Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutus

2021

Viljami Ruokonen

VERKKOKARTOITUKSEN VISUALISOINTI SIGMA.JS -KIRJASTOLLA

– case yritys X

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutus

2021 | 28

Viljami Ruokonen

VERKKOKARTOITUKSEN VISUALISOINTI SIGMA.JS -KIRJASTOLLA

- case yritys X

Tämän opinnäytetyön tavoitteena oli luoda tietoturvapalveluita tarjoavalle yritykselle X heidän jo olemassa olevaan verkkoportaaliin käyttöliittymä, joka visualisoi verkonkartoitustulosten perusteella kuvan asiakkaan verkosta. Työkalu tulee yritys X:n tietoturva-asiantuntijoiden käyttöön helpottamaan heidän työtään asiakkaiden verkkojen tarkkailussa. Toisaalta työkalu hyödyttää myös yritys X:n asiakkaita, joilla ei välttämättä ole teknistä osaamista tulkita, mitä verkonkartoitustulosten raaka-data itsessään esittää.

Perehdyin jo olemassa oleviin verkonkartoitustyökaluihin ja niiden käyttöön. Tutkin miten kyseiset työkalut esittävät kartoitustulokset ja kuinka helppoa niillä on seurata verkon muutoksia.

Työn tuloksena syntyi toimiva visualisointityökalu, joka näyttää verkonkartoitustuloksen verkkotopologiana ja siitä on myös mahdollista nähdä avoimet portit ja porteissa olevat palvelut. Sillä on myös mahdollista vertailla graafisesti eroja kahden eri verkonkartoitustuloksen välillä ja tarkastella verkonkartoituksia aikajanassa.

ASIASANAT:

verkkokartoitus, visualisointi, Sigma.js, Vue.js, Nmap, Zenmap, OpenVas

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2021 | 28

Viljami Ruokonen

VISUALIZING A NETWORK MAPPER RESULT WITH THE SIGMA.JS -LIBRARY

- case Company X

This thesis aimed to create an interface that visualizes an image of the customer's network based on the network mapping results for cyber security Company X. The tool will be used by Company X's security experts to facilitate their work in monitoring customer networks. The tool also benefits customers of Company X who may not have the technical expertise to interpret what raw data from network mapping results presents.

To achieve the objective of this thesis, the author studied existing network mapping tools and their use. The author also became familiar with how these tools present mapping results and how you can track network changes.

The result of the thesis was a working visualization tool. The tool displays the network mapping result as a network topology. From the network topology, it is possible to see open ports and services in the ports. This tool also makes it possible to graphically compare the differences between two network mapping results and view network mappings in a timeline.

KEYWORDS:

network mapping, visualization, Sigma.js, Vue.js, Nmap, Zenmap, OpenVas

SISÄLTÖ

KUVAT

KOODIT

1 JOHDANTO

Verkkoteknologioiden ja ohjelmistojen kehittyessä internet ja tietoverkot ovat yhä tärkeämmässä roolissa jokapäiväisessä elämässä. Tänäpäivänä suurin osa yritysten ja virastojen palveluista ovat internetissä. Yritykset ja virastot kohtaavat jokapäiväisessä toiminnassaan väistämättä erilaisia tietoturvauhkia. Ihmisillä, jotka ylläpitävät yritysten ja virastojen verkkoja, on yhä vaikeampi suorittaa seurantatehtäviä tietoverkkojen kasvaessa. Järjestelmävalvojien työmäärän helpottamiseksi selkeä ja tehokas tietoverkon visualisointi on hyödyllinen tietoverkkojen ylläpidossa ja valvonnassa. (Kan, ym. 2010) Järjestelmävalvojien on tärkeää olla perillä tietoverkon topologiasta, jotta verkon suojausta ja valvontaa voidaan toteuttaa tehokkaammin. Esimerkiksi havaittaessa verkkohyökkäys tai saastunut isäntäkone järjestelmänvalvojan on kyettävä paikantamaan kone topologisesti tarkan analyysin tai isäntäkoneen puhdistuksen tai eristämisen suorittamiseksi. (Kienzle;Evans ja Elder 2013)

Opinnäytetyön toimeksiantona oli luoda tietoturvapalveluita tarjoavalle yritykselle X heidän jo olemassa olevaan verkkoportaaliin käyttöliittymä, joka visualisoi verkonkartoitustulosten perusteella kuvan asiakkaan verkosta. Käyttölittymällä pitäisi olla mahdollista seurata verkon nykytilannetta ja tarkastella, miten verkko on muuttunut ajansaatossa. On olemassa ohjelmistoja, jotka mahdollistavat verkkojen skannauksen ja tuloksen visualisoinnin, mutta kyseiset ohjelmat tarvitsevat paljon teknistä osaamista ja niillä ei ole mahdollista tarkastella verkkojen kehitystä aikajanamuodossa. Työkalu tulee yritys X:n tietoturva-asiantuntijoiden käyttöön helpottamaan heidän työtään asiakkaiden verkkojen kartoittamisessa ja tarkkailussa, mutta myös yritys X:n asiakkaille, joilla ei välttämättä ole teknistä osaamista tulkita, mitä verkonkartoitustulosten raakadata itsessään esittää.

Nykyajan web-teknologioita hyödyntäen on mahdollista rakentaa käyttäjäystävällisiä ja interaktiivisia käyttöliittymiä, jotka yhdistettynä datan visualisointiin luovat mahdollisuuden toteuttaa todella näyttäviä ja informatiivisia visualisointeja. Tulen käyttämään työssäni HTML5-, JavaScript-, Vue.js- ja Sigma.js-teknologioita. Luvussa 2 vertailen erilaisia verkonkartoitusohjelmia ja miten nämä ohjelmat esittävät verkonkartoituksen tulokset. Luvussa 3 kertaan toteutuksen vaihe vaiheelta ja luvussa 4 pohdin, miten onnistuin toteutuksessa ja miten visualisointityökalua voisi vielä mahdollisesti jatkojalostaa.

2 VERKONKARTOITUSOHJELMAT

Yritysten palveluiden siirtyessä internetiin tietoverkot kasvavat kovaa vauhtia (Kan, ym. 2010). Tietoverkot koostuvat toisiinsa kytketyistä tietokoneista, joita kutsutaan solmuiksi. Tietoverkkojen ja käyttäjämäärien kasvaessa tietoverkon ylläpitäjillä tulee yhä hankalammaksi valvoa verkon turvallisuutta. Tietoverkkojen ylläpitäjien työn helpottamiseksi on olemassa työkaluja, joilla on mahdollista tunnistaa laitteet ja laitteissa toimivat protokollat, portit ja käyttöjärjestelmät. Näiden tietojen määrittämistä kutsutaan sormenjälkien ottamiseksi. Laitteiden sormenjälkien ottamisella on mahdollista havaita luvattomia tai vaarallisia laitteita, tai mahdollisia haavoittuvuuksia laitteissa ja niiden ohjelmistoissa. (Ghanem ja Belaton 2013)

Tässä luvussa tutustun erillaisiin verkonkartoitusohjelmiin ja vertailen niiden tapaa esittää kartoituksien tuloksia.

2.1 Nmap-ohjelma

Nmap on lyhenne sanoista ”network mapper” eli verkkokartoitin. Se on ilmainen ja avoimen lähdekoodin komentoriviohjelma, jota käytetään laitteiden ja palveluiden löytämiseen tietoverkoista lähettämällä paketteja ja analysoimalla niiden vastauksia. (Nmap 2019)

Laitteiden etsiminen tietoverkosta Nmap:lla onnistuu komennolla *nmap -sn* <verkon ip osoite>. Tätä kutsutaan ”ping skanniksi”, joka ei suorita porttien kartoitusta. Tulos tulostuu komentoriville, kuten kuvan 1 esimerkissä.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sn 192.168.56.0/24
Starting Nmap 7.91 ( https://nmap.org ) at 2021-05-09 10:27 EDT
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled. Try using --system-dns or specify vali
d servers with --dns-servers
Nmap scan report for 192.168.56.1
Host is up (0.0036s latency).
Nmap scan report for 192.168.56.105
Host is up (0.00030s latency).
Nmap scan report for 192.168.56.106
Host is up (0.0059s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.43 seconds
```

Kuva 1. Laitteiden kartoitustulos Nmap-ohjelmalla.

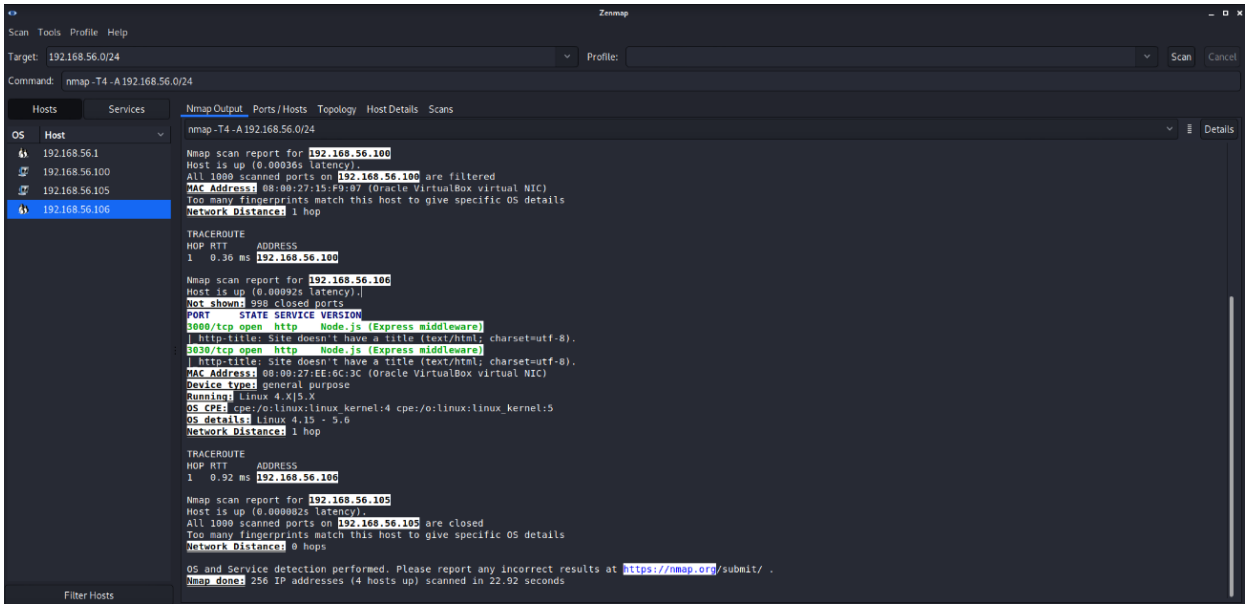
Esimerkin Nmap-kartoituksen tuloksena löytyi kolme laitetta. Yritysten tietoverkoissa laitemäärät voivat olla paljonkin suurempia. Tulosten luku pelkästään tekstitulosteena

alkaa olla erittäin hankalaa, jos laitteita on verkossa kymmenkunta ja haluttaisiin vielä mahdollisesti listata avoimet portit ja käyttöjärjestelmät.

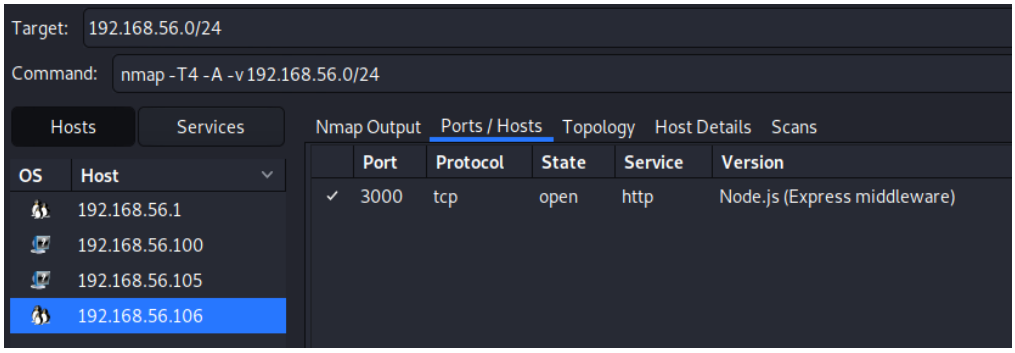
2.2 Zenmap-ohjelma

Zenmap on graafinen käyttöliittymä Nmap:lle. Se on tarkoitettu helpottamaan Nmap:n käyttöä. Usein käytetyt skannaukset on mahdollista tallentaa Zenmap:ssa profiileina, jotta niitä olisi helppo suorittaa toistuvasti. Zenmap myös mahdollistaa skannaustulosten keskinäisen vertailun ja skannaustuloksen tarkastelun graafisena verkkotopologiana. (Nmap 2019) Verkkotopologia on visualisointi tietoverkon rakenteesta. Se kuvaa tapaa, jolla tietoverkon laitteet ovat yhteydessä toisiinsa. (Oulun seudun ammattiopisto 2016)

Vaikka Zenmap on graafinen käyttöliittymä, skannauskomennot annetaan sille samassa muodossa, kuin ne kirjoitetaan Nmap:ssa komentoriville. Eli mitään suurta helpotusta tämä ei tarjoa verrattuna Nmap:n käyttöön. Kuvassa 2 on esimerkki skannauksen teosta ja sen tuloksesta. Tulosta on mahdollista tarkastella tekstimuodossa tai valitsemalla yksittäisiä laitteita vasemmasta sivupalkista. Tämä helpottaa yksittäisen laitteen avoimien porttien ja palveluiden tarkastelun. Kuvassa 3 on esimerkki yksittäisen laitteen avoimien porttien tarkastelusta.

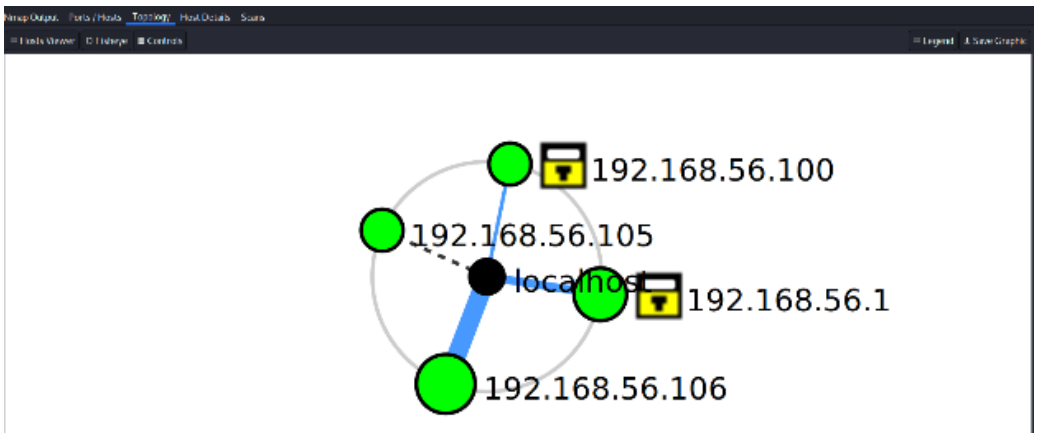


Kuva 2. Zenmap-skannauksen suoritus.



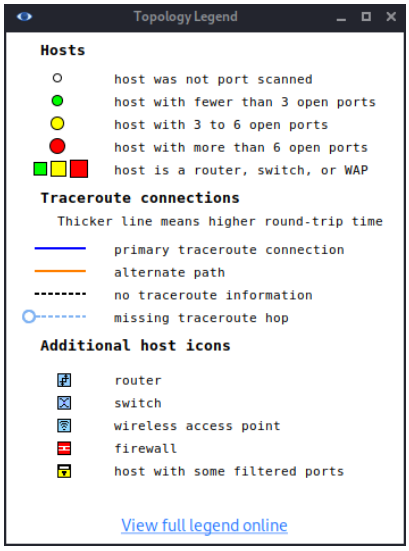
Kuva 3. Zenmap:lla yksittäisen laitteen tarkastelu.

Zenmap mahdollistaa myös tuloksen tarkastelun verkkotopologisena visualisointina, kuten kuvassa 4. Verkkotopologia näyttää kaikki laitteet, jotka skannaus on löytänyt ja sen, miten laitteet on liitetty toisiinsa verkossa.



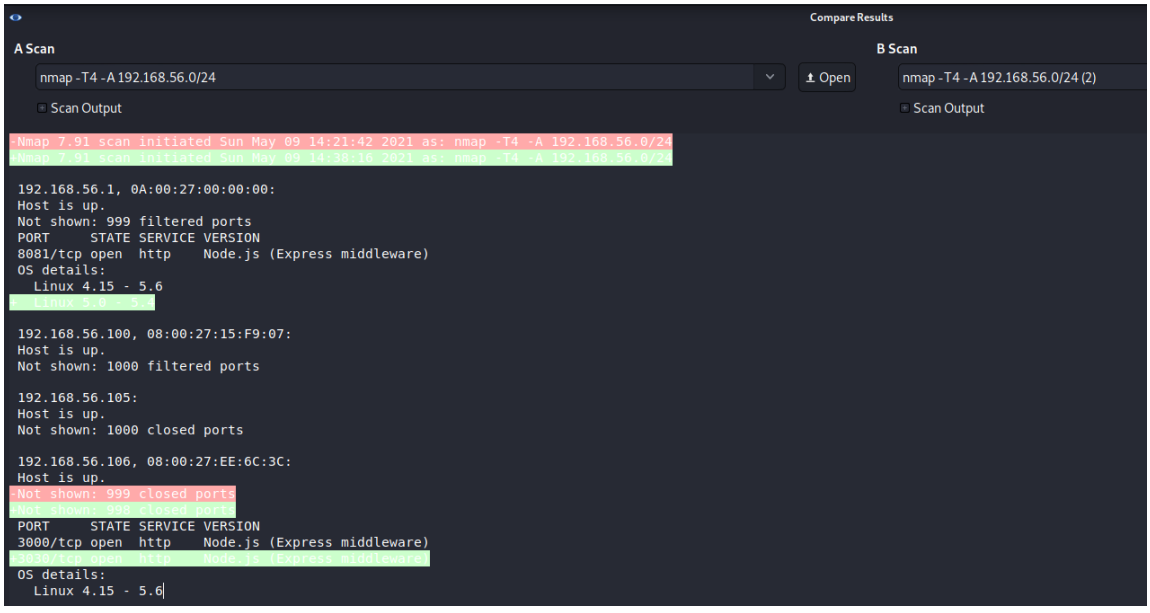
Kuva 4. Zenmap-skannaustulos verkkotopologiavisualisointina.

Verkkotopologiasta ei varsinaisesti ole mahdollista nähdä laitteiden avoimia portteja, mutta laitteet on värikoodattu sen mukaan, montako porttia laitteessa on auki. Esimerkiksi vihreä tarkoittaa, että portteja on auki vähemmän kuin kolme. Kuvassa 5 on selitteet verkkotopologian värikoodeille ja kuville.



Kuva 5. Zenmap-verkkotopologian kuvien selitteet.

Zenmap:ssa on mahdollista vertailla kahta eri kartoitustulosta, kuten kuvassa 6. Vertailu tehdään kuitenkin vain kahden tekstitiedoston välisistä eroista, kuten ylärivin kellonajan kohdalla on nähtävissä.



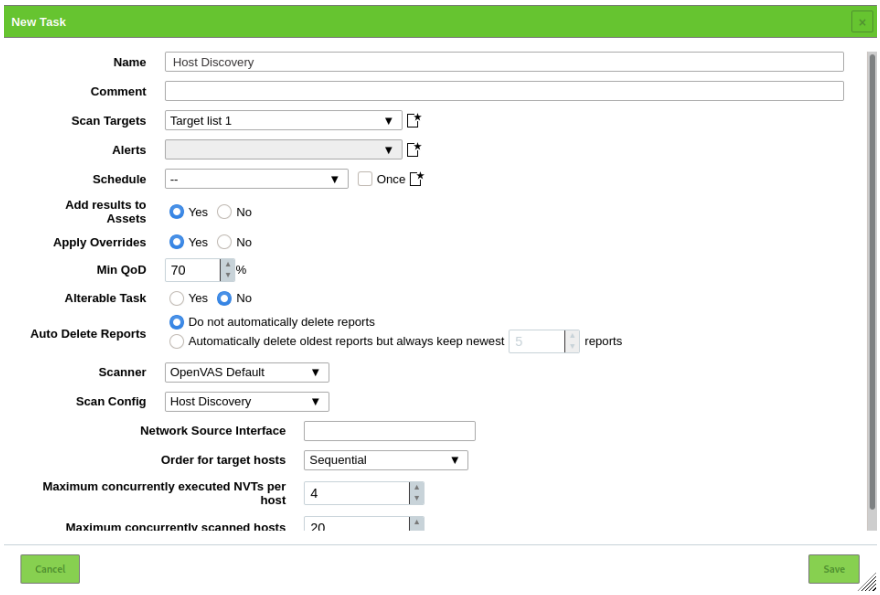
Kuva 6. Zenmap-skannausten vertailu.

Jos verkkotopologian visualisoinnissa olisi mahdollista nähdä avoimet portit ja palvelut ja skannausten eroavaisuuksia olisi mahdollista tarkastella kyseisessä visualisoinnissa, helpottaisi se suuresti verkkotopologian muutosten seuraamista.

2.3 OpenVas-ohjelma

OpenVas on avoimen lähdekoodin haavoittuvuuksien etsintään tarkoitettu työkalu, mutta sillä on myös mahdollista kartoittaa verkkoja. Skannerin mukana tulee haavoittuvuus-syötteet, joita päivitetään päivittäin. (Greenbone Networks 2021)

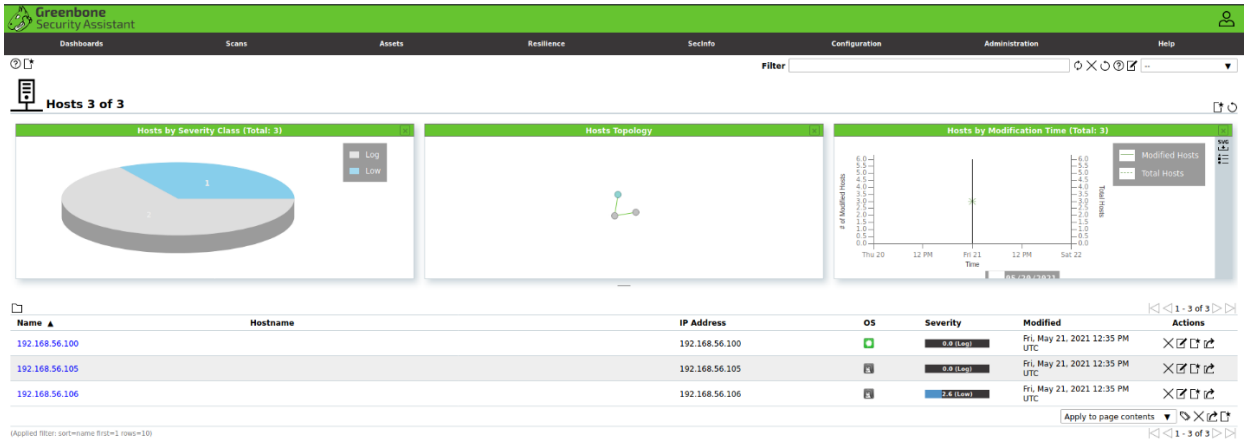
Kuvassa 7 on esimerkki verkkokartoituksen teosta OpenVas-ohjelmalla. ”*Scan Targets”*-alasvetovalikon vierestä kuvaketta painamalla saa syötettyä verkon osoitteen. ”*Scan Config”*-alasvetovalikosta valitaan *Host Discovery*, jonka jälkeen painetaan *save*-nappia. Tämän jälkeen tehtävä (engl. *task)* ilmestyy tehtävälistaan, josta se voidaan käynnistää *play*-kuvaketta painamalla.



Kuva 7. OpenVas-verkkokartoituksen luonti.

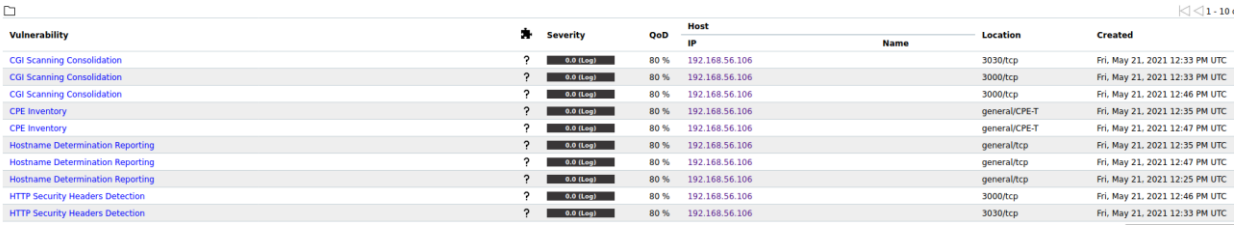
Verkkokartoituksen jälkeen, verkossa olevat laitteet voi nähdä *assets*-valikon alta. Kuvassa 8 näkyy listaus löydetyistä laitteista. Löydetyistä laitteista on nyt mahdollista tehdä *target*-lista, jota vasten voidaan suorittaa perusteellisempi kartoitus. Kuvan 8 listaus on perusteellisemman kartoituksen jälkeen. Siinä on mahdollista nähdä laitteella oleva käyttöjärjestelmä ja mahdolliset löydetyt haavoittuvuudet. Se näyttää myös yksinkertaisen

verkkotopologian löydetyistä laitteista. Klikkaamalla laitteen ip-osoitetta on mahdollista tarkastella laitetta tarkemmin.



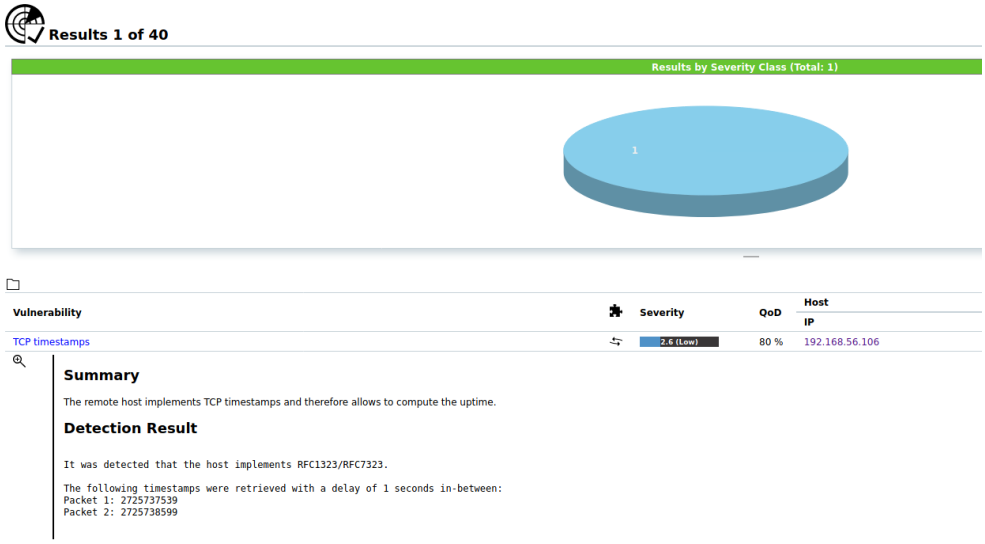
Kuva 8. OpenVas-skannaustuloksen laitteet listattuna.

Kuvassa 9 näkyy listattuna laitekohtaiset tulokset. Kuvan laitteella *192.168.56.106* oli kaksi http-palvelinta. Toinen oli portissa numero 3000 ja toinen portissa numero 3030. Suoritin kaksi erillistä kartoitusta. Ennen toista kartoitusta suljin toisen http-palvelimista, jotta näkisin, miten muutos näkyisi tuloksissa. Päätason laitelistauksessa muutos ei tullut ilmi. Avattaessa laitteen laitekohtaiset tulokset porttien muutokset on mahdollista nähdä ainoastaan *created*-kellonajoista mihin kellonaikaan portti on löydetty.



Kuva 9. OpenVas-ohjelman laitekohtaiset löydökset.

Kuvassa 8 näkyi, että laitteesta *192.168.56.106* oli löytynyt *low*-tason haavoittuvuus. Haavoittuvuuksia OpenVas-työkalussa on helppo tarkastella ja OpenVas tarjoaakin selitteet kaikille löytämilleen haavoittuvuuksille, kuten kuvassa 10.



Kuva 10. OpenVas-ohjelmalla löytynyt haavoittuvuus.

OpenVasin käyttö on vähän monimutkaisempaa kuin Zenmap:n. Sillä on mahdollista tehdä onnistuneesti verkonkartoitusta, joka listaa verkossa olevat laitteet ja pirtää niistä yksinkertaisen verkkotopologian. Sillä on mahdollista havaita, jos verkkoon on tullut uusia laitteita, mutta hankala havaita, jos laitteen porteissa on tapahtunut muutoksia. Openvas näyttää hyvin ja selkeästi laitteiden haavoittuvuudet, johon se pääasiassa onkin tarkoitettu.

3 TOTEUTUS

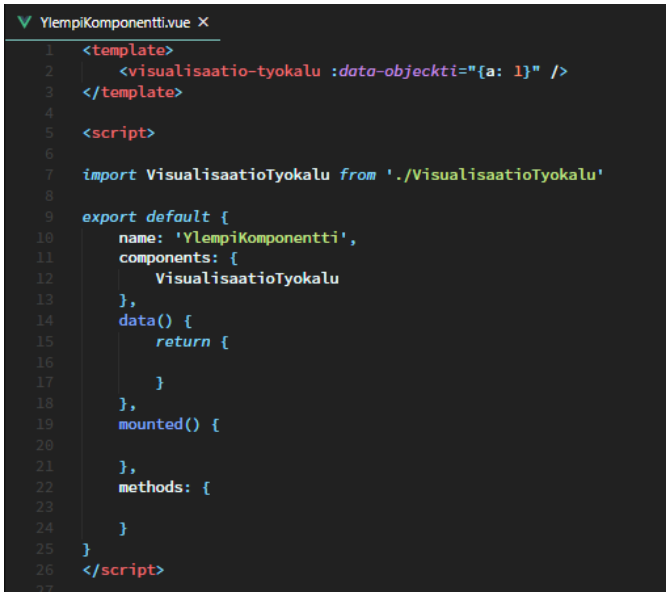
Opinnäytetyön toteutus lähti käyntiin kartoittamalla yrityksen X toiveet visualisointityökalun suhteen. Yrityksen edustajat toivoivat, että verkon visualisointia kuvattaisiin ympyrämuodostelmassa. Visualisoinnista pitäisi olla mahdollisuus klikata yksittäistä solmua, jolloin visualisaatio muuttuisi ja näyttäisi ainoastaan klikatun solmun ja sen kaikki alisolmut. Visualisoinnista toivottiin visuaalisesti näyttävää ja olisi hyvä, jos siinä näkyisi yrityksen X värit. Lähdin tutkimaan mahdollisia JavaScript-kirjastoja, joilla olisi mahdollista toteuttaa kyseinen visualisaatio. Käytyäni eri JavaScript-kirjastoja läpi päädyin Sigma.js-kirjastoon, koska se on monipuolinen ja muokattavissa oman tarpeen mukaan. Sigma.js-kirjasto mahdollistaa solmujen piirtämisen, niiden yhteen liittämisen, graafin liikuttelun ja graafin suurentamisen ja pienentämisen. Piirrettäessä solmuille on mahdollista määritellä muoto ja väri, ja ne on mahdollista piirtää svg- tai canvas-muodossa. Lisäksi Sigma.js-kirjastolla on erittäin kattava dokumentaatio, joten kaikkia sen ominaisuuksia on helppo soveltaa ja käyttää. Sigma.js on hyvin ylläpidetty ja siitä on riippuvaisia yli 400 JavaScript-kirjastoa github.com sivuston mukaan (GitHub 2021).

3.1 Vue.js-komponentin luonti

Yrityksellä X on verkkoportaalin käyttöliittymässä käytössä Vue.js. Se on avoimen lähdekoodin JavaScript-ohjelmistokehys reaktiivisten käyttöliittymien luomiseen. Se on keskittynyt ainoastaan kerrokseen, joka ajetaan selaimessa. (Vue.js 2021) Jotta visualisaatiotyökalusta tulisi mahdollisimman uudelleenkäytettävä, tein siitä oman vue-komponentin. Vue-komponentin luonti aloitetaan luomalla *VisualisaatioTyokalu.vue*-tiedosto. Koodissa 1 on esimerkki vue-komponentin sisällöstä. *Template*-tägien sisään tulee kaikki HTML-koodi ja kaikki JavaScript-koodi tulee *script*-tägien sisälle. Vue-komponentin nimi annetaan *name*-ominaisuudessa rivillä 7. *Props*-ominaisuudessa listataan kaikki ominaisuudet, mitä tälle komponentille voidaan syöttää tätä komponenttia kutsuttaessa. *Data*-funktion *return*-objektin sisään määritellään muuttujat, joista tämä vue-komponentti pitää kirjaa. *Mounted*-funktiota kutsutaan heti, kun tämä vue-komponentti on liitetty HTML-dokumenttipuuhun (Vue.js 2021). *Methods*-objektin sisälle määritellään kaikki metodit, joita tämä komponentti käyttää.

Koodi 1. Esimerkki vue-komponentista.

Kun *VisualisaatioTyokalu.vue*-komponentti on valmis ja sitä halutaan käyttää projektissa, kutsutaan sitä koodin 2 esimerkin mukaisesti.



Koodi 2. Vue-komponentin kutsuminen.

Koodissa 2 rivillä 7 *VisualisaatioTyokalu.vue* tuodaan *import*-lausekkeella, jonka jälkeen se lisätään vue-komponentin *components*-objektin sisään. *Components*-objektissa on listattuna kaikki tämän vue-komponentin käyttämät komponentit. Nyt sitä on mahdollista käyttää *<visualisaatio-tyokalu>* tägillä. HTML-tägit ja -attribuutit eivät erota isoja ja pieniä kirjaimia, minkä takia jotkin selaimet muuttavat ne automaattisesti pieniksi kirjaimiksi.

Tämän takia Vue:ssa käytetään *kebab-case*-systeemiä. Eli kutsuttaessa *VisualisaatioTyokalu.vue*-komponenttia on sen html-tägi *visualisaatio-tyokalu* ja *dataObjekti*-ominaisuus *data-objekti*, kuten koodin 2 esimerkissä. *Data-objekti*-attribuuttia edeltävä kaksoispiste on lyhenne vue:n ”*v-bind:”*-ominaisuudesta, joka sitoo dynaamisesti datan kyseiseen elementtiin. Jos tuo data muuttuu, niin vue päivittää kyseisen elementin. (Vue.js 2021)

3.2 Sigma.js-kirjasto

Sigma.js on verkkograafien tekemiseen tarkoitettu avoimen lähdekoodin JavaScript-kirjasto. Se on alusta, jolla voi luoda interaktiivisia visualisointigraafeja nykyaikaisissa verkkosovelluksissa. Se mahdollistaa kuuntelemaan käyttäjän klikkauksia, raahauksia ja hiiren rullan liikkeitä. Sigma myös tietää aina graafin asemoinnin suhteessa näyttöön. Tämä tarkoittaa, että Sigmalla piirretyissä graafeissa on mahdollista suurentaa ja pienentää kuvaa, kuin myös liikkuttaa graafia ruudulla raahaamalla. (Jacomy 2019)

Koska Sigma.js on JavaScript-kirjasto ja se on rekisteröity JavaScript-paketinhallintarekisteri Npm:ään, on se mahdollista asentaa käyttäen Npm-paketinhallintaohjelmaa. Npm on JavaScriptin oma paketinhallintaohjelma, joka asentuu Node.js:n mukana. Sigma.js:n viimeisin vakaa versio on v1.2.1 ja kaikki dokumentaatiot, jotka Sigma.js:n wikissä on, koskevat tätä versiota. Npm:n verkkosivuilla kuitenkin on julkaistuna beta-versio v2 Sigma.js:stä, joten asentaessa Sigmaa pitää erikseen kertoa, että asennettava kirjasto on versio v1.2.1. Tämä onnistuu komennolla *npm install sigma@1.2.1*.

Koodissa 3 on esimerkki Sigma-instanssin luonnista. Rivillä 6 tuodaan Sigma *import*-lausekkeella, jonka jälkeen *mounted*-funktiossa kutsutaan *renderSigma*-funktiota. *New*-avainsanalla luodaan uusi Sigma-instanssi, jolle annetaan argumentteina *graph*-objekti ja *renderer*-objekti. *Graph*-objekti pitää sisällään listan piirrettävistä solmuista ja listan piirrettävistä solmuja yhdistävistä kaarista. *Renderer*-objektissa määritellään, miten ja mihin graafi tullaan piirtämään. *Container*-ominaisuudessa määritellään elementti, johon graafi luodaan. *Type*-ominaisuudessa määritellään graafin renderöintityyppi. Tässä tapauksessa käytän *canvas*-renderöintityyppiä, koska se on suorituskyvyltään kevyempi ja selaintuki on laajempi kuin toisena vaihtoehtona oleva WebGL. Sigma.js:n wiki-sivulla sanotaan, että *canvas*-piirtäjä luo yhteensä neljä eri päällekkäistä HTML *canvas*-elementtiä. Ensimmäiselle *canvas*-elementille luodaan solmut, toiselle solmuja yhdistävät kaaret, kolmannelle solmujen otsakkeet ja neljännellä *canvas*-elementillä on

tapahtumakuuntelija, joka tarkkailee käyttäjän hiiren tapahtumia ja piirtää leijuvat otsakkeet hiiren osuessa solmujen kohdalle. (Jacomy 2019)

Elementtejä tarkastellessa selaimessa voi kuitenkin huomata, että Sigma luo ainoastaan kaksi *canvas*-elementtiä, jossa ensimmäisellä *canvas*-elementillä on piirrettynä solmut, solmut yhdistävät kaaret ja otsakkeet ja toisella *canvas*-elementillä on käyttäjän syötteiden tapahtumakuuntelija. Kuvassa 11 näkyy *canvas*-elementit selaimen *elements*-työkalussa.

```
VisualisaatioTyokalu.vue U
<template>
  <div id="sigma-container"></div>
</template>

<script>
import sigma from 'sigma'

export default {
  name: 'VisualisaatioTyokalu',
  props: ['dataObjekti'],
  data() {
    return {
      sigma: null,
      nodes: [{id: '1', label: 'Node 1', x: 0, y: 0, size: 1, color: '#000'}, {id: '2', label: 'Node 2', x: 0, y: 20, size: 1, color: '#000'}],
      edges: [{id: 'edge-#1-#2', source: '1', target: '2'}]
    }
  },
  mounted() {
    this.renderSigma()
  },
  methods: {
    renderSigma() {
      this.sigma = new sigma({
        graph: {
          nodes: this.nodes,
          edges: this.edges
        },
        renderer: {
          container: document.getElementById('sigma-container'),
          type: 'canvas',
          camera: 'kamera'
        }
      })
      this.sigma.refresh()
    }
  }
}
</script>
```

Koodi 3. Sigma-instanssin luonti.

```
<div data-v-6eca5bae id="sigma-container">
  <canvas class="sigma-scene" width="313px" height="313px" style="position: absolute; width: 313px; height: 313px;">
  <canvas class="sigma-mouse" width="313px" height="313px" style="position: absolute; width: 313px; height: 313px;">
</div>
```

Kuva 11. Elementtien tarkastelu selaimessa.

Rivillä 31 Koodissa 3 määritetään *camera*-ominaisuudelle nimi. Sigman *camera*-ohjelmointirajapinta toimii aivan kuten videopeleissä, se mahdollistaa graafeissa liikkumisen hiirellä ja zoomauksen hiiren rullalla. Sigma-graafi toimii kyllä ilman kameran nimeämistä, mutta nimeämällä kameran, on mahdollista käyttää *camera*-ohjelmointirajapinnan metodeita, kuten päivittää kameran sijainti alkuasentoon syöttämällä kameralle uudet

koordinaatit *sigInst.cameras.kameranNimi.goTo(0,0, 1)*. *GoTo*-metodi ottaa parametreina x-koordinaatin, y-koordinaatin ja zoom-ration. (Jacomy 2019)

3.3 Solmujen ja kaarien paikoitus-algoritmi

Sigma.js-graafissa pitää jokaiselle solmulle ja solmuja yhdistävälle kaarelle määrittää omat x- ja y-koordinaattinsa. Data, joka graafissa visualisoidaan, tulee palvelimelta JSON-formaatissa. JSON on lyhenne sanoista JavaScript Object Notation. Se tarkoittaa, että data on jäsennelty JavaScript-objektin syntaksin mukaisesti. (MDN contributors 2021) Koodissa 4 on esimerkki JSON-objektista. Objekteilla voi olla lapsiobjekteja, joilla taas voi olla lisää lapsiobjekteja.

```
let solmu = {
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
    }
  ]
}
```

Koodi 4. Esimerkki JSON-objektista (MDN contributors 2021).

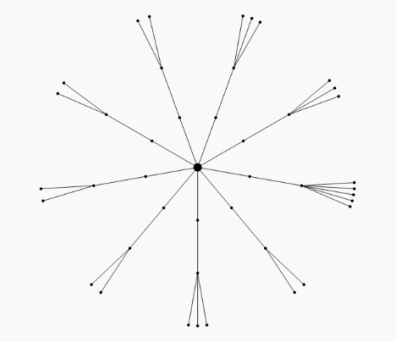
Yritys X:n toiveena oli, että solmut näkyisivät graafissa pyöreässä muodostelmassa. Ajattelin, että paras tapa lähestyä tätä ongelmaa olisi rekursiivinen funktio, joka tarkoittaa funktiota, joka kutsuu itseään uudestaan ja uudestaan. Esimerkiksi tämän graafin tapauksessa, jos olisi rekursiivinen funktio nimeltä *luoSolmu*. Sen ollessa ensimmäisellä rekursiokierroksella se luo ylemmän tason solmun, jonka jälkeen ohjelma kutsuu itseään ja antaa argumenttina tämän rekursiokierroksen solmun lapsisolmun.

Funktion pitäisi pitää kirjaa ylemmän solmun koordinaateista, ylemmän solmun id:stä, solmut sisältävästä objektista, ylemmän solmun etenemiskulmasta ja ylemmän solmun käyttämästä kulmasta. Koodissa 5 on pseudokoodi suunnittelemastani algoritmista.

```
22      laskeSolmujenPaikoitus(YlemmänsolmunXjaY, YlemmänSolmunId, SolmutObjekti, YlemmänSolmunEtenemisKulma, YlemmänSolmunKäyttämäKulma) {
23        let YlemmänSolmunX = YlemmänsolmunXjaY[0]
24        let YlemmänSolmunY = YlemmänsolmunXjaY[1]
25        let solmujenLukumäärä = LaskeSolmut(SolmutObjekti)
26
27        let kulmaEtenemä = YlemmänSolmunKäyttämäKulma / solmujenLukumäärä
28        let tasausKulma = (kulmaEtenemä * solmujenLukumäärä - 1) / 2
29
30        Object.keys(SolmutObjekti).forEach((solmu, kierros) => { //Käy läpi kaikki solmut
31          let tämänSolmunId = YlemmänSolmunId + ',' + solmu
32          let tämänSolmunX = YlemmänSolmunX + (Säde * Math.sin(Math.PI * 2 * (kierros * kulmaEtenemä + YlemmänSolmunEtenemisKulma + tasausKulma) / 360))
33          let tämänSolmunY = YlemmänSolmunY + (Säde * Math.cos(Math.PI * 2 * (kierros * kulmaEtenemä + YlemmänSolmunEtenemisKulma + tasausKulma) / 360))
34
35          teeSolmu({
36            id: tämänSolmunId,
37            label: solmu,
38            x: tämänSolmunX,
39            y: tämänSolmunY
40          })
41
42          jos(tämäOnArray(SolmutObjekti[solmu]) || tämäOnObjekcti(SolmutObjekti[solmu])) {
43            this.laskeSolmujenPaikoitus([tämänSolmunX, tämänSolmunY], (kierros * kulmaEtenemä + tasausKulma), kulmaEtenemä)
44          }
45
46          teeSolmutYhdistäväKaari({
47            id: 'edge#' + YlemmänSolmunId + '-' + tämänSolmunId,
48            source: YlemmänSolmunId,
49            target: tämänSolmunId
50          })
51        });
52      },
```

Koodi 5. Pseudokoodi solmujen paikoitus-algoritmista.

Solmujen kulman etenemä lasketaan jakamalla nykyisten solmujen määrällä ylemmän solmun käyttämä kulma, jonka jälkeen lasketaan tasauskulma, eli jos kulman etenemä on 45 astetta ja solmujen lukumäärä on 2, niin tasauskulman tulee olla 45 / 2, jotta solmut lähtevät oikeaan suuntaan suhteessa ylempään solmuun. Tämän jälkeen ohjelma käy jokaisen objektiavaimen läpi ja tekee siitä oman solmunsa. Id:ksi tulee ylemmän solmun id yhdistettynä nykyiseen objektiavaimeen, jotta jälkeen päin olisi mahdollista paikantaa juuri tämä objekti alkuperäisestä *dataObjektista*, joka ohjelmalle syötettiin. Se mahdollistaa ominaisuuden, että klikkaamalla solmua pystyy avaamaan kyseisen solmun kaikki alisolmut muiden solmujen mennessä piiloon. Rivillä 42 tarkastetaan, että jos nykyinen solmu on lista tai objekti, niin ohjelma kutsuu itseään syöttämällä tämän nykyisen solmun argumenttina funktioon. Viimeisenä yhdistetään nykyinen solmu ja nykyistä solmua kutsunut solmu kaarella toisiinsa. Kuvassa 12 on graafi paikoitus-algoritmin paikoittamilla solmuilla ja kaarilla.



Kuva 12. Graafi.

3.4 Graafin solmujen ulkoasun muokkaaminen

Solmuille on mahdollista luoda uusia piirustusmetodeita omien tarpeiden mukaan käyttäen HTML *canvas*-piirustustekniikkaa kuten Koodi 6 esimerkissä (Jacomy 2019).

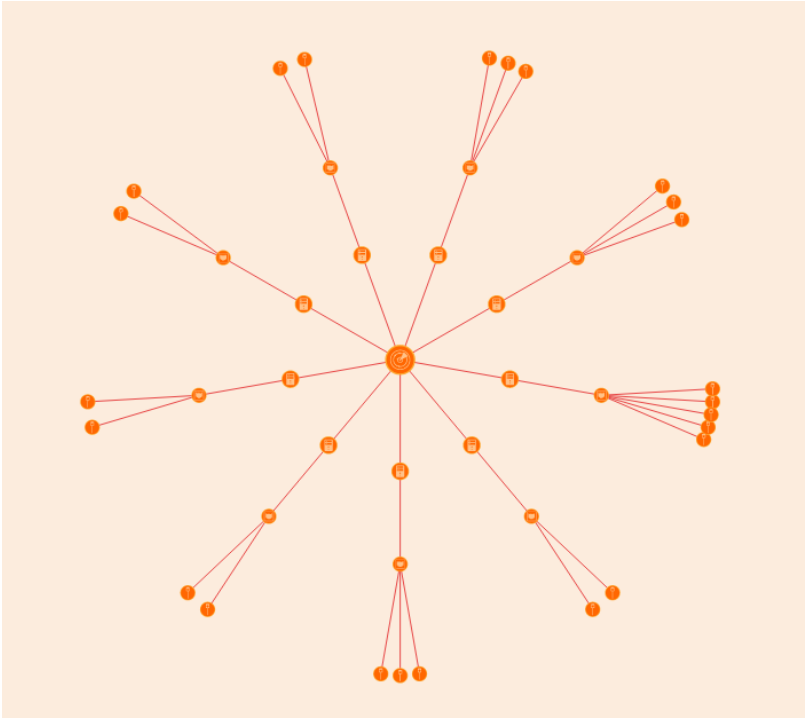
```
sigma.canvas.nodes.nelikulmio = function(node, context, settings) {
var prefix = settings('prefix') || '',
size = node[prefix + 'size'];

context.fillStyle = node.color || settings('defaultNodeColor');
context.beginPath();
context.rect(
node[prefix + 'x'] - size,
node[prefix + 'y'] - size,
size * 2,
size * 2
);

context.closePath();
context.fill();
};
```

Koodi 6. Solmun piirustusmetodin luonti (Jacomy 2019).

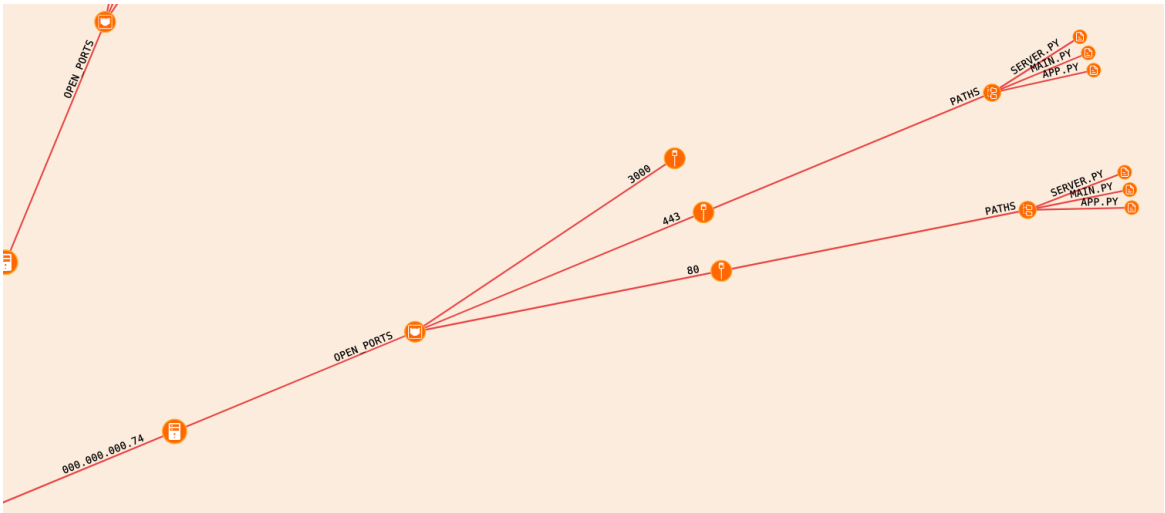
Koodissa 6 luodaan uusi piirustusmetodi solmulle, jonka nimi on *nelikulmio*. Se on funktio, joka ottaa argumentteina vastaan solmun, kontekstin ja asetukset. Konteksti tarkoittaa tässä tapauksessa *canvas*-elementin *getContext*-metodia. Solmun kutsuessa tätä *nelikulmio*-metodia solmulle on määritelty väri ja koko. *Context.rect*-metodi luo nelikulmion ja *context.fill*-metodi täyttää nelikulmion solmussa määritellyllä värillä. Samalla menetelmällä on mahdollista luoda myös otsakkeille ja kaarille omia piirustusmetodeita.

Yritys X antoi *png*-kuvakkeita, joita tulisi käyttää graafissa. Muutin *png*-kuvakkeet HTML *canvas*-muotoon käyttäen Inkscape-kuvankäsittelyohjelmaa. Tämän jälkeen kirjoitin pythonilla yksinkertaisen skriptin, joka muuttaa x- ja y-koordinaatit dynaamisiksi, kuten koodissa 6. Kuvakkeiden käyttöönotto graafissa vaatii, että solmuille annetaan *type*-

ominaisuus. Kirjoitin funktion, jolta voidaan kysyä jokaisen solmun kohdalla, mihin solmutyyppiin tämä solmu kuuluu. Funktiolle syötetään argumenttina solmun id ja solmua kutsuneen solmun id ja funktio palauttaa solmutyypin. Kuvassa 13 on graafi solmutyyppien määrittämisen jälkeen. Kuvassa 14 on suurennettu kuva graafista. Suurennetussa kuvassa tulee näkyviin solmujen otsakkeet.



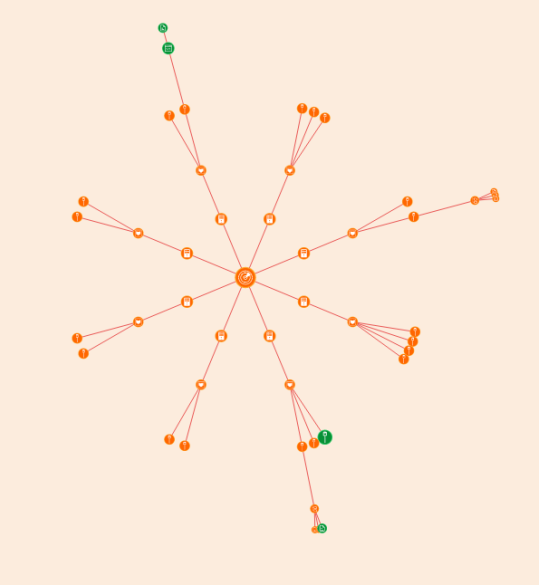
Kuva 13. Graafi solmutyyppien määrittelyn jälkeen.



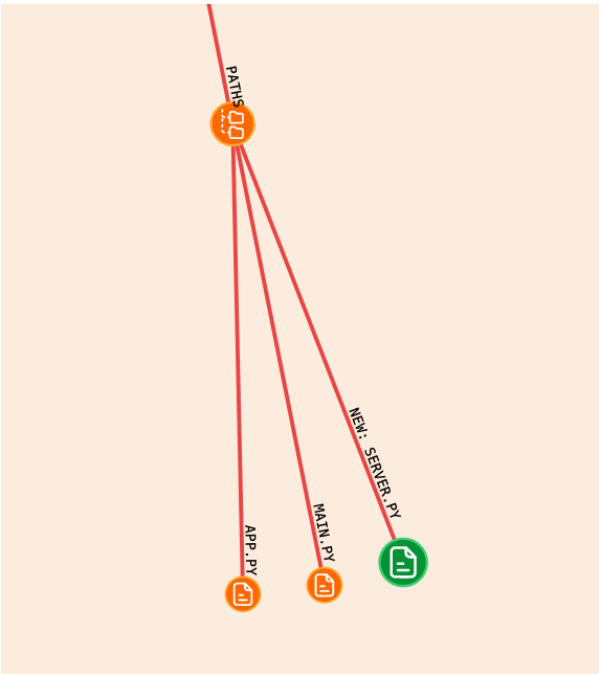
Kuva 14. Graafi suurennettuna.

3.5 Eroavaisuuksien esittäminen graafissa

Graafissa on helppo esittää erot kahden eri verkonkartoitustuloksen välillä. Vihreät solmut ovat uusia tuloksia ja punaisena loistavat solmut ovat poistuneita tuloksia. Jotta olisi mahdollista esittää nämä tulokset, pitää olla ylipäänsä kaksi tulosta jota vertailla ja *VisualisaatioTyökalu.vue*-komponenttiin täytyy lisätä kolme uutta propsia: ”halutaanko eroavaisuudet näyttää”-totuusarvo, lista poistuneista id:stä ja lista lisätyistä id:stä. Itse *VisualisaatioTyökalu.vue*-komponentin solmun paikoitus-algoritmiin ei tarvita muita muutoksia kuin *if*-tarkastus, löytyykö kyseisen solmun id jommasta kummasta listasta, lisätyistä tai poistuneista. Komponentti, joka on vastuussa *VisualisaatioTyökalu*.vue-komponentin kutsumisesta, purkaa ensin molemmat vertailtavat verkonkartoitustulokset samaan muotoon kuin solmujen id:t, eli merkkijoinoiksi, jotka ovat mallia ’objektiavain1,lapsiobjektiavain1,lapsiobjektinobjektiavain1’. Tämän jälkeen käydään molempien objektiavainten listat läpi ja verrataan niitä toisiinsa. Jos ensimmäisestä tuloksesta löytyy jokin tulos, jota ei ole enää seuraavassa tuloksessa, lisätään kyseinen tulos punaisella merkattavien listaan, ja jos jotain toisessa olevassa objektiavainten listassa olevaa ei löydy ensimmäisestä listasta, lisätään se vihreillä merkattavien listaan. Kuvassa 15 näkyy, miten helppoa on havaita uudet solmut kahden verkonkartoitustuloksen välillä. Kuvassa 16 on suurennettu kuva uudesta solmusta.



Kuva 15. Graafi näyttää uudet solmut.



Kuva 16. Suurennettu kuva eroavaisuuksien vertailusta.

3.6 Alisolmun solmujen tarkempi tarkastelu

Useat *Sigma.js*-objektit voivat käynnistää tapahtumia, kuten esimerkiksi solmujen ja kaarien klikkaus tai *hooveeraminen.* Sen lisäksi, että niillä on omat oletustapahtumankäsittelijät, on niille mahdollista kirjoittaa *sigma.classes.dispatcher*-ohjelmointirajapinnan avulla omia tapahtumankäsittelijöitä. (Jacomy 2019) Koodissa 7 on esimerkki tapahtumakuuntelijan rekisteröinnistä.

```
this.sigma.bind('clickNode', node => {
  this.poistaKaikkiSolmutjaKaaret()
  let lapsiSolmuObjekti = this.dataObjekti
  let klikatunSolmunId = node.data.node.id
  let klikatunSolmunLabel = node.data.node.label

  klikatunSolmunId.split(',').forEach(x => {
    lapsiSolmuObjekti = lapsiSolmuObjekti[x]
  })

  teeSolmu({
      id: klikatunSolmunId,
      label: klikatunSolmunLabel,
      x: 0,
      y: 0
  })

  this.laskeSolmujenPaikoitus([0, 0], lapsiSolmuObjekti, 0, 360)
})
```

Koodi 7. Sigma-tapahtumakuuntelijan rekisteröinti.

Koodin 7 esimerkissä rivillä 66 rekisteröidään *clickNode*-tapahtumankäsittelijä. Tapahtuman käynnistyessä anonyymi-funktio saa argumentikseen klikatun solmun. Funktio poistaa kaikki graafissa jo olemassa olevat solmut ja kaaret ja asettaa *lapsiSolmuObjekti*-objektiin nykyisen *dataObjekti*-objektin. Rivillä 72 klikatun solmun *id*-merkkijono jaetaan taulukoksi erottelemalla merkkijono jokaisesta pilkusta. Näin saadaan lista objektiavaimista, jotka osoittavat klikatun solmun lapsiobjektiin. Käyttämällä *forEach*-metodia ja asettamalla *lapsiSolmuObjekti*-objektiin silloisen vuorossa olevan objektiavaimen, löydämme lopulta oikean objektin, joka rivillä 83 annetaan argumentiksi aiemmin luodulle *laskeSolmujenPaikoitus*-funktiolle, joka luo graafin uudelleen.

3.7 Graafien esittäminen aikajanassa

Graafien esittämiseen aikajanamuodossa loin uuden vue-komponentin, joka ottaa propsina vastaan taulukon skannaustuloksia. Vuen *v-for*-direktiivillä luon *div*-elementtejä, jotka toimivat graafin kontainereina, niin monta kuin vastaanotetun skannaustulosten taulukon pituus on. Koodissa 8 on esimerkki *v-for*-direktiivin käytöstä.

```
<template>
<div>
  <div v-for="i in dataObjektiTaulukko.length" :key="`sigma-container#${i}`" :id="`sigma-container#${i}`"></div>
</div>
</template>
```

Koodi 8. Vuen *v-for*-direktiivin käyttö.

Vue tarvitsee *v-for*-direktiivia käytettäessä *key*-arvon jokaiselle elementille. Modernissa web-ohjelmassa *DOM*-elementtien paikat voivat muuttua, kun käyttöliittymä poistaa ja lisää elementtejä sitä mukaa, mitä käyttäjä sivuilla tekee. Tätä varten Vue käyttää elementeissään *key*-arvoa, jottei Vuen tarvitse käydä läpi koko elementti-listaa päivittäessään yhtä elementtiä, vaan löytää oikean elementin *key*-indexillä. (Vue.js 2021)

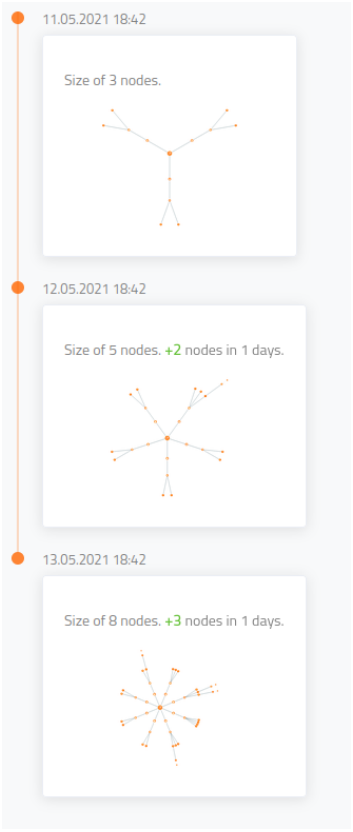
Kun *VisualisaatioTyokalu.vue*-komponentissa *mounted*-funktiossa kutsuttiin *renderSigma*-funktiota, joka piirsi graafin, *AikajanaTyökalu.vue*-komponentissa *mounted*-funktiossa on *forEach*-metodi, joka käy läpi kaikki skannaustulokset ja lähettää ne indekseineen *renderSigma*-funktiolle. *RenderSigma*-funktio luo uuden *Sigma*-instanssin, johon se asettaa *container*-ominaisuudeksi *'graph-container-<argumenttina saatu indexi>'*. Sen sijaan, että luotaisiin yksi *this.sigma*-objekti, joka pitää sisällään *Sigma*-instanssin, niin lisätään *Sigma*-instanssi taulukkoon. Kun graafi on renderöity *canvas*-elementille,

niin muutetaan se *HTMLCanvasElement.toDataURL-metodilla data-URL*-muotoon. Se muuttaa *canvas*-elementin *png*-muotoon ja tekee siitä merkkijonon, joka on mahdollista tallettaa taulukkoon. Koodissa 9 esimerkki *canvas*-elementin muuttamisesta *data-URL*-muotoon. Kun *canvas*-elementti on muutettu *data-URL:ksi* ja lisätty taulukkoon, joka pitää sisällään *png*-kuvat, voidaan se poistaa *html*-dokumenttipuusta.

```
var canvas = document.getElementById('canvas');
var dataURL = canvas.toDataURL();
console.log(dataURL);
// "
// blAAAADElEQVQImWNgoBMAAABpAAFEI8ARAAAAAElFTkSuQmCC"
```

Koodi 9. *Canvas*-elementin muuttaminen *data-URL*-muotoon (MDN contributors 2021).

Data-URL:a on mahdollista käyttää normaalina kuvana *img*-tägin kanssa syöttämällä se *img*-tägin *src*-attribuuttiin. Nyt on mahdollista luoda aikajana käyttämällä Vuen *v-for*-direktiiviä *img*-tägin sisällä syöttämällä *v-for*-direktiiviin taulukon, joka pitää sisällään kaikki *data-URL*-merkkijonot. Kuvassa 17 on skannaustulokset aikajanassa.



Kuva 17. Skannaustulokset aikajanassa.

4 LOPUKSI

Opinnäytetyön tavoitteena oli luoda yritykselle X verkonkartoitustulosten visualisointityökalu yrityksen jo olemassa olevaan verkkoportaaliin. Visualisointityökalulla piti olla mahdollista tarkastella verkon nykytilannetta, vertailla eroavaisuuksia kahden verkonkartoitustuloksen välillä ja katsoa kuinka verkko on muuttunut ajan saatossa. Visualisoinnin piti olla sen verran helposti luettava, että yritys X:n asiakkaatkin osaisivat lukea visualisointia.

Työn tuloksena syntyi toimiva visualisointityökalu. Työkalulla on mahdollista näyttää verkonkartoitustulos verkkotopologiana ja siitä näkee myös avoimet portit ja porteissa olevat palvelut. Sillä on mahdollista vertailla graafisesti eroja kahden eri verkonkartoitustuloksen välillä ja tarkastella verkonkartoituksia aikajanassa. Verkonkartoitustuloksien vertailussa työkalu esittää selkeästi uudet laitteet ja poistuneet laitteet.

Verrattuna Zenmap:n kartoitustuloksien näyttämiseen, tekemässäni toteutuksessa kaikki saatu verkonkartoitustieto löytyy samasta näkymästä. Käyttäjän ei tarvitse klikkailla eri valikoita löytääkseen etsimäänsä tietoa. Tekemässäni toteutuksessa näkyy laitteiden kaikki avoinna olevat portit, kun taas Zenmap:ssa laitteet olivat vain värikoodattuna, sen mukaan montako porttia laitteessa oli auki. Vertaillessa kahta erillistä verkonkartoitustulosta Zenmap näytti tulokset ainoastaan tekstitiedostona. Omassa ratkaisussani näitä kahta kartoitustulosta on mahdollista vertailla samassa verkkopologian visualisoinnissa. Tämä helpottaa näkemään koko verkon kuvan, mitä on löytynyt ja mistä tai mikä on poistunut ja mistä. Kaiken tämän tiedon näkee yhdestä kuvasta.

OpenVas työkalussa laitteet olivat listattuna taulukossa ja pienestä kuvasta oli mahdollista nähdä yksinkertainen visualisaatio verkkotopologiasta. Nähdäkseen yksittäisen laitteen avoimet portit piti avata laitekohtaiset löydökset. Siinä oli listattuna allekkain kaikki, mitä laitteelta oli löydetty. Tähän listaan tuli siis useamman kartoituksen tulokset. Havaitakseen laitteella muutoksia avatuissa tai suljetuissa porteissa piti verrata *created*-kellonaikaa, milloin portti oli löytynyt. Laitekohtaisia muutoksia OpenVas-työkalulla oli siis erittäin hankala seurata. OpenVas kuitenkin esitti laitteilta löytyneet haavoittuvuudet todella tehokkaasti, tätä ominaisuutta tekemässäni toteutuksessa ei ole. Tämän kaltainen ominaisuus olisi mahdollista lisätä, omaan toteutukseeni jälkikäteen ja se toisi työkalulle paljon lisäarvoa.

Yhdelläkään kolmesta tutkimastani verkonkartoitustyökalusta ei ollut mahdollista tarkastella suoritettuja verkonkartoitustuloksia aikajanassa. Tekemässäni toteutuksessa tämä on mahdollista. Verkonvalvontaan tämä ominaisuus ei välttämättä tuo hirveästi lisäarvoa, mutta yrityksen johto pystyy seuraamaan tästä verkon kehittymistä. Tämän avulla yrityksen johdon on esimerkiksi mahdollista ennustaa tulevia laitehankintakustannuksia.

Yritys X:n edustajat olivat visualisointityökaluun erittäin tyytyväisiä ja työn laatu ylitti heidän odotuksensa. Visualisointityökalu on nyt yrityksen verkkoportaalissa käytössä. Siellä sillä on mahdollista tarkastella verkonkartoitustuloksia ja mahdollisia hälytyksiä, joita syntyy jos verkonkartoituksissa tapahtuu muutoksia.

Visualisointityökalua olisi mahdollista jatkokehittää. Tällä hetkellä sillä on mahdollista visualisoida onnistuneesti yhden verkon laitteet. Lisäarvoa voisi tuoda, jos olisi mahdollista yhdistää samaan kuvaan useamman aliverkon tulokset. Olisi myös hyvä, jos visualisointi OpenVas:n tavoin näyttäisi mahdolliset haavoittuvuudet, jotka ovat tulleet ilmi verkonkartoituksessa.

LÄHTEET

Ghanem, Waheed Ali H. M., ja Bahari Belaton. ”Improving accuracy of applications fingerprinting on local networks using NMAP-AMAP-ETTERCAP as a hybrid framework.” *2013 IEEE International Conference on Control System, Computing and Engineering.* IEEE, 2013.

GitHub. *GitHub.* 2021. https://github.com/jacomyal/sigma.js/network/dependents (haettu 6. 6 2021).

Greenbone Networks. *OpenVas.* 1. 1 2021. https://www.openvas.org (haettu 21. 5 2021).

Jacomy, Alexis. *sigma.js wiki.* 2019. https://github.com/jacomyal/sigma.js/wiki (haettu 11. 4 2021).

Kan, Zhongyang, Changzhen Hu, Zhigang Wang, Guoqiang Wang, ja Xiaolong Huang. ”NetVis: A network security management visualization tool based on treemap.” *2010 2nd International Conference on Advanced Computer Control.* IEEE, 2010.

Kienzle, Darrell, Nathan Evans, ja Matthew Elder. ”NICE: Network Introspection by Collaborating Endpoints.” *2013 IEEE Conference on Communications and Network Security (CNS).* IEEE, 2013.

MDN contributors. *MDN Web Docs.* 19. 3 2021. https://developer.mozilla.org/fi/docs/Web/HTML (haettu 21. 3 2021).

Nmap. *Nmap Security Scanner.* 2019. https://nmap.org/ (haettu 8. 5 2021).

Oulun seudun ammattiopisto. *Verkon topologiat.* 2016. https://web.archive.org/web/20160305153536/http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/lahiverkko_internet/lanjaint/johdanto_verkkotekniikkaan/johdanto3.htm (haettu 10. 5 2021).

Vue.js. *Vue.js Documentations.* 2021. https://vuejs.org/v2/guide/ (haettu 30. 3 2021).