:n oksa säilyy verkkolaitteiden muistissa niin kauan, kuin päätelaitteen naapurireititin lähettää Join-viestejä kohti RP:tä. Jos kaikki päätelaitteet poistuvat kyseistä RPT:tä koskevasta multicast-ryhmästä, DR lähettää Prune-viestin kohti RP:tä, kertoen RP:lle ja matkalla oleville verkkolaitteille, että kyseinen RPT:n oksa halutaan karsia pois. Jos Join-viestien lähetys lakkaa, eikä Prune-viestiä jostain syystä lähetetä, reititin unohtaa RPT:n oksan tietyn ajan kuluttua.

Multicast-liikenteen lähettäjä ryhtyy yksinkertaisesti lähettämään dataa, joka on osoitettu vastaanottavalle ryhmälähetysryhmälle. Lähettäjän DR enkapsuloi paketit Register-paketeiksi, jotka lähetetään kohti RP:tä. RP reitittää paketit RPT:n (*, G) oksaa pitkin päätelaitteille. Toimintaa havainnollistetaan kuviossa 4. (RFC 7761:2016, 7.)



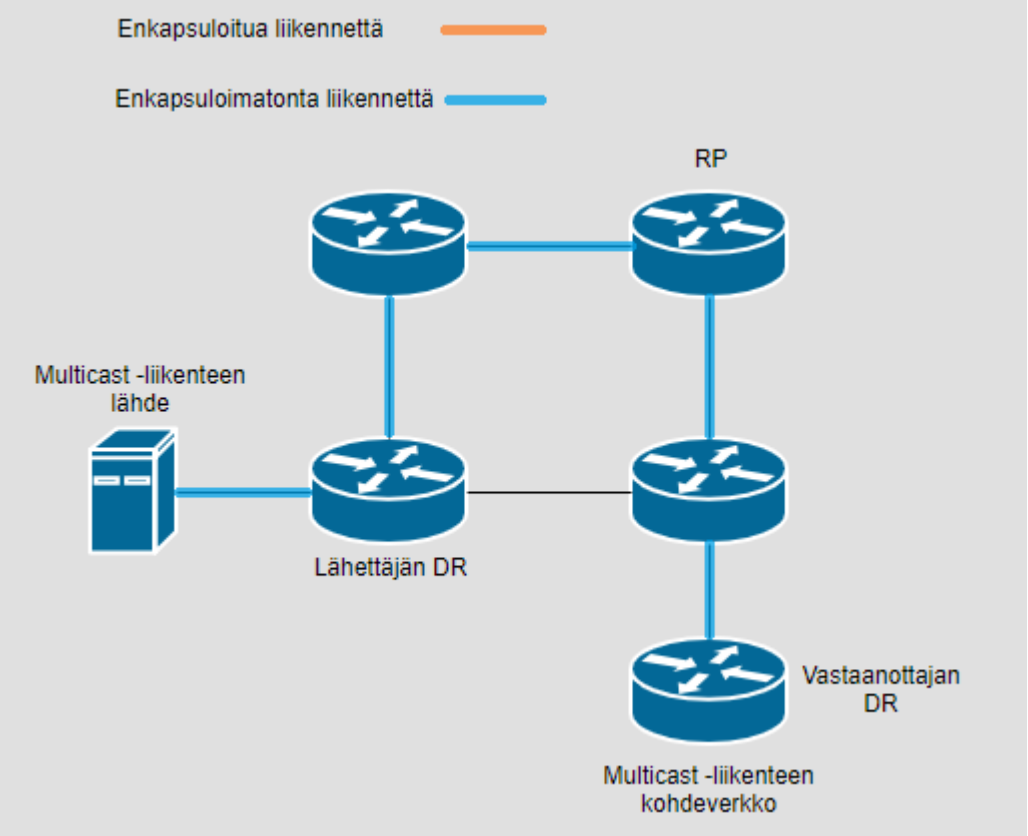
Kuvio 4. PIM-SM ensimmäinen vaihe

3.3.3 Lähdekohtainen reititys

Vaikka pakettien enkapsulointi ja kierrättäminen RP:n kautta on toimiva ratkaisu, se ei välttämättä ole kaikkein tehokkain. Pakettien enkapsulointi on verkkolaitteen resursseja verottava toimenpide ja reitti lähteestä RP:n kautta saattaa kiertää päätelaitteelle ylimääräisten hyppyjen kautta. Niinpä RP yleensä pyrkii vaihtamaan reititysjärjestelyjä niin, että lähde lähettää liikennettä päätelaitteelle enkapsuloimatta datapaketteja.

Kun RP ryhtyy vastaanottamaan Register -paketteja jollain rajapinnallaan, se lähettää (S, G) Join -viestin kohti liikenteen lähdettä. Tämä viesti kulkee verkkolaitteelta toiselle, alustaen uutta (S, G) lähdekohtaista reittiä. Tätä reittiä käytetään ainoastaan lähteestä S multicast -ryhmälle G suunnatun liikenteen reitittämiseen. Kun Join -viesti saavuttaa lähteen aliverkon tai jonkin reitittimen, jolla on jo kyseinen (S, G) -reitti tiedossaan, multicast -paketit reititetään enkapsuloimatta tätä uutta oksaa pitkin RP:lle (Kuvio 5). RP:stä eteenpäin paketit reititetään RPT:tä pitkin päätelaitteille. Kun RP huomaa vastaanottavansa samoja datapaketteja enkapsuloituna ja alkuperäisinä, se lähettää Register Stop -viestin lähteen DR:lle. Vastaanottaessaan Register Stop -

viestin DR lakkaa enkapsuloimasta ryhmälähetysliikennettä Register -viesteiksi. (RFC 7761:2016, 8.)



Kuvio 5. PIM-SM toinen vaihe

3.3.4 SPT -reititys

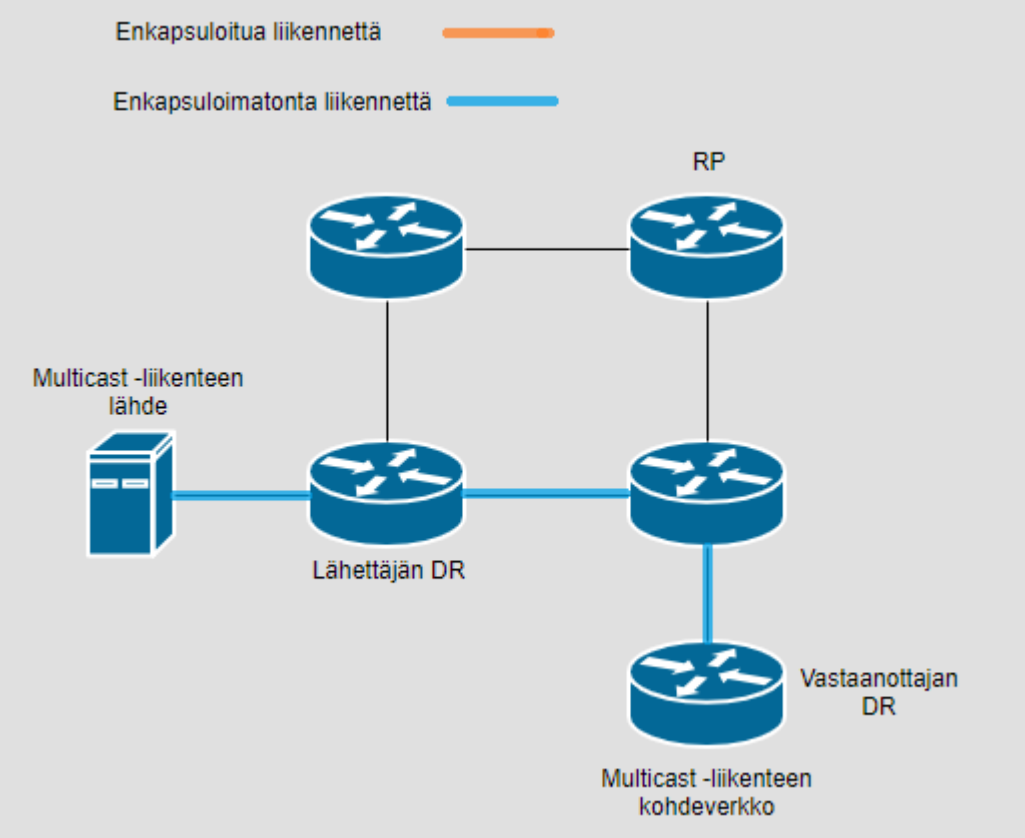
Vaikka data liikkuisikin enkapsuloimattomana lähteestä RP:n kautta päätelaitteille, pakettien reitittäminen RP:n kautta ei välttämättä ole optimaalista. Tästä syystä multicast-liikenteen vastaanottajan DR saattaa pyrkiä siirtymään RPT :stä Shortest Path Tree (SPT) -reititykseen.

Muuttaakseen reititysjärjestelyjä vastaanottajan DR lähettää lähdekohtaisen (S, G) Join-viestin kohti lähdettä S. Tämä viesti muodostaa uuden (S, G) oksan. Kun viesti saavuttaa joko lähteen aliverkon tai reitittimen, jolla on jo (S, G) reitti kyseiseen lähteeseen, (S, G) reitti on muodostettu ja datapaketit virtaavat sitä pitkin päätelaitteille (Kuvio 6).

Lähdekohtaisen (S, G) reitin muodostamisen jälkeen päätelaite tai päätelaitteesta ylävirtaan sijaitseva reititin alkaa vastaanottamaan samoja multicast-paketteja kahta eri reittiä. Vastaanottajan DR:n havaitessa tällaisen tilanteen se lähettää

lähdekohtaisen Prune-viestin kohti RP:tä. Viesti liikkuu kohti RP:tä kertoen matkan varrella oleville verkkolaitteille, ettei ryhmälähetysliikennettä (S, G) tulisi enää ohjata kyseistä RPT:n oksaa pitkin. Viesti jatkaa matkaansa kunnes se saavuttaa RP:n tai jonkin reitittimen, joka vielä reitittää samaa ryhmälähetysliikennettä omille päätelaitteilleen.

IGMP versio 3 mahdollistaa lähdekohtaisten Join ja Prune-viestien lähettämisen heti reitityksen ensimmäisessä vaiheessa. Tätä tekniikkaa kutsutaan PIM Source-Specific Multicastiksi (PIM-SSM). Kun päätelaite pyytää liikennettä tietyltä lähteeltä, sen DR voi (*, G) Join-viestin sijaan lähettää välittömästi (S, G) Join-viestin. Päätelaite voi myös kertoa haluavansa multicast-liikennettä ainoastaan silloin, jos liikenne ei ole lähtöisin tietyistä lähteistä. Tässä tilanteessa päätelaitteen DR lähettää tavanomaisen (*, G) Join-viestin, mutta saattaa lisäksi lähettää lähdekohtaisia Prune-viestejä niihin lähteisiin, joista päätelaite ei halua vastaanottaa liikennettä. (RFC 7761:2016, 9.)



Kuvio 6. PIM-SM kolmas vaihe

3.4 Liikenteen tahattoman monistumisen välttäminen

Joissain tilanteissa verkossa saatta olla useampia laitteitta, jotka reitittävät samaa ryhmälähetysliikennettä, aiheuttaen liikenteen tahatonta monistumista verkossa. PIM käsittelee tällaiset tilanteet käyttäen Assert-viestejä. Kun reitittimet havaitsevat samojen datapakettien virtaavan verkossa eri reittejä pitkin, ne äänestävät vain yhden reitittimen hoitamaan liikenteen reitittämistä. Äänestyksen voittaa se reititin, jolla on jo olemassa lähdekohtainen reitti (S, G). Jos äänestykseen osallistuvilla reitittimillä ei ole (S, G) -reittiä, äänestyksen voittaa se reititin, josta on lyhin matka joko RP:hen tai lähteeseen, riippuen siitä, onko kyseessä RPT vai lähdekohtainen reitti. (RFC 7761:2016, 10.)

3.5 BSR-mekanismi

Jotta PIM-SM reitittimet voivat lähettää Join -viestejä RP:lle, niiden on tiedettävä, mikä on kyseisen multicast -ryhmän RP:n osoite. Tämä tieto on joko konfiguroitu reitittimiin tai se selvitetään dynaamisesti. Yksi tapa on käyttää Bootstrap Router (BSR) -mekanismia.

Ensiksi reitittimet äänestävät joukostaan yhden reitittimen BSR:n rooliin. Tämän jälkeen ne lähettävät BSR:lle Candidate RP Advertisement-viestejä tietyin väliajoin. Näissä viesteissä reitittimet kertovat, minkä multicast-ryhmien RP reitittimiksi ne ovat valmiita rytymään. BSR muodostaa ehdokkaista RP joukkion, jonka se ilmoittaa Bootstrap Message-viestillä muille reitittimille. RFC 7761:2016, 11.)

4 Toteutukseen valitut teknologiat

4.1 Scapy

Scapy on IP-pakettien lähettämistä ja vastaanottamista varten luotu Python -ohjelma. Scapy mahdollistaa lähetettävien pakettien monipuolisen manipuloinnin. Käyttäjä voi vapaasti muokata kaikkia paketin kenttiä ja määritellä uusia paketteja, tarvittaessa tavu tavulta. Scapy tukee useita eri verkkoprotokollia, mukaanlukien IGMP versiot 2 ja 3. Scapy on avoimen lähdekoodin projekti.

Scapylla määritellään paketteja Scapyn omalla DSL-kielellä. Tämä kieli noudattaa kuitenkin Python-kielen syntaksia, ja Scapy käyttää tulkkinaan Python-ohjelmointikielen tulkkia. Näin Pythonin kanssa tuttujen kehittäjien ei tarvitse opetella jälleen uutta kieltä käyttääkseen Scapya. (Welcome to Scapy's documentation 2018.)

Scapy kuvailee paketit toinen toisensa päälle aseteltujen kerroksien avulla. Jokaisen kerroksen kenttiin on asetettu valmiiksi oletusarvot, jotka voidaan tarvittaessa ylikirjoittaa. Tämä mahdollistaa Scapyn joustavan käytön moniin eri tarkoituksiin. Monesti IP-paketti voidaan määritellä vain yhdellä rivillä koodia (mt.). Kuviossa 7 käytetään Scapya pingaamaan Googlen nimipalvelinta sekä tulostetaan vastaukseksi saatu ICMP echo-reply.

```
>>> ping = IP(dst="8.8.8.8")/ICMP()
>>> answer = sr1(ping)
Begin emission:
.Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> answer.show()
###[ IP ]###
  version= 4
  ihl= 5
  tos= 0x0
  len= 28
  id= 5036
  flags=
  frag= 0
  ttl= 119
  proto= icmp
  chksum= 0x5f12
  src= 8.8.8.8
  dst= 192.168.0.107
  \options\
###[ ICMP ]###
     type= echo-reply
     code= 0
     chksum= 0x0
     id= 0x0
     seq= 0x0
###[ Padding ]###
        load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'

>>>
```

Kuvio 7. Scapyn käyttöä

4.2 Robot Framework

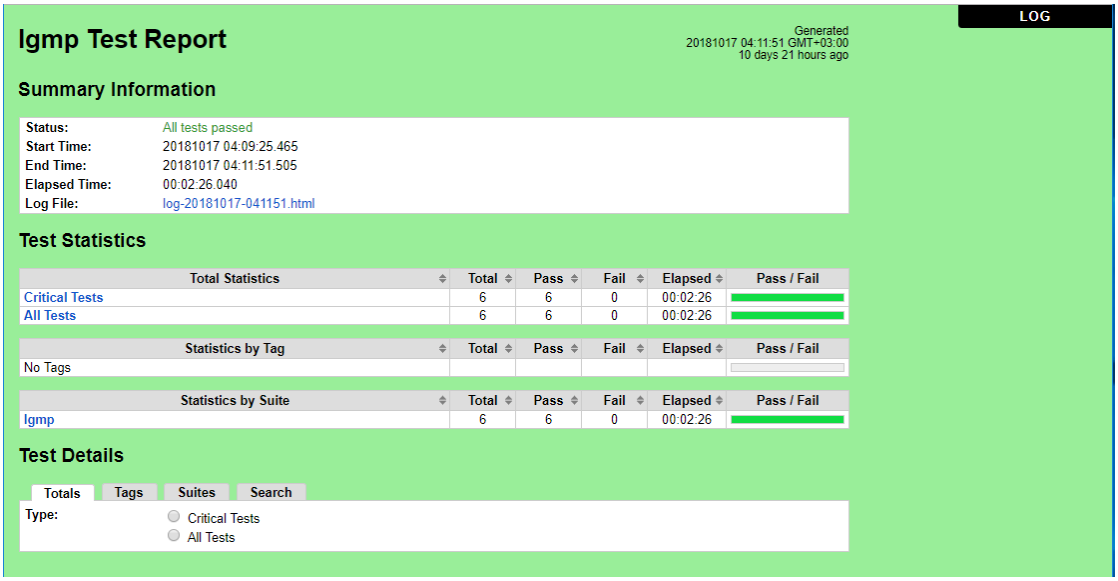
Robot Framework on Python-pohjainen ohjelmistokehys, jolla voidaan luoda hyväksyntätestejä ATDD-ympäristöissä. Testitiedostot kirjoitetaan käyttäen välilyönneillä eroteltua syntaksia ja avainsanoihin perustuvaa testaustapaa. Robot Frameworkin testauskapasiteettia on mahdollista kasvattaa kirjoittamalla uusia

testikirjastoja joko Python tai Java -ohjelmointikielillä. Robot Framework ja suurin osa sen kirjastoista on avointa lähdekoodia. (Robot Framework/ 2018.)

Robot-ohjelmistokehyksellä testit rakennetaan luomalla ensin testitiedosto. Tämä tiedosto kokonaisuudessaan muodostaa testisetin. Useampi testitiedosto samassa kansiossa muodostaa ylemmän tason testisetin. Testitiedostojen testitapaukset koostuvat avainsanoista, jotka puolestaan koostuvat matalamman tason avainsanoista tai ovat kirjastoavainsanoja. Kirjastoavainsanat ovat matalimman tason avainsanoja. Niitä käytetään testattavana olevan kohteen kanssa vuorovaikuttamiseen. Lisäksi testitiedostoon saatetaan tuoda muuttuja- tai resurssitiedostoja. (mt.)

Käytännössä Python -kielellä toteutettu Robot kirjasto on Python luokka ja kirjaston sisältämät avainsanat ovat kyseisen luokan metodeja. Testitapauksientila määräytyy sen mukaan, tapahtuuko metodin suorittamisessa poikkeuksia vai ei. Robot Framework parsii kirjastoon määritellyt metodit testitiedoston käytettäviksi avainsanoiksi.

Testien tulokset raportoidaan HTML -formaatissa. Mikäli lokitiedosto on luotu, siihen luodaan linkki suoraan raportista. Kun kaikki testit onnistuvat, raportin taustaväri on vihreä (Kuvio 8). Muussa tapuksessa raportti on punainen. Raportin ominaisuuksia voi hienosäätää komentoriviparametreilla.



Kuvio 8. Robot raportti onnistuneesta testisetistä

4.3 Jenkins

Jenkins on avoimen lähdekoodin automatisointiserveri. Sen avulla voi helposti rakentaa CI-ympäristöjä ja automatisoida kehitystyössä yleisiä tehtäviä. Jenkins on saatavilla Docker konttina ja asennuspakettina monille eri käyttöjärjestelmille. Jenkinsiä käytetään web-käyttöliittymän kautta. Sen toiminnallisuutta voi laajentaa Jenkins-yhteisön julkaisemien pluginien avulla. Jenkins on kirjoitettu Java-ohjelmointikielellä. (Heller, 2018.) Toimeksiannossa käytettävään Jenkins -automatisointiserveriin oli asennettu Robot Framework Plugin, jonka avulla testaaja voi helposti tutkia Robot Frameworkin luomia testiraportteja Jenkinsin käyttöliittymästä käsin. Plugin mahdollistaa myös testistatistiikan piirtämisen.

5 Ratkaisun toteutus

5.1 Vaatimukset

Testit ajetaan koneelta, johon on asennettu Linux Debian 9.4. Ratkaisu on riippuvainen Scapy- ja Robot Framework Python-kirjastoista. Testattava kohde oli IGMP ja PIM-protokollat toteuttava ohjelmisto.

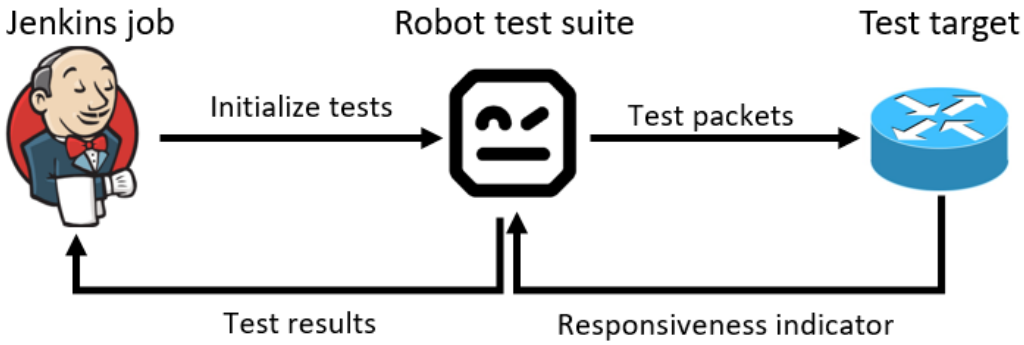
Ratkaisu tuli toteuttaa siten, että testien ajaminen ja tulosten tulkitseminen oli automatisoitavissa käyttäen Robot Framework-testauskehystä. Tarvittavat tiedot testattavasta kohteesta kuten sen IP-osoite tai käytettävä verkkorajapinta täytyi olla parametrisoitavissa. Testit tuli implementoida ja olla rakennettavissa käyttäen Robot Framework-tyylisiä avainsanoja. Ratkaisun tuli olla integroitavissa Jenkins-automatisointiserverille.

Testitapausten tuli olla destruktiivisia. Lähetettävien pakettien tuli olla joko viallisia rakenteeltaan tai muuten rikkoa niillä implementoitavaa protokollaa. Uusien räätälöityjen testitapausten luominen käyttäen hyväksi luotuja Robot-avainsanoja piti olla mahdollista. Ratkaisun implementaation ja joidenkin testitapausten luonteen vuoksi ratkaisulla kyettiin myös lähettämään valideja paketteja.

Testien tuli joko onnistua tai epäonnistua riippuen siitä, jäikö testattava kohde testin jälkeen responsiiviseksi vai ei. Mikäli testi epäonnistui, sen aiheuttaneet paketit tuli

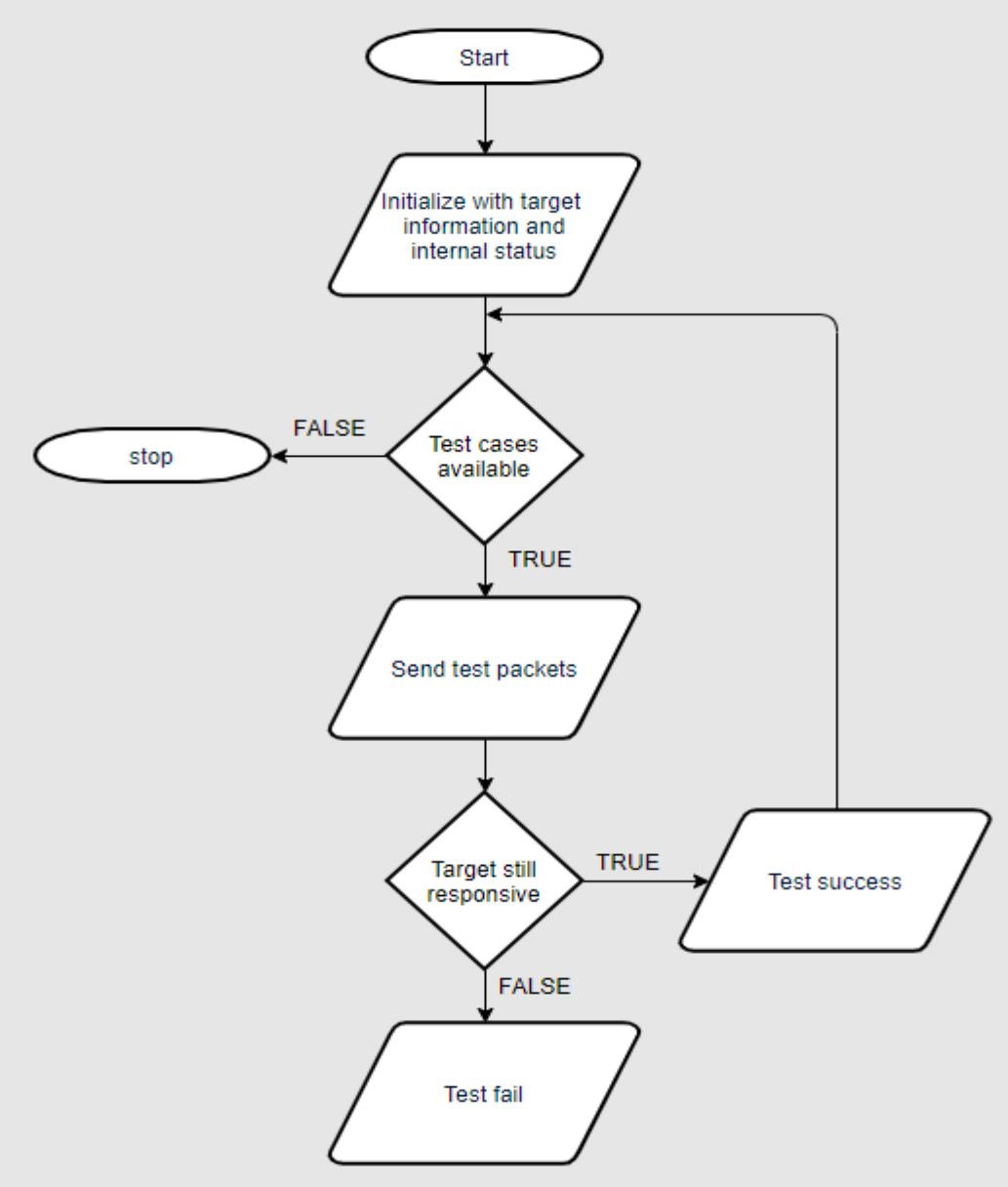
tallentaa ja arkistoida. Testiraportit tuli muodostaa Robot Frameworkin raportointiominaisuutta hyväksi käyttäen.

Koska ratkaisun kehittäminen toteutettiin ketteränä ohjelmistokehityksenä, suunnittelua ja toteutusta tapahtui vuorotellen koko projektin ajan. Ylätasolla projektin rakenne on kuitenkin varsin yksinkertainen (Kuvio 9). Jenkins -automatisointiserveri käynnistää Robot-testisetin, jonka sisältämät testisetit ajetaan läpi. Testisetit on ajastettu käynnistymään tietyin väliajoin. Robot luo testeistä testiraportit, jotka arkistoidaan ja joita voidaan tarkastella Jenkinsin Robot Framework Pluginin avulla.



Kuvio 9. Projektin ylätason rakenne

Robot-testisetit nuodattavat myös suoraviivaista logiikkaa. Testit alustetaan tarvittavalla informaatiolla, kuten kohteen IP-osoitteella ja käytettävän verkkorajapinnan nimellä. Tämän jälkeen ajetaan setin sisältämät testitapaukset. Jokainen testitapaus sisältää kaksi päävaihetta: testipakettien lähettäminen kohteelle ja tarkistus kohteen vikaantumisen varalta. Testi on onnistunut, jos kohde on testin jälkeen edelleen responsiivinen. Mikäli kohde ei vastaa tiedusteluihin, testi epäonnistuu. Kuvio 10 havainnollistaa testitiedostojen suorituslogiikkaa.



Kuvio 10. Testitiedoston suorituslogiikka

5.2 Toteutus

5.2.1 IGMP Robot -kirjasto

Projekti aloitettiin kirjoittamalla Robot kirjasto *IgmpLibrary.py*. Kirjasto sisältää funktiot IGMP-protokollan mukaisten pakettien rakentamista ja manipulointia sekä muita testissä tarvittavia toiminnallisuuksia varten. Kirjasto alustetaan kohteen IP-osoitteella ja verkkorajapinnan nimellä, sekä sisäisillä pakettilista- ja statusmuuttujilla.

Multicast-kontrolliliikenteeseen tarkoitetut paketit rakennetaan kirjaston käyttämällä sisäisellä funktiolla, joka muuntaa muuttujana annetun multicast-osoitteen sopivaksi MAC-osoitteeksi.

Pakettien lähettäminen hoidetaan myös sisäisellä funktiolla, jota kutsutaan jokaisen pakettien lähetykseen liittyvän metodin lopuksi. Funktio ottaa nimettyinä argumentteina vastaan kaiken pakettia koskevan olennaisen datan, sekä rakentaa ja lähettää paketin käyttäen Scapyn sendp-funktiota. Sendp-funktio on tarkoitettu sellaisten pakettien lähetykseen, joille Scapyn ei toivota laskevan tason 2 kenttiä automaattisesti, vaan ne määritellään erikseen. Lähetetyt paketit lisätään myös sisäiseen pakettilistaan.

Pakettien tyypin määritteleminen paketin tyypin nimellä (esim. ”Membership Query”) onnistuu sisäisen funktion avulla, joka vastaanottaa muuttujana tyypin nimen ja palauttaa standardin mukaisen tavun kokoisen paketin tyyppiä osoittavan arvon.

Kohteen responsiivisuuden toteamiseksi kirjasto sisältää kuuntelijan, joka nauhoittaa verkkorajapintaan saapuvia paketteja. Tämä kuuntelija käyttää Scapyn sniff-funktiota, joka mahdollistaa määriteltyyn rajapintaan saapuvien pakettien kaappauksen ja manipuloinnin. Koska testattava kohde oli verkossa IGMP tiedustelijan roolissa, rajapintaan kohteen IP-osoitteesta saapuva Membership Query-viestin katsottiin osoittavan kohteen olevan tavoitettavissa. Kuuntelija manipuloi suoraan kirjaston sisäistä statusmuuttujaa sen mukaan, oliko kohde tavoitettavissa vai ei. Status tarkistetaan **Status Should Be**-avainsanalla, joka odottaa sisäisen statusmuuttujan olevan muuttujana annettu odotettu status. **Status Should Be**-avainsanaa käytetään myös testien onnistumisen määrittelyyn. Jos status on jotain muuta kuin odotettu status, tapahtuu poikkeus.

Poikkeuksen tapahtuessa testin aikana lähetetyt paketit luetaan sisäiseltä pakettilistalta .pcap-tiedostoon, joka nimetään aikaleimalla kaksoiskappaleiden välttämiseksi. Näin poikkeuksen aiheuttamat paketit saadaan talteen mahdollista testin toistoa varten.

Robot testitiedostossa käytettäviä (ei-sisäisiä) avainsanoja kirjastosta löytyy seuraavat:

Send IGMPv2 Packet With Padding. Lähettää muuttujien avulla määritellyn IGMPv2-paketin. Paketti on myös mahdollista topata muuttujana annettavalla määrällä ”A” -kirjaimia. Näin voidaan luoda ylisuuria rikkinäisiä paketteja.

Send IGMPv2 Membership Query. Muuttujien avulla määriteltävä IGMP-paketti, joka on oletuksena standardi IGMPv2 Membership Query.

Enumerate IGMP Field. Lähettää IGMP paketteja jokaisella mahdollisella mrcode tai type-kentän arvolla.

Send IGMPv3 Membership Query. Lähettää muuttujien mukaan määritellyn IGMPv3 Membership Queryn. Lähdeosoitelistan voi määritellä pilkulla erotettuna tai CIDR -notaationa.

IGMP Fuzz. Lähettää muuttujalla määritellyn määrän paketteja, joiden IGMP-tason kentät on täytetty satunnaisilla arvoilla.

5.2.2 IGMP Robot -testitiedosto

Testitiedostossa *igmp.robot* (liite 1) luodaan testitapauksia käyttäen IGMP Robot-kirjastossa määriteltyjä avainsanoja. Testitiedostossa määritellään yksi sisäinen avainsana, **Check If The Link Is Still Up**. Kyseistä avainsanaa käytetään kohteen responsiivisuuden varmistamiseen ja testin läpäisyn määrittelemiseen. Sitä kutsutaan jokaisen testitapauksen lopuksi.

Tiedosto sisältää kuusi testitapausta, joissa lähetetään rikkinäisiä IGMP-viestejä tai useita tietyntyyppisiä viestejä useita kappaleita nopeaan tahtiin. IGMP Robot -kirjasto alustetaan uudestaan jokaisen testitapauksen yhteydessä.

5.2.3 PIM Scapy -kirjasto

Scapy tukee useita protokollia, mutta siitä puuttuvat kokonaan PIM-SM protokollan mukaiset paketit. Jotta IGMP Robot -kirjaston tyylisen PIM -kirjaston luominen oli mahdollista, tuli ensin toteuttaa *pim.py* Scapy -kirjasto. *Pim.py* ei kuitenkaan toteuta PIM-SM-protokollaa kokonaisuudessaan.

Uuden protokollan tai Scapyn termeissä tason määritteleminen onnistuu käyttämällä Scapyn Packet-luokan alaluokkia. Jokainen taso on lista ennalta määriteltyjä kenttiä,

joilla on nimi ja oletusarvo. Kentän arvo voi olla myös lista muita kenttiä. Tasot voidaan sitoa toisiinsa jonkin tietyn tason sisältämän kentän arvon perusteella. Esimerkiksi ICMP-taso on sidottu IP-tasoon IP-paketin protokollakentän arvon 1 perusteella. Kun tasot on näin nivottu yhteen, on lopputuloksena objekti, joka voidaan Scapyn avulla tulkita joko raa'aksi tavujonoksi verkkoon lähetettäväksi tai printattavaksi ihmisen luettavaan muotoon. Kuviossa 11 on Scapylla luotu ICMP-paketti tulostettuna näissä kahdessa eri muodossa.

```
>>> packet = IP()/ICMP()
>>> packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0

>>> raw(packet)
b'E\x00\x00\x1c\x00\x01\x00\x00@\x01|\xde\x7f\x00\x00\x01\x7f\x0
0\x00\x01\x08\x00\xf7\xff\x00\x00\x00\x00'
>>>
```

Kuvio 11. Paketin rakenne Scapyssa

PIM kirjasto sisältää Scapyn ennalta määriteltyjä kenttiä käyttämällä luodun PIMv2-tason, jonka rakenne vastaa kaikille PIM-viesteille yhteisen otsakkeen rakennetta (Kuvio 12). PIMv2-taso on sidottu ylempään IP-tasoon PIM-protkollanumeron 103 perusteella. Kirjastoon on myös toteutettu PIM-protokollan mukaiset Hello, Join/Prune, Assert, Bootstrap ja Candidate RP Advertisement-tasot. Nämä tasot on puolestaan sidottu PIMv2 -otsakkeeseen sen Type-kentän arvon perusteella. Lisäksi

kirjastosta löytyvät PIM-protokollan mukaisesti enkoodatut lähde-, multicast- ja unicast-osoitteet.

tavu	1		2	3	4
paketti	PIM Ver	Type	Reserved	Checksum	

Kuvio 12. PIM viestin otsake

5.2.4 PIM Robot -kirjasto

PIM protokollan mukaiset ja sitä rikkovat paketit rakennetaan *PimLibrary.py* Robot kirjaston avulla. Kirjasto alustetaan IGMP Robot kirjaston tapaan kohteen IP-osoitteella ja verkkorajapinnan nimellä. Myös status- ja pakettilistamuuttujaa käytetään samalla tavalla, samoin kuin kohteen responsiivisuutta luotavaa kuuntelijaa. Kirjastosta löytyy kuitenkin myös muutama uusi sisäinen metodi.

Pakettien lähetyksestä huolehtii kaksi eri sisäistä metodia, yksi multicast ja toinen unicast-osoitteisiin lähteviä paketteja varten. Tämä on tarpeellista, koska Candidate RP Advertisement viestit lähetetään suoraan BSR:n IP-osoitteeseen unicast-lähetyksenä.

Satunnaisilla arvoilla täytettyjen enkoodattujen lähde-, multicast- ja unicast-osoitteiden sekä osoitelistojen generointiin löytyy myös sisäiset funktiot. Niitä käytetään apuna eri PIM-viestien fuzzaamisessa.

Testitiedostossa käytettäviä avainsanoja on kirjastossa yhdeksän:

Send PIMv2 Hello. Rakentaa ja lähettää PIMv2 Hello-viestejä. Funktiolle voidaan antaa muuttujina eri PIMv2 Hell -viestin vaihtoehtoisia arvoja.

Send PIMv2 With Padding. Rakentaa ja lähettää PIMv2-otsakkeen topattuna muuttujalla asetettavalla määrällä ylimääräisiä tavuja. Ylimääräisten tavujen määrä voidaan myös generoida satunnaisesti. Tavut voivat myös olla joko satunnaisia tai pelkkiä ”A” -kirjaimia.

Send PIMv2 Assert. Rakentaa ja lähettää Assert-viestin.

Send PIMv2 Candidate RP Advertisement. Käytetään Candidate RP Advertisement-viestien rakentamiseen ja lähettämiseen. Niiden multicast-ryhmien listan, joille

viestissä määritelty RP on kandidaattina, voidaan määritellä joko pilkulla erotetulla listana tai CIDR-notaationa.

Send PIMv2 Join. Käytetään Join/Prune-viestien lähettämiseen. Viesti sisältää sekä lisättyjen että karsittujen osoitteiden listat. Käyttäjä määrittelee listojen sisällön käsin käyttäen seuraavaa formaattia:

{Multicast group address}:joined:{Source Address}:srw{SRW}:pruned:{Source Address}:srw:;{another similar sequence}

Ylläolevassa formaatissa Multicast group address on se multicast osoite, jota Join/Prune koskee. Sitä seuraa lista lisättyjä ja karsittuja osoitteita. Srw-lohko määrittelee enkoodatun lähdeosoitteen S, R ja W-bitit.

Send PIMv2 Bootstrap. Rakentaa ja lähettää Bootsrap-viestin. Käyttäjä määrittelee viestin sisältämän RP joukon käsin käyttäen seuraavaa formaattia:

{Group address}:rpaddrs:{RP address}:holdtime:{Holdtime value}:priority:{Priority value},{another RP address sequence};{another sequence}

Ylläolevassa formaatissa Group Address on sen multicast-ryhmän osoite, jota lista koskee. Sitä seuraa lista RP-osoitteita, jotka ovat valideja kyseiselle multicast-ryhmälle.

Fuzz Assert. Käytetään Assert-viestien fuzzaamiseen. Kaikki Assert-viestin kentät on määriteltävissä. Ne kentät, joiden arvoa käyttäjä ei määrittele, täytetään satunnaisilla arvoilla.

Fuzz Join. Käytetään Join/Prune-viestien fuzzaamiseen. Kaikki Join/Prune-viestin kentät on määriteltävissä, mukaanlukien listat karsituista ja lisätyistä osoitteista. Ne kentät, joiden arvot käyttäjä jättää määrittelemättä, täytetään satunnaisilla arvoilla.

Fuzz Bootstrap. Käytetään Bootsrap-viestin fuzzaamiseen. Toimii samalla periaatteella kuin muut kirjastossa määritellyt fuzzerit.

5.2.5 PIM Robot -testitiedosto

IGMP Robot testitiedoston tapaan *pim.robot* (liite 2) testitiedosto käyttää tiedoston sisällä määriteltyä avainsanaa **Check If Link Is Still Up** määritelläkseen kohteen tilan

testitapausten jälkeen. Kohteen IP-osoite ja käytettävä rajapinta määritellään testitiedoston sisäisiksi muuttujiksi. PIM Robot-testitiedosto sisältää viisi rikkovaa testitapausta, joissa käytetään hyväksi Pim Robot -kirjaston avainsanoja. Kolmessa testitapauksessa ajetaan eri fuzzereita asettaen joitain paketin kenttien arvoja staattisiksi. Esimerkiksi testitapauksessa **Bootstrap Fuzz** asetetaan rpcount-kenttään staattinen arvo 10, vaikka rp-kenttien määrä on sattumanvarainen. Tällainen ristiriitta voisi aiheuttaa kohdeohjelmistossa käsittelemättömän virheen.

5.2.6 Integrointi Jenkinsiin

Testien parametrisointia ja ajamista Jenkins-automatisointiserveriltä varten kirjoitettiin kaksi shell skriptiä, yksi kumpaakin testitiedostoa varten. Skripteissä määritellään raporttien muodostamiseen käytettävä kansio ja asetetaan raporttien nimeämistekniikaksi aikaleimaus. Testattavan kohteen IP-osoite ja käytettävän verkkorajapinnan nimi annetaan skripteille komentoriviparametreinä.

Jenkins serverille luotiin uusi tehtävä, joka otti etäyhteyden testit suorittavaan laitteeseen ja ajoi shell skriptit Jenkinsissä määritellyillä parametreillä. Robot Framework Plugin asetettiin hakemaan testitulokset skripteissä määritellystä kansiosta. Lisäksi Plugin konfiguroitiin säilyttämään kaikki testin aikana muodostuneet .pcap-tiedostot. Lopuksi Jenkins-tehtävä määriteltiin ajamaan itsensä kerran joka yö.

5.3 Ratkaisun toiminta

Kun ratkaisu oli implementoitu CI-ympäristöön, sen toimintaa testattiin muutamia kertoja ajamalla Jenkins -tehtävä manuaalisesti ja seuraamalla verkossa liikkuvia paketteja Wireshark -verkkoanalyysityökalulla. Testit toimivat odotetulla tavalla ja pienen hienosäädön jälkeen myös Jenkins raportointi ja arkistointi saatiin toimimaan. Kun testit olivat olleet jo jotain viikkoja päivittäisessä ajossa, huomattiin joidenkin testien epäonnistuneen. Tämä kuitenkin johtui testausympäristössä tapahtuneista muutoksista eivätkä olleet rikkovien testien aikaansaannosta.

6 Tulokset ja pohdinta

Ratkaisu täytti toimeksiannossa asetetu vaatimukset. Haluttujen testien suorittaminen ja tulosten raportoiminen automaattisesti implementoitiin onnistuneesti osaksi CI-kehitysympäristöä.

Ratkaisu jätti kuitenkin toivomisen varaa varsinkin avainsanojen yleiskäyttöisyyden ja johdonmukaisuuden osalta. Tämä käy erityisesti selväksi IGMP Robot -kirjaston avainsanoja ja rakennetta tutkailtaessa. Esimerkiksi erillisten avainsanojen olemassaoloa topattujen IGMP-pakettien ja IGMPv2 Membershio Query:jen tekemiseen voidaan pitää kyseenalaisena. Ratkaisu ei myöskään sinällään ole yleiskäyttöinen työkalu minkä tahansa verkkoprotokollan implementaation testaukseen, vaan vaatii aina uuden protokollakohtaisen Robot -kirjaston kehittämisen.

Ratkaisun avulla ei huomattu vakavia ongelmia kohdetta testatessa. Muutama erikoisuus havaittiin. Testikohdetta ei saatu kaatumaan testeissä generoiduilla paketeilla. Ratkaisun tulevaisuus on vielä epävarma. Siinä kehitettyjä menetelmiä käyttäen luodaan kuitenkin todennäköisesti testisettejä vielä ainakin joidenkin verkkoprotokollien ohjelmallisille toteutuksille. Jatkokehityksessä tullaan erottelemaan selkeämmin fuzz-tyyliset ja toiminnallisten ominaisuudet. PIM Robot -kirjastossa onkin jo pyritty pitämään nämä osat erillään.

Henkilökohtaisesti tunsin projektin olevan varsin antoisa. Vaikka verkkoprotokollien standardit ovat välillä puista purtavaa, protokollien mukaisten pakettien implementonti ja toimimaan saattaminen oli tyydyttävää. Ratkaisun implementointi kehitysympäristöön antoi minulle myös uusia näkökulmia ohjelmistokehityksen ja -testauksen maailmaan. Tunnen oppineeni runsaasti multicast-teknologioista sekä käytetyistä ohjelmistokehyksistä ja Python-ohjelmointikielestä itsestään.

Lähteet

Dutta, R. 2015. A Framework for Software Security Testing and Evaluation. Pro gradu: Uumajan yliopisto http://liu.diva-portal.org/smash/get/diva2:858033/FULLTEXT01.pdf

Foundation Level Syllabus. 2018. International Software Testing Qualifications Board (ISTQB) -yhteisön julkaisemaa materiaalia ISTQB sertifikaattia tavoitteleville testaajille. Viitattu 18.11.2018. https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html

Heller, Martin. 2017. What is Jenkins? The CI server explained. Artikkeli Jenkins automatisointiserveristä infoworld.com -sivustolla. Viitattu 28.10.2018. https://www.infoworld.com/article/3239666/devops/what-is-jenkins-the-ci-server-explained.html

Insta DefSec – Keitä me olemme?. 2018. Insta DefSec OY:tä esittelevä lyhyt artikkeli Insta DefSec:in sivuilla. Viitattu 9.12.2018. https://www.instadefsec.fi/insta-defsec/tietoa-meista.html

Insta on turvallisen ja kilpailukykyisen yhteiskunnan rakentaja ja kehittäjä. 2018. Insta konsernia esittelevä artikkeli Instan sivuilla. Viitattu 9.12.2018. https://www.insta.fi/insta-group/tietoa-konsernista.html

IP Multicast Technology Overview. 2009. Ciscon julkaisema dokumentti Multicast -teknologian toiminnasta. Viitattu 4.11.2018. https://www.cisco.com/en/US/docs/ios/ipmulti/configuration/guide/imc_tech_oview.pdf

ISO 29119-1:2013. Software and systems engineering – Software testing – Part 1: Concepts and definitions. Viitattu 20.11.2018. https://ieeexplore.ieee.org/document/6588537

Khan, E. & Khan, F. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techinuqes. International Journal of Advanced Computer Science and Applications, 3, 6, 12-15. Viitattu 28.10.2018. http://thesai.org/Publications/ViewIssue?volume=3&issue=6&code=IJACSA

McGraw, G. & Potter, B. 2004. Software Security Testing. IEE Security & Privacy, 2, 5, 81-85. Viitattu 13.11.2018. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1341418

Noppari, T. 2008. IP-RYHMÄLÄHETYS. Opinnäytetyö, AMK. Satakunnan ammattikorkeakoulu, Tietotekniikan koulutusohjelma, Department of Information Technology. Viitattu 4.11.2018. http://www.theseus.fi/bitstream/handle/10024/746/Noppari%20Teemu.pdf

Osterman, L. 2007. Threat Modeling Again, STRIDE. Artikkeli Microsoftin blogissa, jossa kuvaillaan STRIDE uhkamallinnuskategorioita. Viitattu 13.11.2018. https://blogs.msdn.microsoft.com/larryosterman/2007/09/04/threat-modeling-again-stride/

OWASP Testing Guide. 2015. Open Web Application Security Project -organisaation julkaisema dokumentti tietoturvatestauksesta, versio 4. Viitattu 12.11.2018. https://www.owasp.org/images/1/19/OTGv4.pdf

RFC 1112:1989. Host Extensions for IP Multicasting. Viitattu 24.10.2018. https://tools.ietf.org/html/rfc1112

RFC 2236:1997. Internet Group Management Protocol, Version 2. Viitattu 25.10.2018. https://tools.ietf.org/html/rfc2236

RFC 3376:2002. Internet Group Management Protocol, Version 3. Viitattu 26.10.2018. https://tools.ietf.org/html/rfc3376

RFC 5771:2010. IANA Guidelines for IPv4 Multicast Address Assignments. Viitattu 24.10.2018. https://tools.ietf.org/html/rfc5771

Robot Framework/. 2018. Johdanto Robot Framework -ohjelmistokehykseen projektin kotisivuilla. Viitattu 18.10.2018. http://robotframework.org/#documentation

Takanen A., DeMott, J. & Miller, C. 2008. Fuzzing for Software Security Testing and Quality Assurance. Artech House.

Welcome to Scapy's documentation. 2018. Scapyn dokumentaatio, versio 2.4.0. Viitattu 18.10.2018. https://scapy.readthedocs.io/en/latest/introduction.html

Liitteet

Liite 1. igmp.robot

```
*** Settings ***

Library         ./libraries/IgmpLibrary.py  ${host}  ${interface}
Test Teardown   Reset Link Status

*** Keywords ***
Check If The Link Is Still Up
   Log to Console          \nChecking if the link is still up..
   Check Link Status
   Status Should Be        OK

*** Variables ***
${host}        192.168.1.0
${interface}    eth0

*** Test Cases ***
Send IGMPv2 Membership Flood
   Send Igmpv2 Packet With Padding                  Membership Report v2  224.0.0.1
0  count=1  src=192.168.1.0/24
   Check If The Link Is Still Up

Send Long IGMPv2 Packets
   Send Igmpv2 Packet With Padding                  Membership Query  224.0.0.1  100
count=50
   Send Igmpv2 Packet With Padding                  Membership Query  224.0.0.2  100
count=50
   Send Igmpv2 Packet With Padding                  Membership Query  224.0.0.13
100  count=50
   Check If The Link Is Still Up

Enumerate IGMPv2 Mrcode And Type Fields
   Enumerate IGMP Field                         mrcode  224.0.0.1  dst=224.0.0.1
   Check If The Link Is Still Up
   Enumerate IGMP Field                         type    224.0.0.1  dst=224.0.0.1
   Check If The Link Is Still Up

Send Fuzzed IGMP Packets
   IGMP Fuzz                            300
   Check If The Link Is Still Up
```

Try The CVE 2012-0207 Exploit

Send Igmpv2 Membership Query 224.0.0.1 mrcode=0xff

Send Igmpv2 Membership Query 0.0.0.0 mrcode=0x00 padding=\x00\x00\x00\x00\x00\x00\x00\x00

Check If The Link Is Still Up

Try IGMPv3 DOS Via Group-And-Source Specific Queries

Send IGMPv3 Membership Query 225.0.2.1 0xff 192.168.1.0/24 dst=225.0.2.1 src=192.168.1.0

Send IGMPv3 Membership Query 225.0.2.1 0xff 192.168.2.0/24 dst=225.0.2.1 src=192.168.1.0

Send IGMPv3 Membership Query 225.0.2.1 0xff 192.168.3.0/24 dst=225.0.2.1 src=192.168.1.0

Send IGMPv3 Membership Query 225.0.2.1 0xff 192.168.4.0/24 dst=225.0.2.1 src=192.168.1.0

Check If The Link Is Still Up

Liite 2. pim.robot

```
*** Settings ***
Library        ./libraries/PimLibrary.py  ${host}  ${interface}
Test Teardown   Reset Link Status

*** Keywords ***
Check If The Link Is Still Up
   Log to Console          \nChecking if the link is still up..
   Check Link Status
   Status Should Be        OK

*** Variables ***
${host}        192.168.1.0
${interface}    eth1

*** Test Cases ***

Send PIMv2Hello
   Send PIMv2 Hello          holdtime=100   drpriority=22  genid=4322  count=1
src=192.168.1.0/24
   Send PIMv2 Hello          holdtime=65535   drpriority=1000  genid=0  count=1
src=192.168.1.0/24
   Check If The Link Is Still Up

Send Malformed PIMv2 Packets
   Send PIMv2 With Padding      length=500   count=4  packet_type=2  src=10.3.3.99
dst=224.0.0.13
   Send PIMv2 With Padding      length=200   count=100  src=192.168.1.99
dst=224.0.0.13  packet_type=3
   Send PIMv2 With Padding      count=500    packet_type=1
randomize_length=True  randomize_padding=True
   # Send PIMv2 Candidate RP Advertisement    count=1   prefixcount=1   priority=100
rpaddr=123.123.123.123   groupaddrs=224.0.1.2.3   holdtime=200
   Check If The Link Is Still Up

Assert Fuzz
   Send PIMv2 Hello           holdtime=100   drpriority=22  genid=4322  count=1
src=192.168.1.20
   Fuzz Assert               dst=224.0.0.13  src=192.168.1.20  count=100
malformed_addrs=True
   Fuzz Assert               dst=224.0.0.13  src=192.168.1.20  count=100
   Check If The Link Is Still Up
```

Join Fuzz

Fuzz Join dst=224.0.0.13 src=192.168.1.20 count=100

Fuzz Join dst=224.0.0.13 src=192.168.1.20 count=100 malformed_addrs=True

Fuzz Join dst=224.0.0.13 src=192.168.1.20 count=100 numjoined=0 numpruned=0

Fuzz Join dst=224.0.0.13 src=192.168.1.20 count=100 numjoined=200 numpruned=200

Fuzz Join dst=224.0.0.13 src=192.168.1.20

Check If The Link Is Still Up

Bootstrap Fuzz

Fuzz BootStrap dst=224.0.0.13 src=192.168.1.20 count=10 rpcount=0 fragrpcount=0

Fuzz BootStrap dst=224.0.0.13 src=192.168.1.20 rpcount=10 fragrpcount=10

Fuzz BootStrap dst=224.0.0.13 src=192.168.1.20 grgroups=225.0.2.1:rpaddrs:192.168.1.20:holdtime:200:priority:1

Fuzz BootStrap dst=224.0.0.13 src=192.168.1.20 count=10 randomize_counters=True

Check If The Link Is Still Up