Jari Jääskelä

Sovelluskerroksen palvelunestohyökkäyksien toteuttaminen ja torjunta

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Sähkö- ja automaatiotekniikan koulutusohjelma
Huhtikuu 2017

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Huhtikuu 2017	**Tekijä/tekijät** Jari Jääskelä
Koulutusohjelma Sähkötekniikka		
Työn nimi SOVELLUSKERROKSEN PALVELUNESTOHYÖKKÄYKSIEN TOTEUTTAMINEN JA TORJUNTA.		
Työn ohjaaja FM Joni Jämsä		**Sivumäärä** 43
Työelämäohjaaja FM Anita Rättyä		

Tämä opinnäytetyö tehtiin CyberWI-tietoturvahankkeelle, jossa on mukana Centria-ammattikorkeakoulu. Opinnäytetyön tarkoitus oli selvittää miten vakavia sovelluskerron palvelunestohyökkäykset ovat ja miten niitä voidaan torjua. Hyökkäyksien kohteena käytettiin Magento-verkkokauppa-alustaa, joka asennettiin IIS-palvelimelle. Työssä testattiin kolmea erilaista hyökkäystä: Keep-Dead, tulvitus ja hidas. Hyökkäykset suoritettiin käyttäen Rugged Tooling Ruge -työkalua, jota käytetään verkon kapasiteetin testaukseen. Apuna hyökkäyksien laatimisessa käytettiin Wireshark- ja Fiddler-työkaluja.

Tuloksista osoittautui, että Keep-Dead- ja tulvitushyökkäykset kuormittavat paljon palvelinta ja verkkokaupan vastausaika voi nousta 3 minuuttiin tai se ei pysty vastaamaan ollenkaan käyttäjälle, vaikka hyökkäyksen kaistankäyttö olisi alle 1 Mbps. Hitailla hyökkäyksillä ei ole suurta vaikutusta IIS-palvelinta vastaan.

Hyökkäyksien vaikutuksia voidaan vähentää palvelimen kapasiteetin ja suorituskyvyn lisäämisen lisäksi käyttämällä käänteistä välityspalvelinta, joka toimii myös välimuistipalvelimena, johtaen varsinaisen palvelimen resurssien käytön vähenemiseen. On olemassa myös palveluita, jotka pyrkivät myös suodattamaan haitallisen liikenteen, kuten Cloudflare.

Asiasanat
Palvelunestohyökkäys, DOS, Magento, Verkkokauppa, HTTP, Tietoturva, Ruge, IIS

ABSTRACT

Centria University of Applied Sciences	**Date** April 2017	**Author** Jari Jääskelä
Degree programme Electrical Engineering		
Name of thesis EXECUTION AND MITIGATION OF APPLICATION LAYER DENIAL OF SERVICE ATTACKS.		
Instructor M.Sc Joni Jämsä		**Pages** 43
Supervisor M.Sc Anita Rättyä		

This thesis was done for CyberWI, which is a cybersecurity project, in which Centria University of Applied Sciences is involved in. The purpose of this thesis was to investigate how big a threat application layer denial of service attacks pose and how these attacks can be mitigated. Magento e-commerce platform was used as a target, which was installed on the IIS web server. Tested attacks include Keep-Dead, flooding and slow attacks. Attacks were performed using a Rugged Tooling Ruge, which is a network stress test tool. Wireshark and Fiddler tools were used as well to help with composing the attacks.

The results suggest that Keep-Dead and flooding attacks can have a major effect on the server's resource usage and the response time can rise up to 3 minutes, or in some cases, it cannot complete the user's request, even when the bandwidth usage of the attack is below 1 Mbps. Slow attacks did not seem to affect the IIS web server.

These attacks can be mitigated by increasing a server's capacity and performance, or by using a reverse proxy that can act as a cache server, thus lowering the resource usage of the server behind the reverse proxy. Services also exist that can mitigate these attacks by filtering malicious traffic, such as Cloudflare.

Key words
Cybersecurity, DOS, HTTP, Magento, Ruge, E-commerce, IIS

KÄSITTEIDEN MÄÄRITTELY

IP	Interner Protocol on protokolla, joka vastaa tietoliikennepakettien toimituksesta oikealle IP-osoitteelle.
MAC	Medium Access Control on protokolla, joka vastaa tietoliikennepakettien toimituksesta oikealle verkkokortille.
FastCGI	Fast Common Gateway Interface on binääri protokolla, jonka tarkoitus on vähentää web-palvelimen kuormitusta, jotta web-palvelin voi käsitellä suuremman määrän pyyntöjä.
Web-palvelin	Ohjelmisto, joka käsittelee palvelimelle tulevat pyynnöt.
CPU	Central Processing Unit on prosessori, joka suorittaa ohjelman konekielisiä käskyjä.
RAM	Random Access Memory eli keskusmuisti on muisti, johon ohjelman koodi ja sen tarvitsema muistialue ladataan.
MySQL	Relaatiotietokantajärjestelmä, jota käytetään tiedon tallentamiseen.
Eväste	Dataa, jonka verkkosivu tallentaa käyttäjän tietokoneelle. Esim. istunnonhallintaa varten.
Dynaaminen nettisivu	Sivu, jonka luonnin hallitsee palvelimella toimiva ohjelma.
Staattinen nettisivu	Sivu, joka on tiedosto ja toimitetaan sellaisenaan käyttäjälle.
Javascript	Ohjelmointikieli, jota käytetään muun muassa verkkosivuilla.

KUVIOT

TAULUKOT

1 JOHDANTO

Palvelunestohyökkäykset ovat suuri riski Internetille, koska ne voivat johtaa uutissivustojen, verkkopankkien, verkkokauppojen ja muiden tärkeiden palveluiden hidastumiseen tai toimimattomuuteen. Opinnäytetyön tavoitteena oli toteuttaa eri sovelluskerroksen palvelunestohyökkäyksiä Magento-verkkokauppa-alustaan, jotta saatiin selville kuinka vakavia sovelluskerroksen palvelunestohyökkäykset ovat. Tutkin myös, miten sovelluskerroksen palvelunestohyökkäyksiä voidaan torjua. Palvelun toiminnan hidastumisen lisäksi palvelunestohyökkäykset voivat avata muita tietoturvariskejä tai aiheuttaa tärkeiden tietojen vuotamisen hyökkääjälle tai muille käyttäjille.

Työssä tutustuttiin, kuinka eri työhön liittyvät protokollat toimivat ja mihin sovelluskerroksen palvelunestohyökkäykset sijoittuvat OSI-mallissa. Käsiteltävät hyökkäykset pohjautuvat HTTP-protokollaan. Selvitin myös, kuinka kuljetuskerroksen palvelunestohyökkäykset eroavat sovelluskerroksen hyökkäyksistä. Keskeisintä työssä oli sovelluskerroksen palvelunestohyökkäyksien suorittaminen ja tuloksien analysointi.

Käyttöönotin Centria-ammattikorkeakoulun tietoliikennelaboratorioon IIS-palvelimen, johon asensin Magento-verkkokauppa-alustan. Hyökkäyksien suorittamiseen käytin Rugged Tooling Ruge -työkalua. Hyökkäykset rakennettiin käyttäen tämän työkalun ohjelmistoa. Apuna käytin Wireshark- ja Fiddler-ohjelmia, jotka auttavat vianselvityksessä ja hyökkäyksien laatimisessa. Kehitin myös ohjelman palvelimen suorituskyvyn seurantaan.

2 PALVELUNESTOHYÖKKÄYKSET

Tietoturvallisuus koostuu kolmesta eri osa tavoitteesta: tietojen luottamuksellisuudesta, eheydestä ja saatavuudesta. Palvelunestohyökkäyksillä vaikutetaan tietojen saatavuuteen hidastamalla tai kaatamalla palvelu. Hyökkäyksiin yleensä käytetään heikosti suojattuja laitteita ja väärin määritettyjä palvelimia. (Viestintävirasto 2016.)

Hajautetussa palvelunestohyökkäyksessä haitallinen liikenne tulee useammalta eri laitteelta. Rikolliset tunkeutuvat laitteille käyttäen eri ohjelmistohaavoittuvuuksia ja haittaohjelmia. He sitten liittävät nämä laitteet bottiverkkoon, joka koostuu komentopalvelimesta ja etähallittavista laitteista. Tämän jälkeen bottiverkon hallitsija voi käyttää näitä laitteita palvelunestohyökkäyksessä. Nämä laitteet voivat olla reitittimiä, jääkaappeja, televisioita ja muita laitteita, joissa on Internet-yhteys. (Viestintävirasto 2016.)

Jos palvelussa on tietynlainen ohjelmistohaavoittuvuus, se voidaan saada tilaan, jossa tietojen luottamuksellisuus tai eheys voi vaarantua. Esimerkiksi joulukuussa 2015 Steam-palvelun välimuistipalvelimen väärä konfigurointi johti käyttäjien henkilökohtaisten tietojen näkymisen muille käyttäjille ruuhkan aikana (Steam 2015). Palvelunestohyökkäyksiä voidaan käyttää myös hämäyksenä, jolloin palvelua ylläpitävien henkilöiden huomio saadaan kiinnitettyä pois päähyökkäyksestä. (Viestintävirasto 2016.)

Palvelunestohyökkäyksen takana voi olla rahallinen etu, kuten kilpailijan toiminnan estäminen tai kiristäminen. Myös julkisuushakuisuus, kiusanteko ja erilaiset poliittiset motiivit voivat olla hyökkäyksen takana. (Viestintävirasto 2016.)

3 PALVELUNESTOHYÖKKÄYKSET OSI-MALLISSA

Open Systems Interconnection (OSI) -mallia käytetään nykyään verkon eri protokollien ja laitteiden toisiinsa vuorovaikutuksien kuvaamisessa. OSI-mallissa verkon protokollat ja laitteet jaetaan seitsemään kerrokseen (TAULUKKO 1). Alimpana on fyysinen kerros, joka hoitaa bittitason liikenteen koodauksen siirtomediumille. Siirtokerros huolehtii kehysten virheettömästä siirrosta, eli kehysten virheentarkistuksesta ja ruuhkanhallinnasta. Verkkokerros hoitaa mm. liikenteen reitityksen verkkojen välillä ja aliverkon ruuhkanhallinnan. Kuljetuskerros hoitaa viestien siirron ja niiden ruuhkanhallinnan. Kaksi yleisintä kuljetuskerroksen protokollaa ovat Transmission Control Protocol (TCP) ja User Diagram Protocol (UDP). Istuntokerros mahdollistaa eri laitteilla sijaitsevien sovellusten yhteyden muodostamisen, käyttämisen ja sulkemisen. Tätä kutsutaan istunnoksi. Esityskerros hoitaa merkkikoodauksen, datan pakkauksen, suojauksen ja muuntamisen. (Microsoft 2014.)

TAULUKKO 1. OSI-mallin kerrokset (mukaillen Microsoft 2014)

7. Sovelluskerros	Esim. HTTP ja FTP
6. Esityskerros	Datan esitys, suojaus ja pakkaus
5. Istuntokerros	Istunnon hallinta
4. Kuljetuskerros	Esim. TCP ja UDP
3. Verkkokerros	Reitin valinta ja IP (looginen osoite)
2. Siirtokerros	MAC (fyysinen osoite)
1. Fyysinen kerros	Bittitason liikenne

Palvelunestohyökkäyksiä voidaan toteuttaa fyysisessä kerroksessa, verkkokerroksessa ja sovelluskerroksessa. Fyysisen kerroksen palvelunestohyökkäys on katkos verkkoyhteydessä, esim. ethernet-kaapelin irrottaminen palvelimesta tai aiheutettu sähkökatkos. Kuljetuskerroksen hyökkäykset pyrkivät kuluttumaan kaiken palvelimen saatavissa olevan kaistan tai käyttämään hyväksi heikkouksia kuljetuskerroksen protokollissa. Nämä vaativat paljon resursseja hyökkääjältä, joten näitä kutsutaan myös volyymipohjaisiksi hyökkäyksiksi.

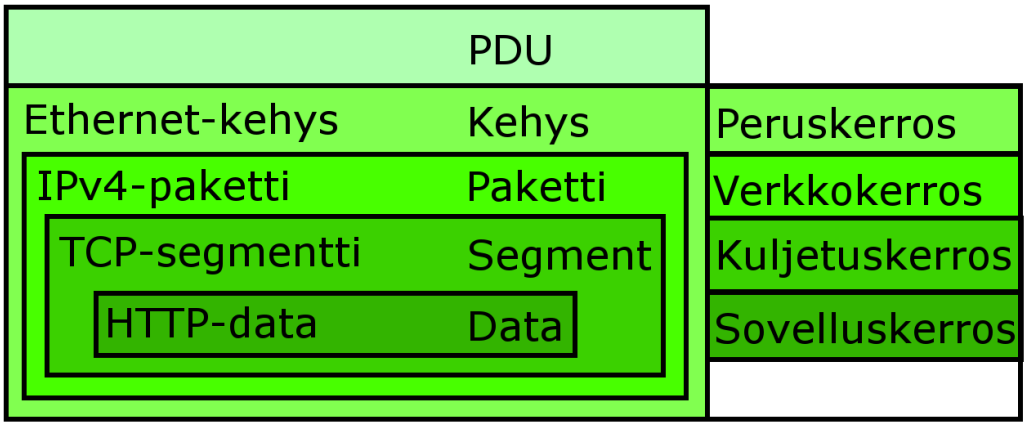
Sovelluskerroksen hyökkäykset käyttävät hyväksi heikkouksia kohde ohjelmassa tai sen protokollapinossa, johtaen palvelimen ohjelman kaatumiseen tai resurssien ehtymiseen. Sovelluskerroksen hyökkäykset kuluttavat enemmän kohde palvelimen resursseja kuin kuljetuskerroksen hyökkäykset, joten nämä vaativat vähemmän resursseja hyökkääjältä. Nämä ovat myös hankalempi huomata, koska hyökkääjien viestit ovat aitoja pyyntöjä palvelimelle.

4 TCP/IP

TCP/IP on joukko protokollia, jotka mahdollistavat tietokoneiden kommunikoinnin keskenään. TCP/IP koostuu neljästä kerroksesta: Perus-, verkko-, kuljetus- ja sovelluskerros (KUVIO 1). Peruskerrokseen kuuluvat käyttöjärjestelmän laiteajurit ja verkkokortti. Nämä hoitavat liikenteen muuntamisen siirtomediumille ja päinvastoin. Verkkokerros hoitaa pakettien reitityksen. Kuljetuskerros hoidetaan tiedon siirto kahden tietokoneen välillä. Kuljetuskerroksessa tähän tarkoitukseen käytetään TCP- ja UDP-protokollia. TCP tarjoaa luotettavan tiedonsiirron. UDP on tämän vastakohta, koska tämän protokollan tarkoituksena on mahdollisimman nopea tiedonsiirto, joten tiedon luotettavuutta ei tarkisteta. Tästä syystä tämä sopii esim. suoratoistolähetyksiin. Sovelluskerros hoitaa kommunikoinnin sovellusten välillä. Esim. HTTP-protokollaa käytetään verkkosivuilla. (Richard 1994, 16-17.)

Eri TCP/IP-kerroksissa käytetään eri datayksikköä. Tämä yksikkö on nimeltään Protocol Data Unit (PDU). Kuviossa 1 on esimerkki HTTP-protokollalla, miten kerrokset liittyvät toisiinsa ja mitä PDU-yksikköä käytetään eri kerroksissa.

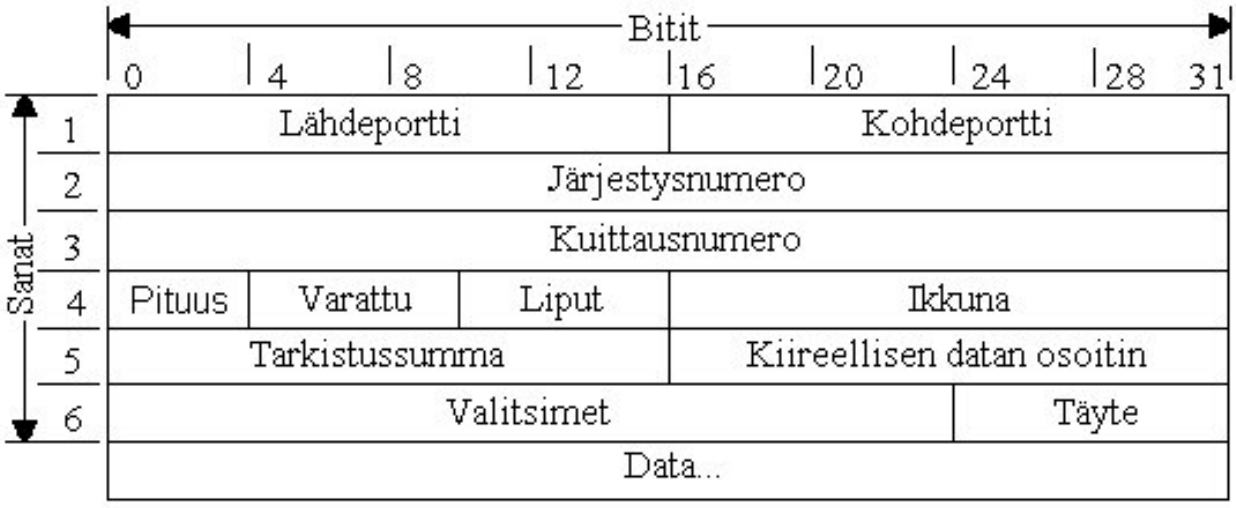
Termiä kehys käytetään peruskerroksen tasolla. Se saa nimensä siitä, kun ylempien kerroksien data kehystetään ”ylimääräisellä” otsikolla, jota tarvitaan paketin reititykseen. Termiä paketti ja datagram käytetään verkkokerroksen tasolla. Kuljetuskerroksessa käytetään termiä segment, jos protokollana on TCP. Jos käytetään UDP-protokollaa, terminä on datagram, koska se ei tue segmentointia, eli tiedon jakamista osiin. Sovelluskerroksessa ja sitä ylemmillä tasoilla käytetään termiä data. (Kozierok 2005, 160.)



KUVIO 1. TCP/IP-kerrokset

5 Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) on kuljetuskerroksen protokolla, joka käyttää asiakas-palvelin-mallia, jonka tarkoituksena on taata luotettava tiedonsiirto tietokoneiden välillä (Richard 1994, 242).



KUVIO 2. TCP-otsikko (Rintala 2001)

Kuviossa 2 on TCP-otsikon rakennekaavio. Otsikossa on lähde- ja kohdeportti. Portteja käytetään liikenteen ohjaamiseen tietylle sovellukselle. (Richard 1994, 244.) Esim. jos kaksi yhteyttä on auki verkkosivulle, ne käyttävät eri lähdeporttia asiakkaan näkökulmasta, jotta näiden yhteyksien liikenne ei menisi sekaisin.

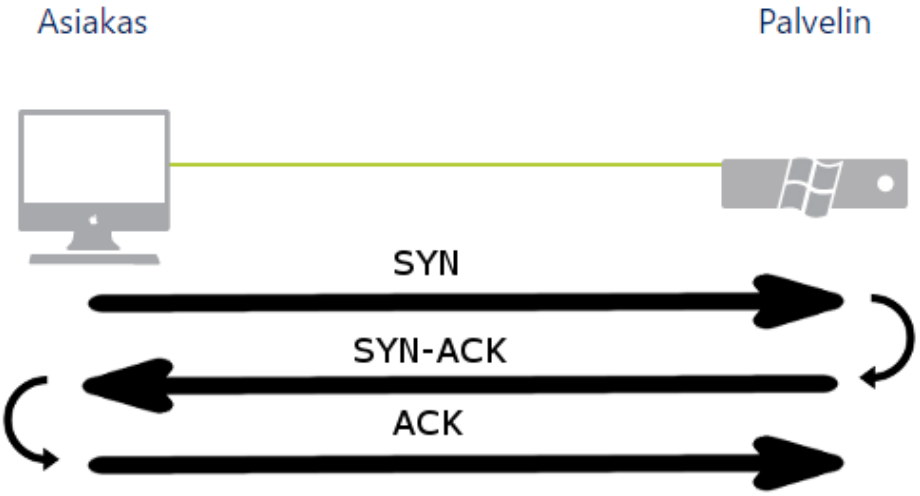
Koska segmentit voivat saapua perille väärässä järjestyksessä, on otsikossa myös järjestysnumero (sequence number), jotta voidaan muodostaa segmenteistä kokonainen viesti oikeassa järjestyksessä. Kuittausnumero (acknowledge number) on seuraava järjestysnumero, minkä lähettäjä odottaa saavan. Kiireellisen datan osoitin (urgent pointer) osoittaa missä kohtaa segmentissä on kiireellistä dataa, jos sitä on. Vastaanottaja päättää mitä tällä tiedolla tehdään. Otsikossa on myös tarkistussumma, jolla voidaan varmistaa, että segmentti ei ole korruptoitunut. Otsikon koko on 20 tavua, jos siinä ole valitsimia. (Richard 1994, 245.)

TAULUKKO 2. TCP liput -kentän bitit (mukaillen Richard 1994, 246)

URG (Urgent)	0010 0000
ACK (Acknowledge)	0001 0000
PSH (Push)	0000 1000
RST (Reset)	0000 0100
SYN (Synchronize)	0000 0010
FIN (Finish)	0000 0001

Taulukossa 2 on esitetty liput-kentän bitit. SYN-lippu tarkoittaa järjestysnumeroiden synkronointia Tämä tehdään yhteydenmuodostuksessa. PSH-lippu tarkoittaa, että data pitää ohjata sovellukselle. FIN-lippu ilmoittaa, että lähettäjä on valmis sulkemaan yhteyden. (Richard 1994, 246.)

Ennen tiedonsiirtoa täytyy TCP-yhteys muodostaa (KUVIO 3). Yhteyden muodostamisvaihe on kolmiosainen. Asiakas eli palvelun käyttäjä aloittaa yhteydenmuodostamisen lähettämällä palvelimelle synchronize (SYN) -viestin, jossa on alustava järjestysnumero, Initial Sequence Number (ISN), jonka lähettävä laite arpoo. Palvelin vastaa tähän omalla SYN-segmentillään, jossa on palvelimen arpoma ISN. Palvelin myös kuittaa asiakkaan lähettämän SYN-segmentin, lähettämällä asiakkaan ISN:n plus yksi acknowledge (ACK) -kentässä. Tätä kutsutaan SYN-ACK -segmentiksi. Asiakas lähettää vielä takaisin ACK-segmentin, eli asiakas kuittaa palvelimen lähettämän SYN-ACK -viestin. (Richard 1994, 250-251.)



KUVIO 3. TCP-yhteydenmuodostus (mukaillen Richard 1994, 251)

Sovelluksen lähettämä viesti jaetaan parhaimman kokoisiin segmentteihin luotettavuuden kannalta. Jos vastaanottaja saa viestin, se ilmoittaa lähettäjälle ACK-viestillä. (Richard 1994, 242-243.)

6 The Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) on sovelluskerroksen protokolla, jota useimmiten käytetään kommunikointiin selaimen ja verkkosivun välillä. HTTP käyttää TCP-protokollaa. Selain lähettää palvelimelle HTTP-pyynnön, johon palvelin vastaa. HTTP-protokolla on tilaton, eli siinä ei ole istunto käsitettä, vaan verkkosivu käyttää selaimen istunnon ylläpitämiseen evästeitä. (RFC 2616 1999, 1.)

6.1 HTTP-metodit

TAULUKKO 3. HTTP-metodit (mukaillen RFC 2616 1999, 51 - 57)

Metodi	**Selitys**
GET	Käytetään yleensä tiedonhakemiseen palvelimelta. Ei tue viesti runkoa.
POST	Tukee viesti runkoa. Käytetään yleensä tiedonsiirtämiseen palvelimelle.
PUT	URI-sijainnin resurssi lisätään/päivitetään.
DELETE	URI-sijainnin resurssi poistetaan.
OPTIONS	Palauttaa listan mitä metodeja URI tukee. Esim. (Allow: HEAD, GET, PUT, DELETE, OPTIONS).
HEAD	Sama kuin GET, paitsi vastaus ei sisällä runko-osaa.
CONNECT	Varattu välityspalvelimelle. Voidaan tunneloida liikenne.
TRACE	Käytetään virheenkorjaukseen. Kaiuttaa lähetetyn viestin takaisin lähettäjälle.

Taulukossa 3 on esitetty HTTP-metodit. Yleisempiä näistä metodeista ovat GET ja POST. GET-metodia käytetään mm. hakukoneissa ja sivujen lataamisessa, jolloin sivun sisältö on vastauksen runko-osassa. POST-metodia käytetään mm. lomakkeiden lähetykseen, jolloin lomake on runko-osassa.

6.2 HTTP-pyyntö

```
GET /päätepiste?parametri1=arvo&parametri2=arvo HTTP/1.1 [CRLF]
Host: www.esimerkki.fi [CRLF]
Accept: */* [CRLF]
[CRLF]
```

KUVIO 4. HTTP GET -pyyntö

Kuviossa 4 on HTTP GET -pyyntö, joka koostuu pyyntörivistä ja otsikoista. Rivit erotetaan toisistaan käyttämällä Carriage Return Line Feed (CRLF) -rivinvaihtoa. Pyyntörivillä määritetään mitä metodia käytetään, Uniform Resource Identifier (URI) ja protokollan versio. Pyynnössä voi olla myös vapaaehtoinen runko-osa. Yleensä tämä on pyynnössä, kun käytetään POST-metodia. Host-otsikko määrittää sivuston verkkotunnuksen, joka mahdollistaa monen sivuston ylläpitämisen samasta IP-osoitteesta. (RFC 2616 1999, 129.)

6.3 HTTP-vastaus

HTTP-vastaus koostuu tilarivistä, otsikoista ja runko-osasta. Tilarivillä on protokolla versio ja numerollinen ja kirjallinen vastauskoodi. Vastauksen rivit erotetaan CRLF-merkillä. (KUVIO 5.) Taulukossa 4 on tämän työn kannalta oleellisimmat statuskoodit.

```
HTTP/1.1 200 OK [CRLF]
Content-Type: text/plain; charset=utf-8 [CRLF]
Date: Mon, 17 Oct 2016 11:48:02 GMT [CRLF]
Server: Microsoft-IIS/7.5 [CRLF]
Content-Length: 70 [CRLF]
Connection: close [CRLF]
[CRLF]
<!DOCTYPE html>
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

KUVIO 5. HTTP-vastaus

TAULUKKO 4. HTTP-statuskoodit (mukaillen RFC 2616 1999, 57-69)

Statuskoodi	Selitys
200 OK	Pyyntö käsitelty onnistuneesti.
400 Bad Request	Pyyntöä ei voitu käsitellä viallisen syntaksin takia.
500 Internal Server Error	Pyyntöä ei voitu käsitellä odottamattoman tilanteen takia.
503 Service Unavailable	Pyyntöä ei voitu käsitellä ylikuormituksen tai huoltokatkoksen vuoksi.

7 TESTATUT PALVELUNESTOHYÖKKÄYKSET

Valitsin testattaviksi hyökkäyksiksi HTTP-protokollaan perustuvat Keep-Dead-, hitaat ja tulvitushyökkäykset. Valitsin nämä koska ne ovat yleisimpiä HTTP-protokollan palvelunestohyökkäyksiä. Näille hyökkäyksille on olemassa työkaluja, joita käytetään muun muassa sietokyvyn testaamiseen.

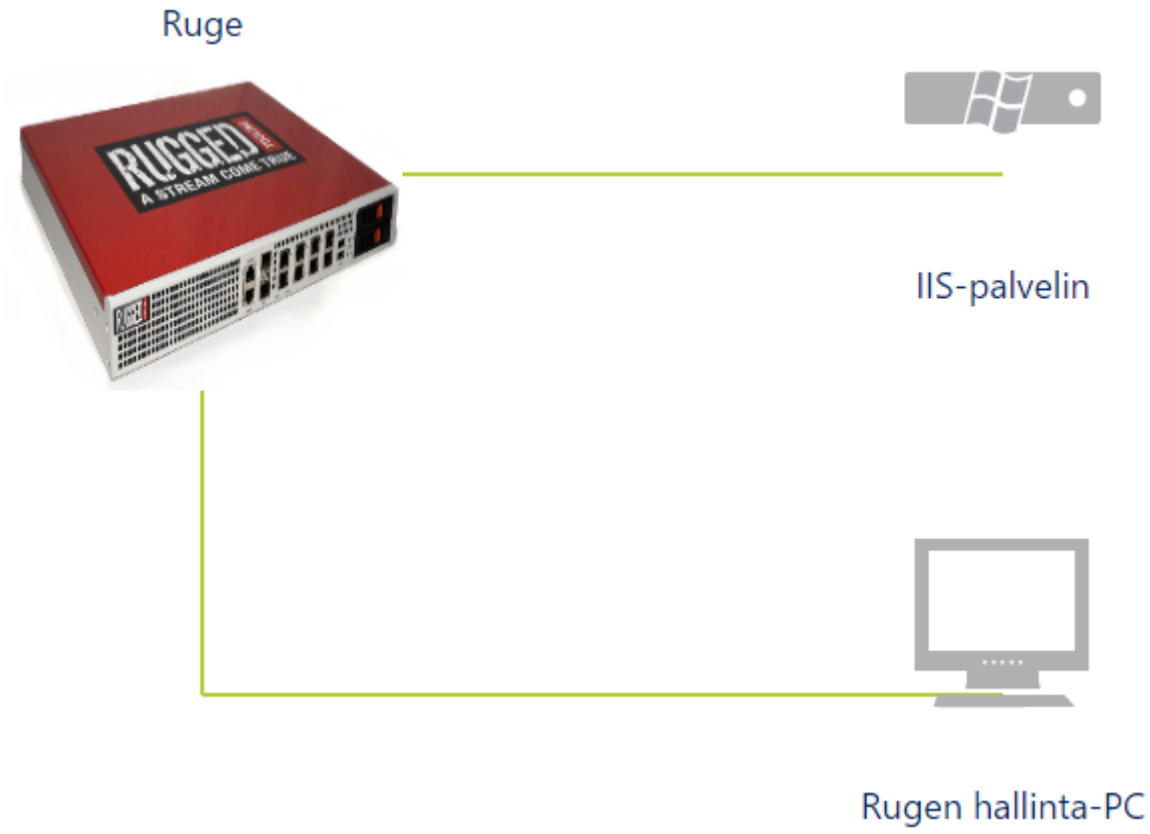
Tulvitushyökkäyksessä on tavoitteena lähettää mahdollisimman monta pyyntöä palvelinohjelmalle kuluttaen sen resurssit, joka johtaa palvelun hitauteen tai toimimattomuuteen. Tulvitushyökkäys voidaan toteuttaa lähettämällä pyyntöjä palvelimelle odottamatta palvelimelta vastausta. (Dantas, Nigam & Fonseca 2014, 77.) HTTP-tulvitushyökkäyksiä voidaan suorittaa Low Orbit Ion Canon (LOIC) -työkalulla. Tätä työkalua on käyttänyt Anonymous-kyberrikollisryhmä palvelunestohyökkäyksen toteuttamiseen. Tähän kategoriaan kuuluu myös Extensible Markup Language (XML) -tulvitushyökkäys, joka käyttää Simple Object Access Protocol (SOAP) -protokollaa HTTP:n päällä. (Osanaiye, Choo, & Dlodlo 2015, 150.) Koska tämä käyttää SOAP-protokollaa tätä ei tässä työssä testata.

Keep-Dead-hyökkäyksessä lähetään HEAD-metodilla pyyntöjä palvelimelle. Hyökkäyksessä lähetetään monta pyyntöä saman yhteyden sisällä. Tämä on mahdollista, jos palvelimessa on ”Keep-Alive”-toiminto päällä. HEAD-metodia käytetään, koska silloin palvelin lähettää vastauksessa ainoastaan otsikot, joten hyökkääjän kaistankäyttö on pienempi kuin esim. GET-metodia käytettäessä. Hyökkäyksessä myös muutetaan yhden URI-parametrin arvoa, joka estää tiedon hakemisen suoraan välimuistista. (Keep-Alive DoS Script 2011.) Tämä hyökkäys voidaan periaatteessa luokitella tulvitushyökkäyksien alle, koska se poikkeaa ainoastaan perinteisestä tulvitushyökkäyksestä, koska se käyttää HEAD-metodia ja on suunniteltu ohittamaan välimuisti muuttamalla URI-parametrin arvoa.

Hitaassa hyökkäyksessä tavoitteena on pitää yhteys auki mahdollisimman kauan. Hitaat hyökkäykset käyttävät muun muassa GET- ja POST-metodeja. (Shekyan 2016.) GET-metodilla ei lähetetä otsikoiden jälkeen viimeistä rivinvaihtoa. Tällä tavalla palvelin saadaan odottamaan lisää dataa. Dataa voidaan lähettää esim. 30 sekunnin välein, jolloin palvelin pitää yhteyttä auki. Tästä hyökkäyksestä käytetään nimiä Slowloris ja Slow-Header. (Pillai 2013.) POST-metodilla runko-osa jaetaan osiin. Runko-osan koko asetetaan esim. 50-tavuksi ja sen sisältö lähetetään osissa esim. 30 sekunnin välein. Tätä kutsutaan Slow POST -hyökkäykseksi. (Dantas, ym. 77.) Hitaita hyökkäyksiä voidaan suorittaa SlowHTTPTest-työkalulla (Shekyan 2016.).

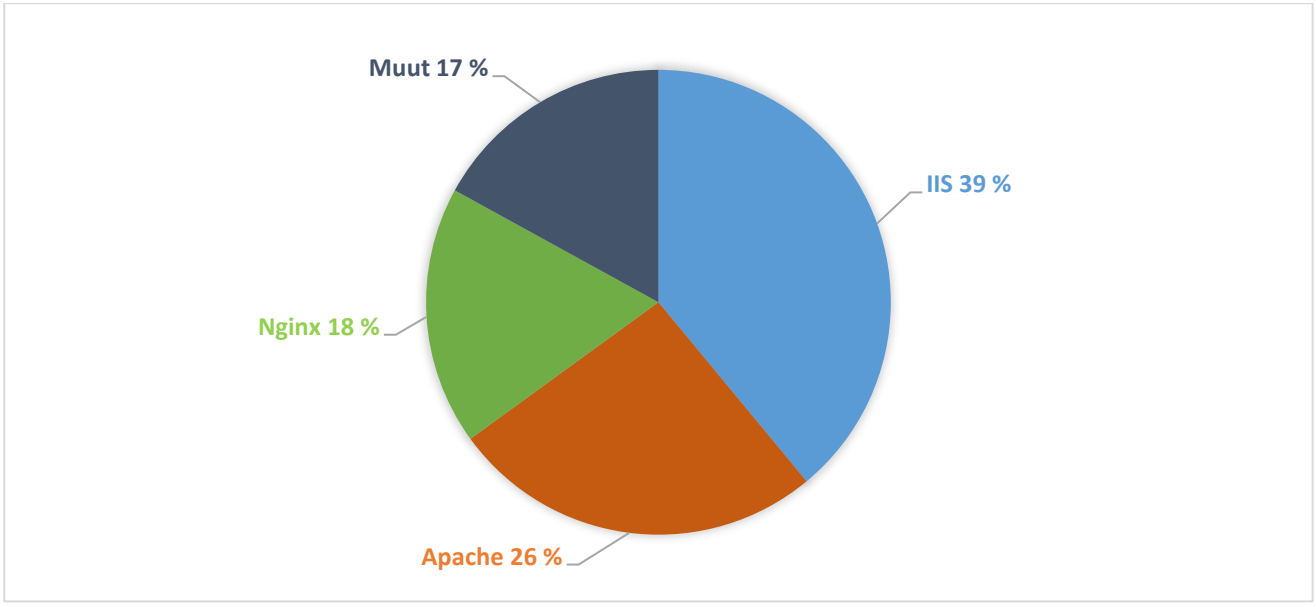
8 TESTAUSYMPÄRISTÖ

Hyökkäyksien toteuttamiseen käytetiin Rugea, joka on Rugged Tooling yrityksen kehittämä työkalu verkon kapasiteetin testaamiseen. Tämä työkalu tukee lukuisia eri siirto- ja sovelluskerroksen protokollia. Rugelle rakennetaan hyökkäykset, käyttäen sen ohjelmistoa. Hajautetun hyökkäyksen simulointi onnistuu myös, eli voidaan simuloida hyökkäyksen tulemista monesta eri paikasta muuttamalla IPv4-otsikon lähde IP-osoitetta.



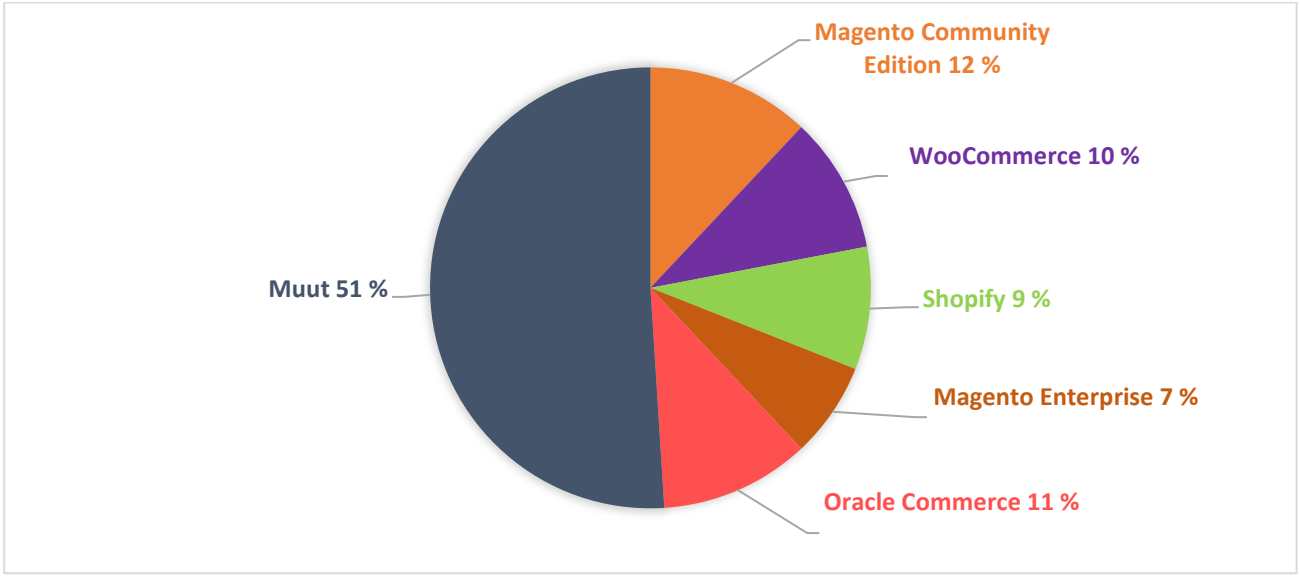
KUVIO 6. Verkkokartta

Ruge on liitetty palvelimeen suoraan. Hallinta-PC on kytketty Rugen ohjausporttiin, josta voidaan antaa Rugelle käskyjä. Rugen ensimmäiseen porttiin on kytketty IIS-palvelin. (KUVIO 6.) Palvelin käyttää IIS 8.0 (Internet Information Services) palvelinohjelmistoa, joka on Microsoftin kehittämä. Palvelimella on Windows 7 64-bit -käyttöjärjestelmä, Intel i5-2400 3,10 GHz -prosessori ja 8 GB -keskusmuistia. IIS on yksi suosituimmista palvelinohjelmistoista Apachen ja Nginx:n kanssa (KUVIO 7). Verkkokauppa asennetaan tämän päälle.



KUVIO 7. Kaikkien sivustojen web-palvelinohjelmien markkinaosuus (mukaillen September 2016 Web Server Survey)

Verkkokaupaksi valitsin Magenton. Valitsin tämän, koska se on yksi suosituimmista verkkokauppa-alustoista (KUVIO 8). Käytin Magenton ilmaista Community Edition -versiota. IIS-sovellusvalikoimasta löytyy magento-asennuspaketti, joka asentaa ja määrittää tarvittavat ohjelmistot palvelimelle automaattisesti, kuten PHP (Hypertext Preprocessor) -tulkin ja tietokannan.



KUVIO 8. 10 000 suosituimman verkkokauppasivustojen verkkokauppa-alustojen markkinaosuus (mukaillen Ecommerce Usage Statistics)

Wireshark- ja Fiddler-ohjelmia voidaan käyttää apuna hyökkäyksien laatimisessa. Wireshark-ohjelmalla voidaan seurata pakettien kulkua. Tällä tavalla saadaan selville missä vika on, jos ei voida muodostaa yhteyttä tai jos palvelin ei vastaanota HTTP-pyyntöä. Fiddler-ohjelmalla voidaan seurata selaimen ja verkkosivun välistä HTTP-liikennettä.

Tein työkalun, joka seuraa palvelimen vastausaikaa lukemalla palvelimen lokitiedoston, Central Processing Unit (CPU) -kuormitusta ja keskusmuistin käyttöä. Microsoft tarjoaa työkalun näiden lokitiedostojen lukemiseen, koska halusin hallita paremmin, millaiset kaaviot piirrän, tein oman työkalun. Tein tämän Javascript-ohjelmointikielellä, käyttäen Node.Js-ympäristöä ja AngularJS-kehystä. Käytin JavaScriptia, koska minulla oli jo aikaisempaa kokemusta kaavioiden piirrosta tällä ohjelmointikielellä.

```
// 1 second interval
$scope.dataPromise = $interval(() => {
    os.cpuUsage((usage) => {
        usage *= 100; // 0 - 1 to 0 - 100

        const millis = new Date().getTime() - $scope.startTime;
        $scope.runtime = formatMillis(millis); // millis to mm:ss
        $scope.cpuData.push({y: usage, x: millis});

        // about 30 seconds avg
        if ($scope.cpuLoads.length == 30) {
            $scope.cpuLoads.shift(); // remove first element
        }
        $scope.cpuLoads.push(usage);
        $scope.cpuDataAvg.push({y: $scope.cpuLoads.avg(), x: millis});

        const usedMem = 100 - (os.freememPercentage() * 100);
        $scope.memData.push({y: usedMem, x: millis});
    });
}, 1000);
```

KUVIO 9. Resurssien käytön seurannan lähdekoodi

Kuviossa 9 lasketaan sekunnin välein prosessorin kuormitus, sen keskiarvo ja käytetty muisti. Keskiarvo koostuu edellisestä 30 sekunnista.

```
function parseIISLog(filePath, callback) {
    fs.readFile(filePath, 'utf8', (err, data) => {
        if (err)
            throw err;

        data = data.split('\r\n'); // split by row
        data.splice(0, 4); // remove header rows

        let requests = [];

        for (const row of data) {
            // skip if row is empty
            if (row.length == 0)
                continue;

            const fields = row.split(' '); // fields are seperated by space

            // fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username
            // c-ip cs(User-Agent) cs(Referer) sc-status sc-substatus sc-win32-status time-taken
            requests.push({
                date: new Date(fields[0] + 'T' + fields[1]),
                ip: fields[2],
                method: fields[3],
                uri: fields[4],
                port: parseInt(fields[6]),
                statusCode: parseInt(fields[fields.length - 4]),
                timeTaken: parseInt(fields[fields.length - 1])
            })
        }
        callback(requests);
    });
}
```

KUVIO 10. Lokitiedoston lukemisen lähdekoodi

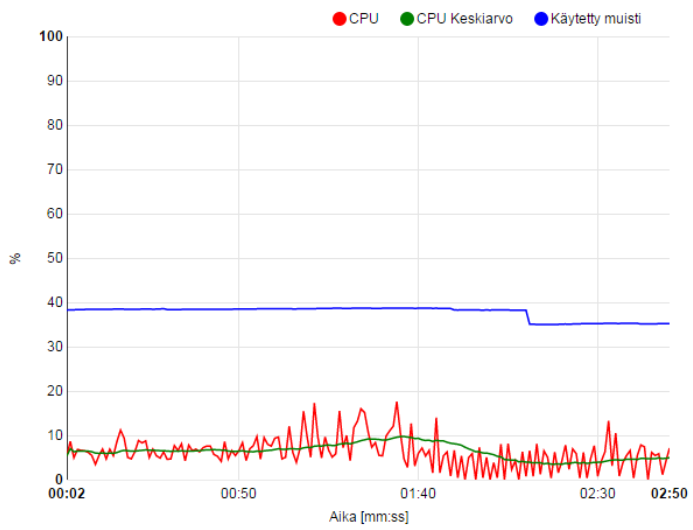
IIS-palvelin tallentaa kiintolevylle lokitiedoston, johon on kirjattu tietoja jokaisesta HTTP-pyynnöstä. Kuviossa 10 luetaan tästä lokitiedostosta jokaisen pyynnön ajankohta, IP-osoite, portti, metodi, URI, statuskoodi ja vastausaika. Kuviossa 11 lasketaan tästä datasta vastausajat ja virheprosentti. Myös jokaisen eri statuskoodin määrä otetaan talteen, josta tehdään ympyräkaavio.

```
parseIISLog($scope.iisFilePath, (requests) => {
    if (requests.length == 0)
        return;

    let timeSum = 0, respAvgSum = 0, respCount = 0, respMin = 0, respMax = 0;
    let srvErrTotalCount = 0, srvErrCount = 0;

    let statusCodeCounts = {};

    const startTime = requests[0].date.getTime();
    let prevTime = startTime;

    for (const req of requests) {
        const time = req.date.getTime();

        if (!statusCodeCounts.hasOwnProperty(req.statusCode)) {
            statusCodeCounts[req.statusCode] = 0;
        }
        statusCodeCounts[req.statusCode]++;

        // calculate chart data
        if (prevTime != time) {
            const deltaTime = prevTime - startTime;

            $scope.netErrData.push({y: (srvErrCount / (respCount + srvErrCount)) * 100, x: deltaTime});

            if (respCount > 0) {
                $scope.netData.push({y: Math.ceil(respAvgSum / respCount), x: deltaTime});
                $scope.netDataMin.push({y: respMin, x: deltaTime});
                $scope.netDataMax.push({y: respMax, x: deltaTime});
            }

            respAvgSum = 0;
            respCount = 0;
            respMin = 0;
            respMax = 0;
            srvErrCount = 0;
        }

        if (Math.floor(req.statusCode / 100) == 5) {
            srvErrTotalCount++;
            srvErrCount++;
            prevTime = time;
            continue;
        }

        if (!respMin || respMin > req.timeTaken) {
            respMin = req.timeTaken;
        }

        if (!respMax || respMax < req.timeTaken) {
            respMax = req.timeTaken;
        }

        prevTime = time;
        respAvgSum += req.timeTaken;
        respCount++;
        timeSum += req.timeTaken;
    }
```

KUVIO 11. Lokitiedoston datan käsittelyn lähdekoodi

9 TESTAUKSET

Jotta hyökkäyksien vaikutuksia palvelimen toimintaan voidaan tarkastella, vertailukohtana käytetään tilannetta, jolloin verkkosivulla ei ole käyttäjiä. Palvelimen vastausajat otetaan lokitiedostosta, jonka palvelin tallentaa kiintolevylle. Palvelimen resurssien käytön lisäksi mitataan myös, kuinka kauan HTTP-pyyntöjen käsittelyssä kesti eli vastausaikaa, millä statuskoodilla palvelin vastasi ja hyökkäystyökalun kaistankäyttöä. Vastausajasta ja prosessorin kuormituksesta seuraamalla saadaan selville, kuinka kiireellinen palvelin on. Keskusmuistia seuraamalla saadaan selville hyökkäyksen vaikutus keskusmuistin käyttöön ja mahdolliset viat ohjelmistossa, jotka aiheuttavat keskusmuistin käytön nousun. Statuskoodeja seuraamalla saadaan selville, kuinka monta prosenttia palvelin pystyi käsittelemään onnistuneesti pyynnöistä. Kaistankäyttöä seuraamalla saadaan selville, kuinka hyökkäyksen toteuttamiseen tarvitaan kaistaa.



KUVIO 12. Palvelimen resurssien käyttö toimettomana

Prosessorin kuormitus on 0 – 18 % ja keskusmuistin käyttöprosentti on 35 - 38 % (KUVIO 12). Palvelimen vastausaika etusivua haettaessa on noin 1 s.

Kuviossa 13 on hyökkäyksen rakenne. Ensimmäiseksi luodaan TCP-yhteys (sarakkeet 2-5). Samalla myös alustetaan HTTP-laskuri nollaksi. Tämän jälkeen ohjelma lähettää pyyntöjä palvelimelle ja jokaisella kerralla kasvatetaan laskuria. Kun laskuri on yli asetetun raja-arvon (LIMIT), palataan takaisin

alkuun ja yhteys muodostetaan uudelleen käyttäen eri porttia. ”Timeout” tarkoittaa, että ohjelma odottaa tietyn ajan, joka on määritetty tälle muuttujalle. Tämä siis määrittää kuinka usein pyyntö lähetään saman yhteyden sisällä. Käytän tästä termiä ”lähetystiheys”. Hyökkäyksen rakenne on tämän mukainen, jos sitä ei ole erikseen mainittu.

	1	2	3	4	5	6	7	8	9	10
OFFSET [us]		1								
ITEM	START	TCP/SYN	INIT_HTTP_CTR	TCP/SYN_ACK						
OFFSET [us]					1					
ITEM				SYN/ACK	TCP/ACK	LOOP				
OFFSET [us]										
ITEM						Timeout	HTTP_CTR<LIMIT			
OFFSET [us]								1		
ITEM							TRUE	HTTP_REQ	INC_HTTP_CTR	LOOP
OFFSET [us]								1		
ITEM							FALSE	TCP/CLOSE	START	

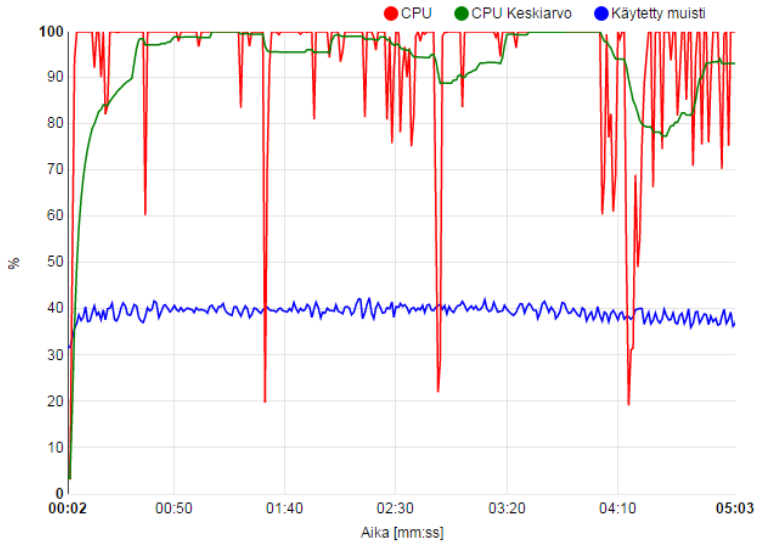
KUVIO 13. Hyökkäyksen rakenne

9.1 Keep-Dead

Hyökkäys tehdään tuotehaku päätepisteeseen. Oletuksena ”Keep-Alive”-toiminto on päälle, joten tätä otsikkoa ei tarvitse olla pyynnössä. Tuotteita haetaan hintavälillä 0 – x, missä x:n arvo on eri joka pyynnössä. (KUVIO 14.) Hyökkäyksen aikana prosessorin kuormitus on lähes koko ajan 100 % ja muistinkäyttö pysyy samana hyökkäyksen ajan. Palvelimen resurssien käyttö on tämän mukainen lähes kaikissa Keep-Dead-testeissä. (KUVIO 15).

```
HEAD http://magento.lan:11104/index.php/catalogsearch/advanced/result/?price[from]=0&price[to]=1234 HTTP/1.1
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: magento.lan:11104
```

KUVIO 14. Keep-Dead HTTP -pyyntö

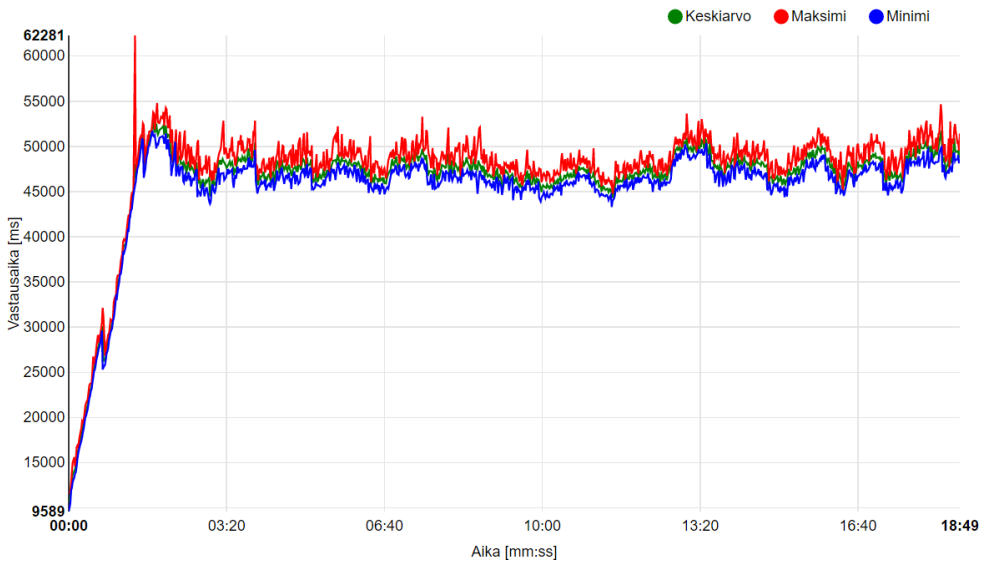


KUVIO 15. Keep-Dead-hyökkäyksen resurssien käyttö

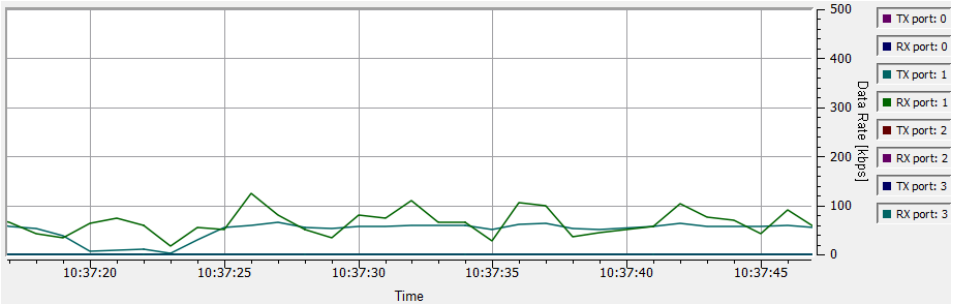
9.1.1 500 yhteyttä

30 sekunnin lähetystiheydellä vastausaika on 45 – 60 sekuntia (KUVIO 16). Ruge lähettää ja vastaanottaa tällöin noin 50 kbps (KUVIO 17).

Hyökkäyksen aikana prosessorin kuormitus on lähes koko ajan 100 % ja muistinkäyttö pysyy samana hyökkäyksen ajan. Palvelimen resurssien käyttö on tämän mukainen lähes kaikissa Keep-Dead-testeissä. (KUVIO 15).

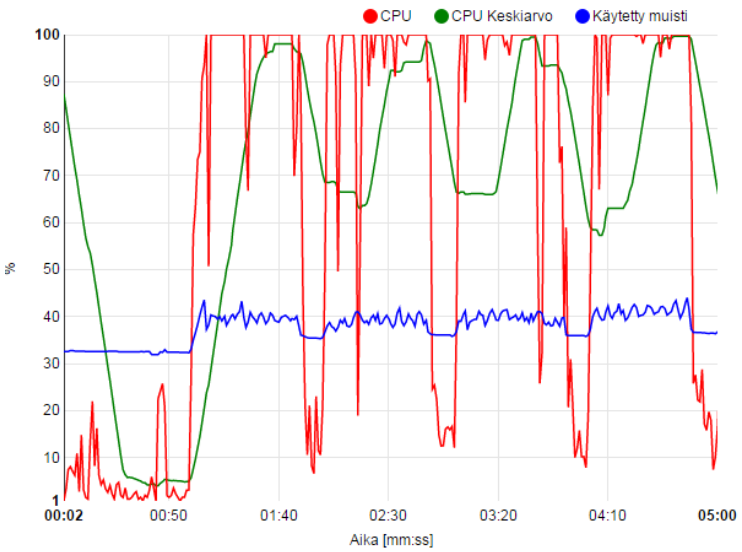


KUVIO 16. Keep-Dead-hyökkäyksen vastausaika 500 yhteydellä ja 30 s lähetystiheydellä

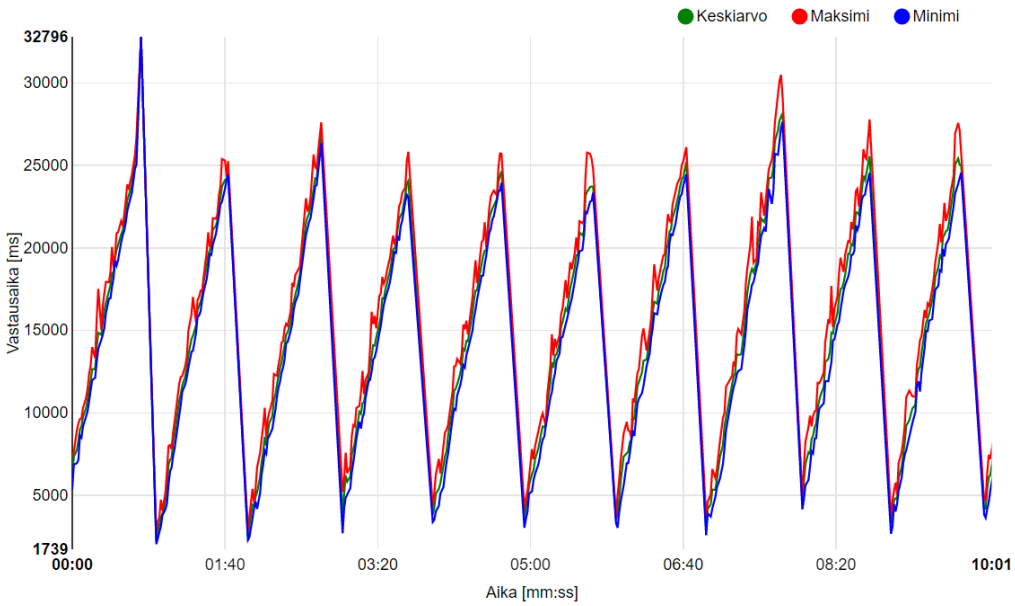


KUVIO 17. Keep-Dead-hyökkäyksen kaistankäyttö 500 yhteydellä ja 30 s lähetystiheydellä

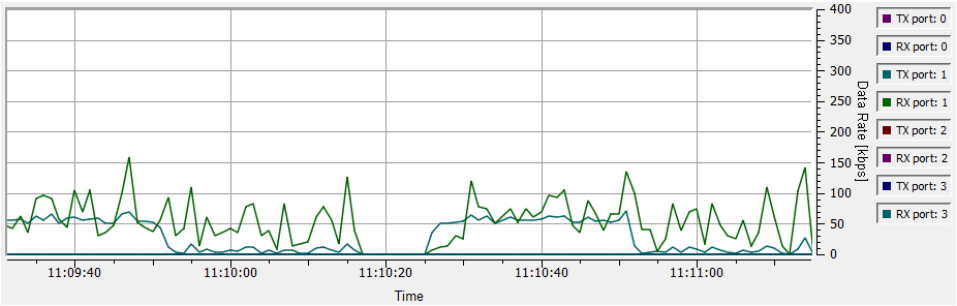
Vastausaika laskee noin 50 sekunnin välein (KUVIO 19) ja samalla myös resurssienkäyttö pienenee (KUVIO 18). Tästä voidaan päätellä, että hyökkääjän pitää lähettää vähintään 50 sekunnin välein pyyntöjä, jotta kuormitus pysyy yllä. Ruge lähettää 0 – 50 kbps ja vastaanottaa 0 – 150 kbps (KUVIO 20).



KUVIO 18. Keep-Dead-hyökkäyksen resurssien käyttö 500 yhteydellä ja 60 s lähetystiheydellä



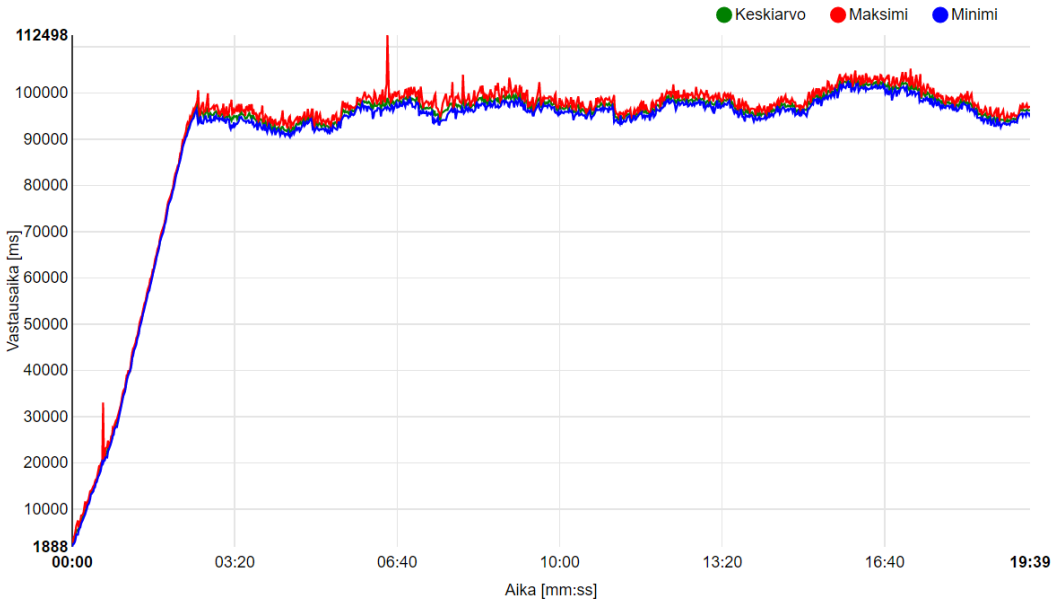
KUVIO 19. Keep-Dead-hyökkäyksen vastausaika 500 yhteydellä ja 60 s lähetystiheydellä



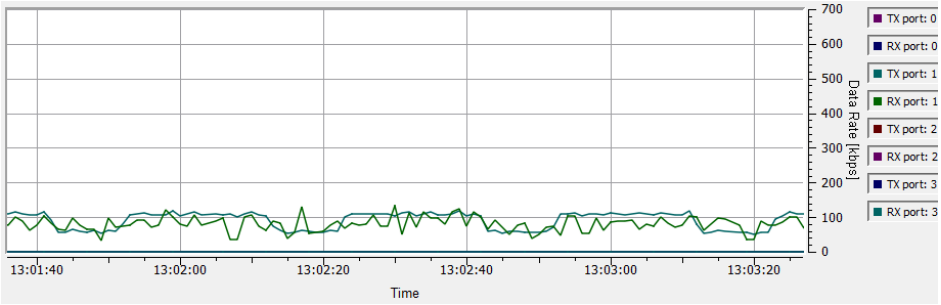
KUVIO 20. Keep-Dead-hyökkäyksen kaistankäyttö 500 yhteydellä ja 60 s lähetystiheydellä

9.1.2 1 000 yhteyttä

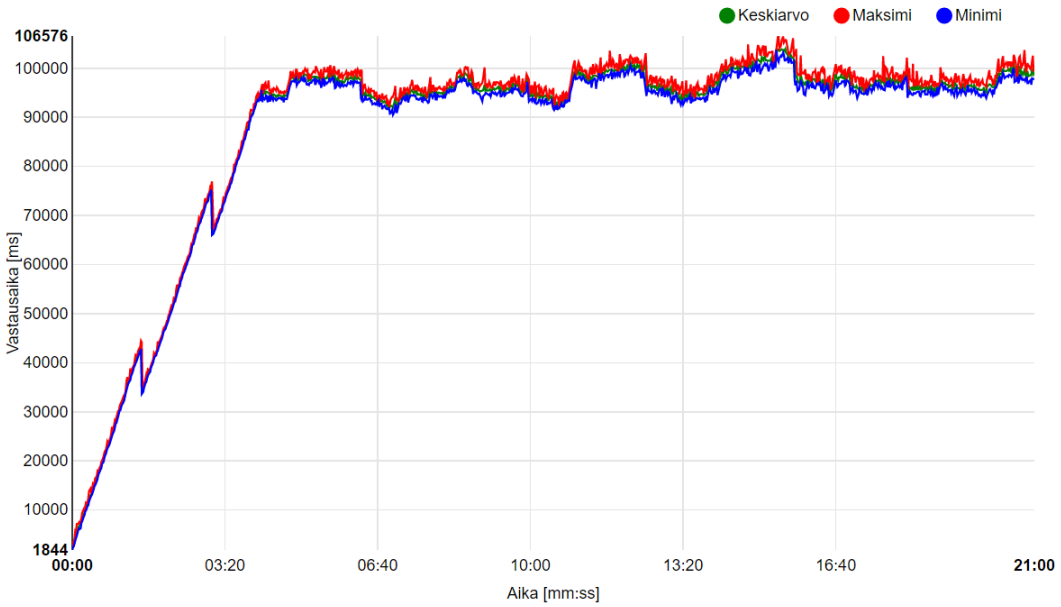
30 sekunnin lähetystiheydellä vastausaika vaihtelee 90 ja 104 sekunnin välillä (KUVIO 21). Vastausaika on lähes sama myös 60 sekunnin lähetystiheydellä (KUVIO 23). Rugen kaistan käyttö on noin 50 – 100 kbs 30 sekunnin lähetystiheydellä (KUVIO 22). 60 sekunnin lähetystiheydellä se on hieman pienempi (KUVIO 24).



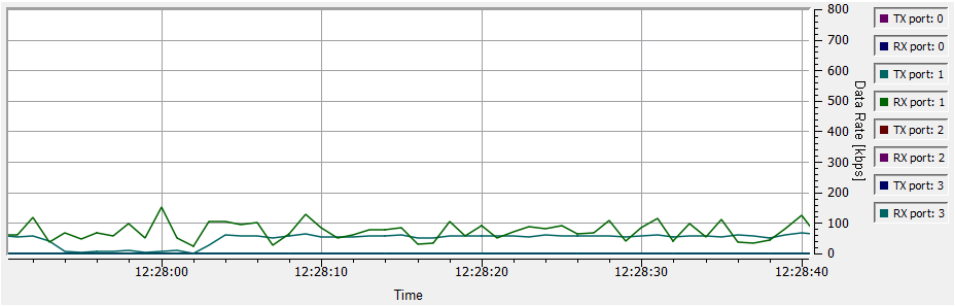
KUVIO 21. Keep-Dead-hyökkäyksen vastausaika 1 000 yhteydellä ja 30 s lähetystiheydellä



KUVIO 22. Keep-Dead-hyökkäyksen kaistankäyttö 1 000 yhteydellä ja 30 s lähetystiheydellä



KUVIO 23. Keep-Dead-hyökkäyksen vastausaika 1 000 yhteydellä ja 60 s lähetystiheydellä



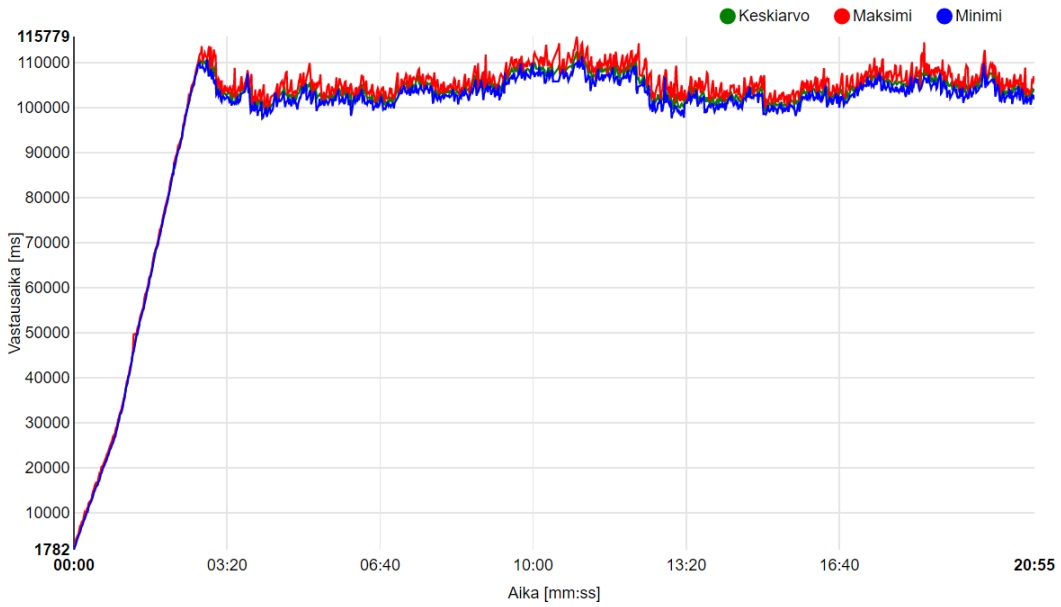
KUVIO 24. Keep-Dead-hyökkäyksen kaistankäyttö 1 000 yhteydellä ja 60 s lähetystiheydellä

9.1.3 2 000 yhteyttä

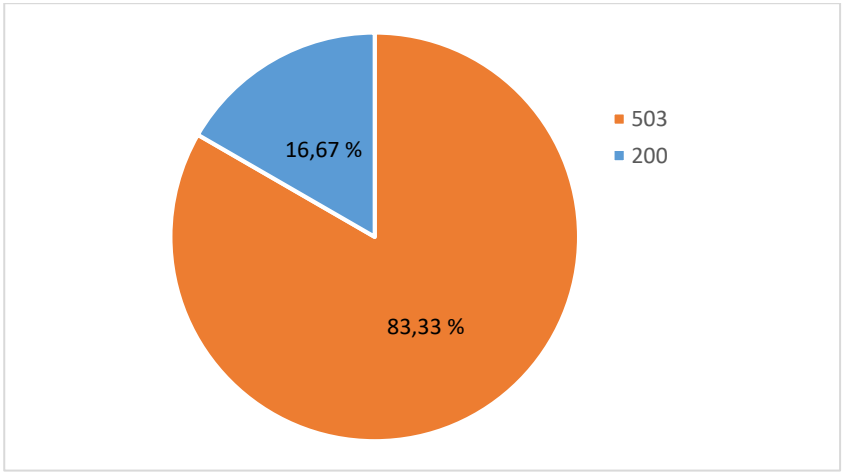
Yli 1 000 yhteydellä osalle yhteyksistä tulee ”HTTP Error 503.4 - Service Unavailable. The FastCGI queue has been exceeded” -virheilmoitus, eli palvelin ei pysty käsittelemään pyyntöä. Tämä johtuu siitä, että oletuksena tämä jonon pituus on 1 000. Tässä osiossa testataan, kuinka suurelle määrälle pyynnöistä tämä tulee.

Palvelimen vastausaika on 100 – 120 sekuntia 30 sekunnin lähetystiheydellä (KUVIO 25). Palvelin pystyy käsittelemään 16,67 % pyynnöistä 30 sekunnin lähetystiheydellä (KUVIO 26). Palvelin käsittelee 34,63 % pyynnöistä 60 sekunnin lähetystiheydellä (KUVIO 29). Tämä on 17,96 % enemmän kuin 30

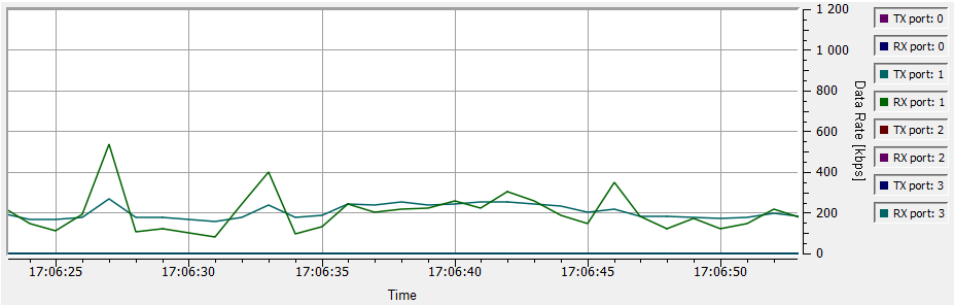
sekunnin lähetystiheydellä. Vastausaika on hieman matalampi 60 sekunnin lähetystiheydellä (KUVIO 28). Maksimi lähetyskaistan käyttö on noin 220 kbps 30 sekunnin ja 120 kbps 60 sekunnin lähetystiheydellä. Maksimi vastaanottokaistan käyttö on noin 550 kbps 30 sekunnin ja 220 kbps 60 sekunnin lähetystiheydellä. (KUVIO 27 ja KUVIO 30.)



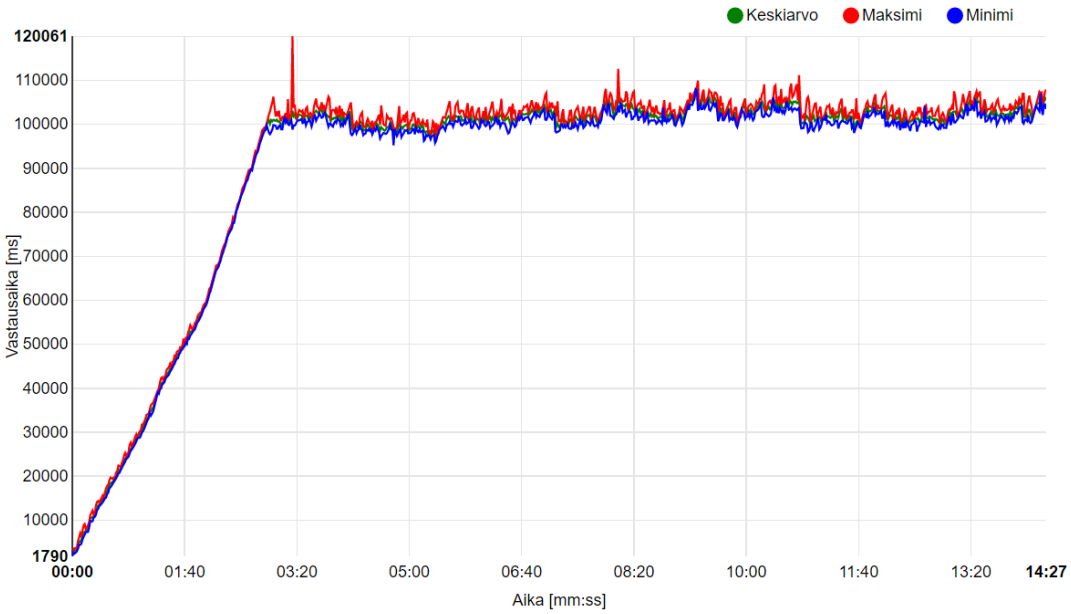
KUVIO 25. Keep-Dead-hyökkäyksen vastausaika 2 000 yhteydellä ja 30 s lähetystiheydellä



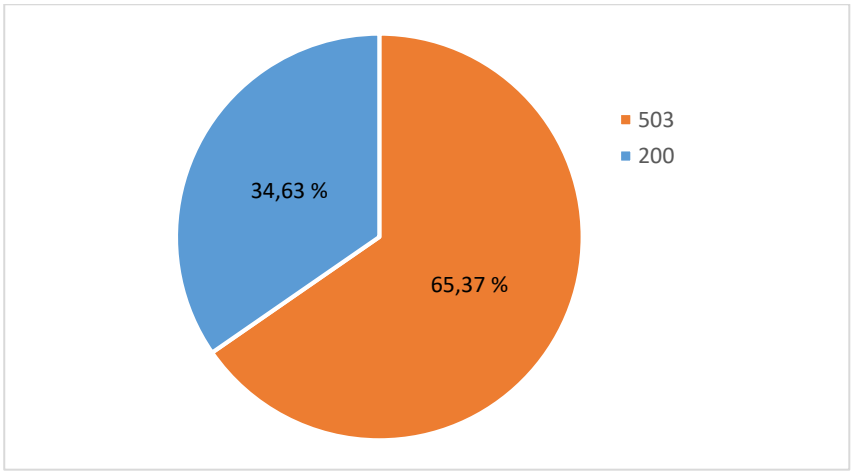
KUVIO 26. Keep-Dead-hyökkäyksen statuskoodit 2 000 yhteydellä ja 30 s lähetystiheydellä



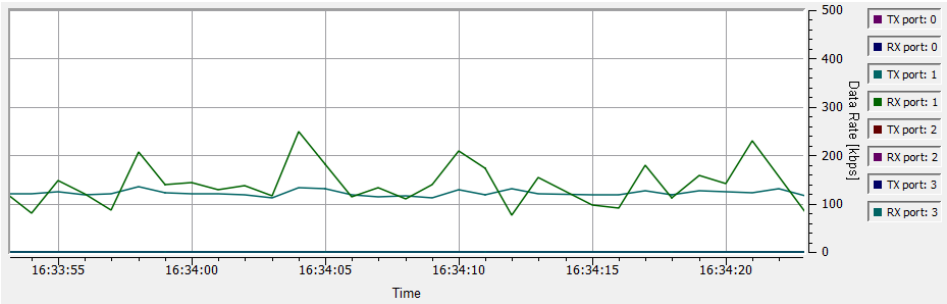
KUVIO 27. Keep-Dead-hyökkäyksen kaistankäyttö 2 000 yhteydellä ja 30 s lähetystiheydellä



KUVIO 28. Keep-Dead-hyökkäyksen vastausaika 2 000 yhteydellä ja 60 s lähetystiheydellä



KUVIO 29. Keep-Dead-hyökkäyksen statuskoodit 2 000 yhteydellä ja 60 s lähetystiheydellä



KUVIO 30. Keep-Dead-hyökkäyksen kaistankäyttö 2 000 yhteydellä ja 60 s lähetystiheydellä

9.2 GET-tulva

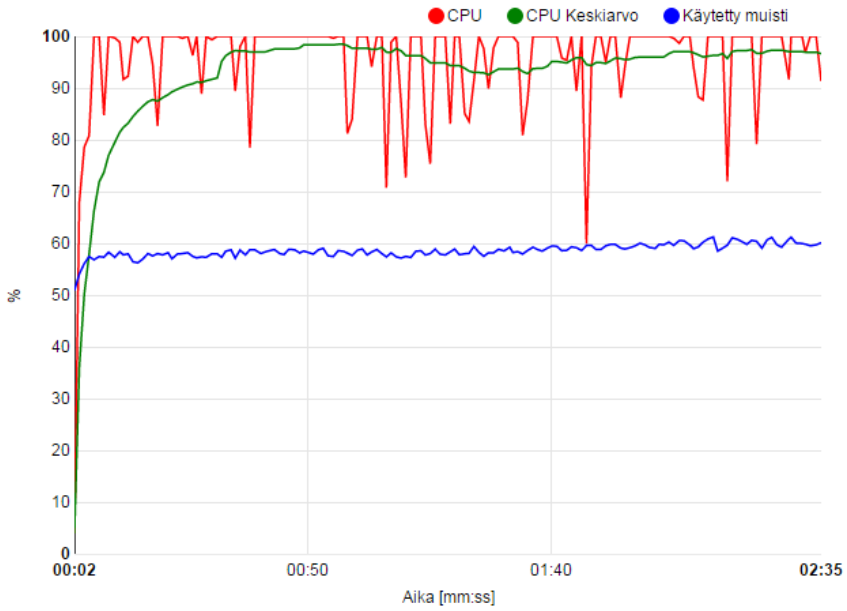
Lähetystiheytenä käytetään 400 millisekuntia ja yhteyksien määrä on 1 000. Hyökkäys suoritetaan dynaamiseen ja staattiseen sivuun.

9.2.1 Dynaaminen sivu

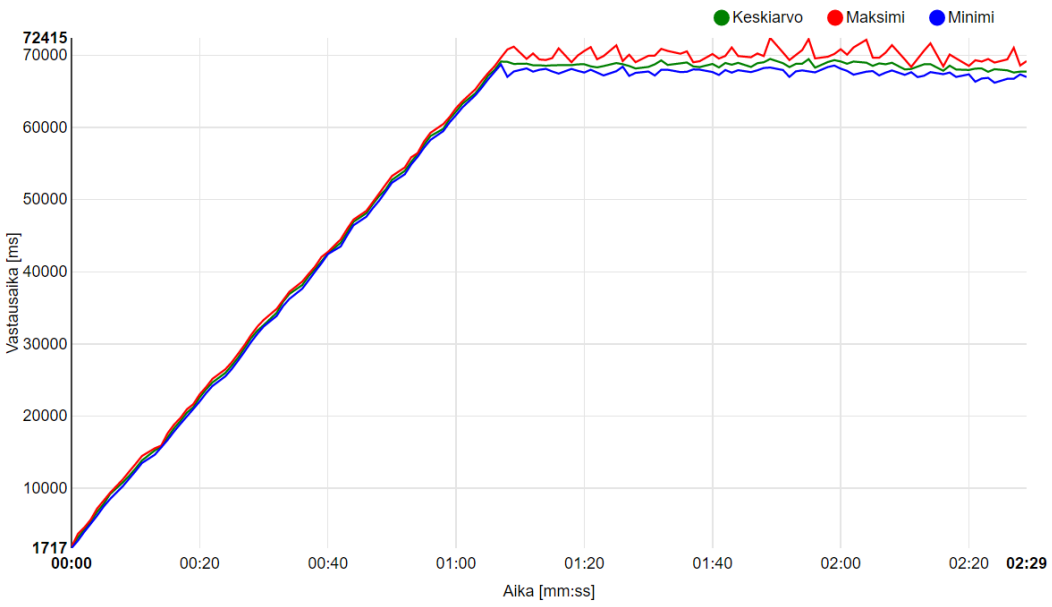
Dynaamisena sivustona käytetään etusivua ja pyyntö on kuvion 31 mukainen. Hyökkäyksen aikana CPU:n kuormitus on lähes koko ajan 100 % ja keskusmuistin käyttö nousee noin 8 % (KUVIO 32). Vastausaika on noin 70 sekuntia (KUVIO 33). Ruge lähettää noin 5 – 7 Mbps ja vastaanottaa 3 – 5 Mbps (KUVIO 34).

```
GET http://magento.lan:11104 HTTP/1.1
Accept: */*
Accept-Language: fi-FI
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: magento.lan:11104
```

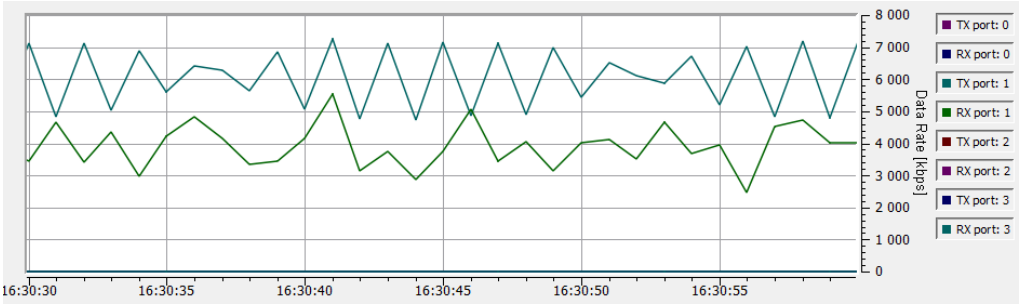
KUVIO 31. Etusivun HTTP-pyyntö



KUVIO 32. Resurssien käyttö dynaamisella sivulla



KUVIO 33. Vastausaika dynaamisella sivulla



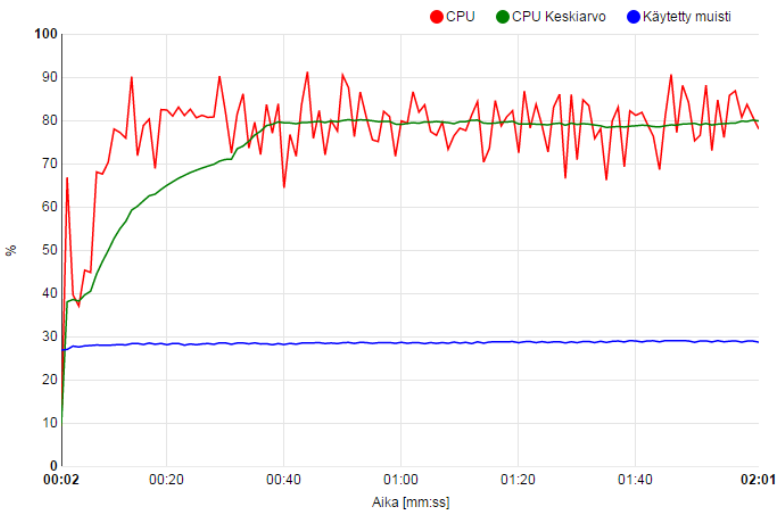
KUVIO 34. Kaistankäyttö dynaamisella sivulla

9.2.2 Staattinen sivu

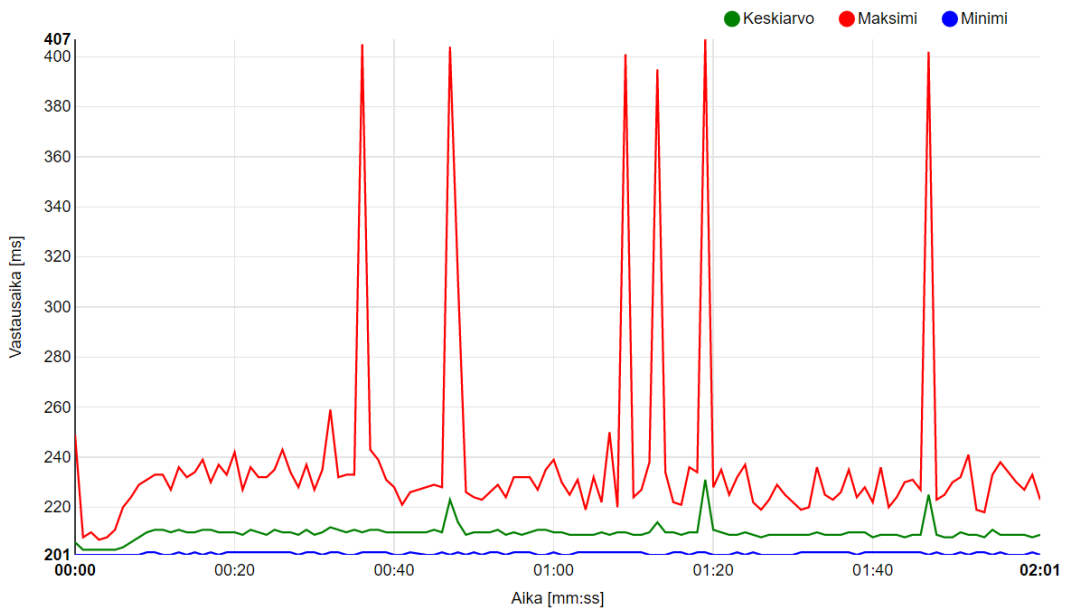
Staattisena sivuna käytetään Magento-logoa. Kuviossa 35 on tämän pyyntö. Hyökkäyksen aikana CPU:n kuormitus nousee noin 80 %. Keskusmuistin käyttöön hyökkäys ei vaikuta paljon. (KUVIO 36.) Suurimpaan osaan pyynnöistä palvelin vastaa noin 220 millisekunnin päästä. Maksimivastausaika on 407 ms. (KUVIO 37.) Ruge lähettää 12 Mbps ja vastaanottaa 58 Mbps (KUVIO 38).

```
GET http://magento.lan:11104/skin/frontend/rwd/default/images/logo.gif HTTP/1.1
Accept: */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: magento.lan:11104
```

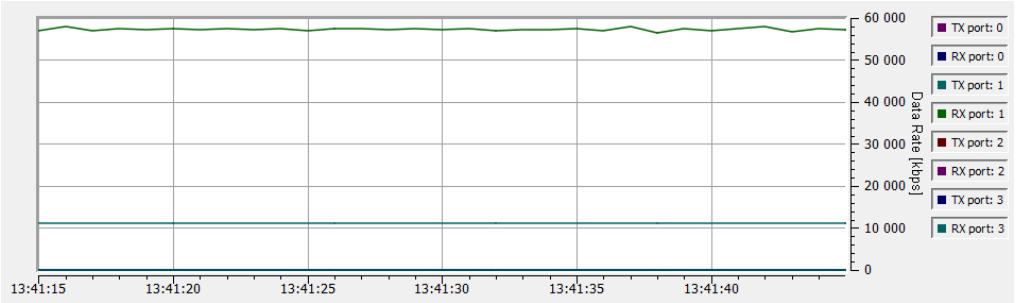
KUVIO 35. Staattisen sivun HTTP-pyyntö



KUVIO 36. Resurssien käyttö staattisella sivulla



KUVIO 37. Vastausaika staattisella sivulla



KUVIO 38. Kaistankäyttö staattisella sivulla

9.3 POST-tulva

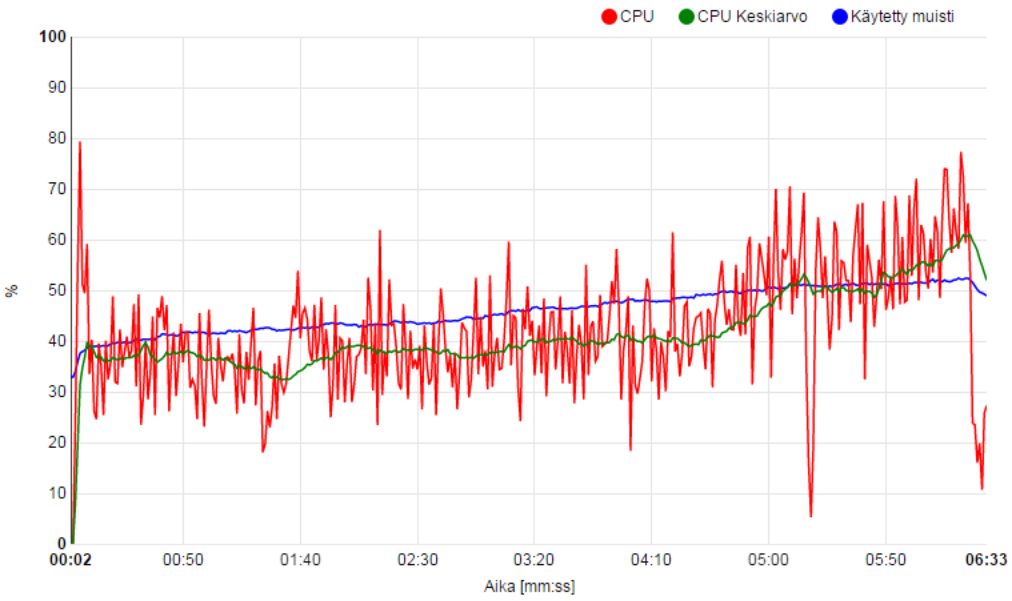
Sivuston ylläpitäjä voi luoda gallupeja. Tässä osiossa testataan miten palvelunestohyökkäys vaikuttaa tähän. ”Frontend”-eväste täytyy olla pyynnössä, muuten palvelin ei hyväksy sitä. Samalla evästeellä voidaan kuitenkin äänestää monta kertaa, mikä helpottaa testin suorittamista. ”Content-Length”-otsikon arvo täytyy olla sama kuin rungon koko, tässä tapauksessa 6 tavua. (KUVIO 39.) Jos pyynnön käsittely onnistuu palvelin vastaa 302 statuskoodilla.

```
POST http://magento.lan:11104/index.php/poll/vote/add/poll_id/1/ HTTP/1.1
Accept: */*
Accept-Language: fi-FI
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Content-Length: 6
Cookie: frontend=q2ros6d9c739ogqbbq5u85sor6
Accept-Encoding: gzip, deflate
Host: magento.lan:11104

vote=1
```

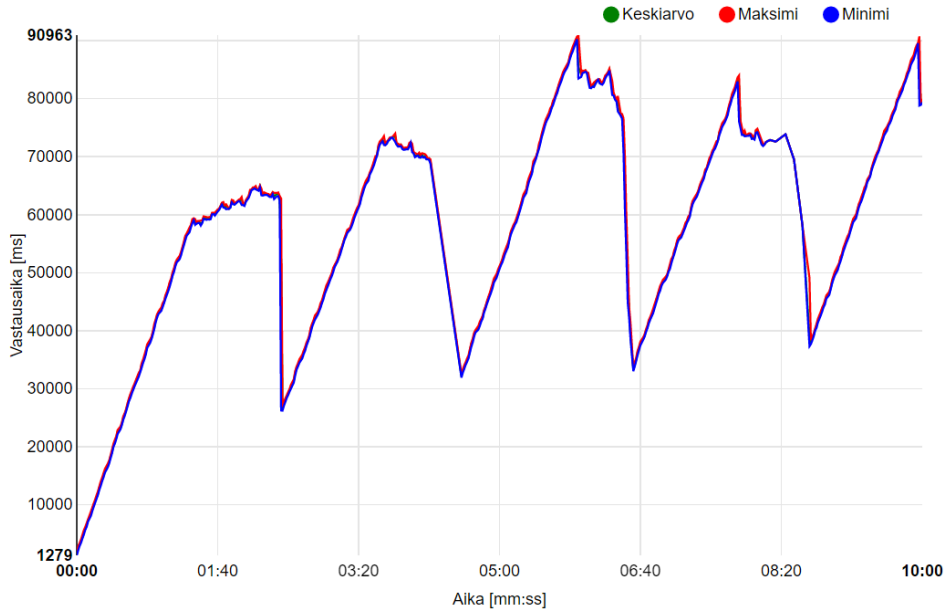
KUVIO 39. HTTP-pyyntö gallup päätepisteeseen

9.3.1 500 yhteyttä

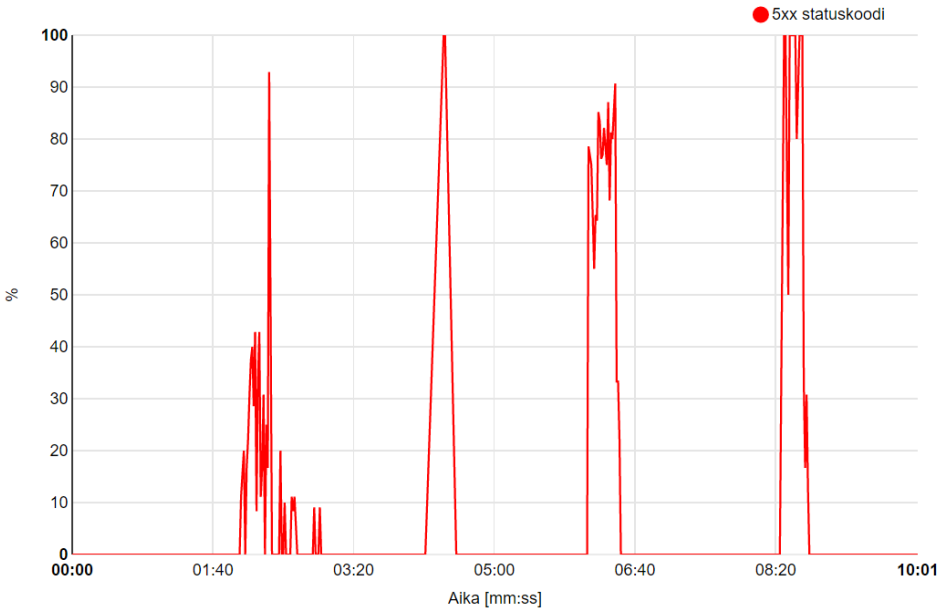
Yhden sekunnin lähetystiheydellä keskusmuistin käyttö nousee noin 20 % (KUVIO 40). Maksimivastausaika on noin 90 sekuntia (KUVIO 41). Vastausajan laskemisen johtuu siitä, kun palvelin ei voi käsitellä enää kaikkia pyyntöjä, niin läpi pääsevien pyyntöjen vastausaika laskee (KUVIO 42). Sama pätee myös muissa tuloksissa. Palvelin käsittelee 83,65 % pyynnöistä (KUVIO 43). Lähetyskaistan käyttö noin 1 750 kbps ja vastaanottokaistan käyttö on noin 450 kbps (KUVIO 44).



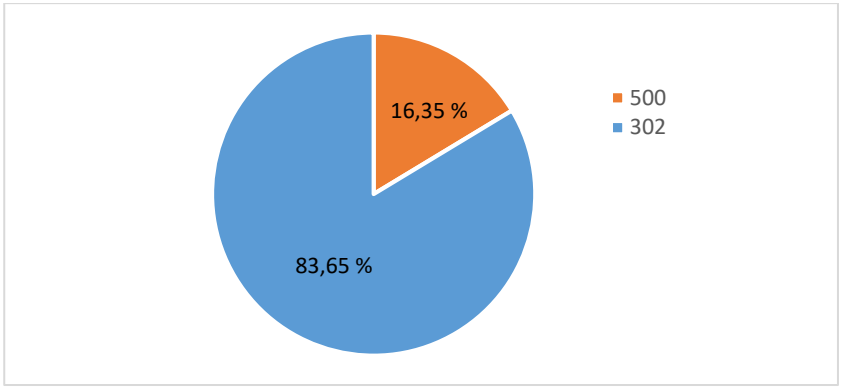
KUVIO 40. POST-tulvan resurssien käyttö 500 yhteydellä ja 1 s lähetystiheydellä



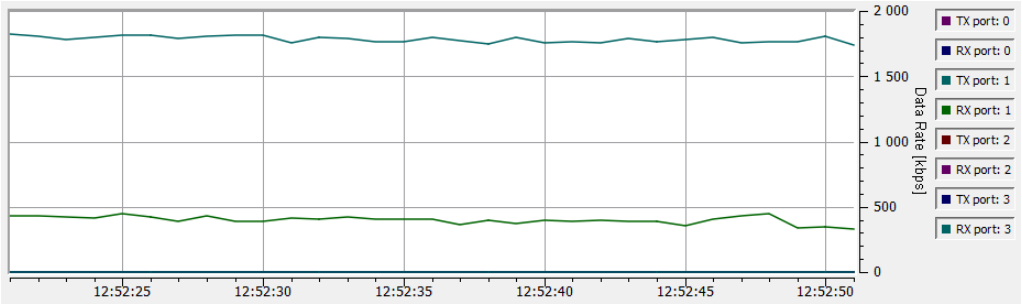
KUVIO 41. POST-tulvan vastausaika 500 yhteydellä ja 1 s lähetystiheydellä



KUVIO 42. POST-tulvan virhekoodien jakautuma 500 yhteydellä ja 1 s lähetystiheydellä



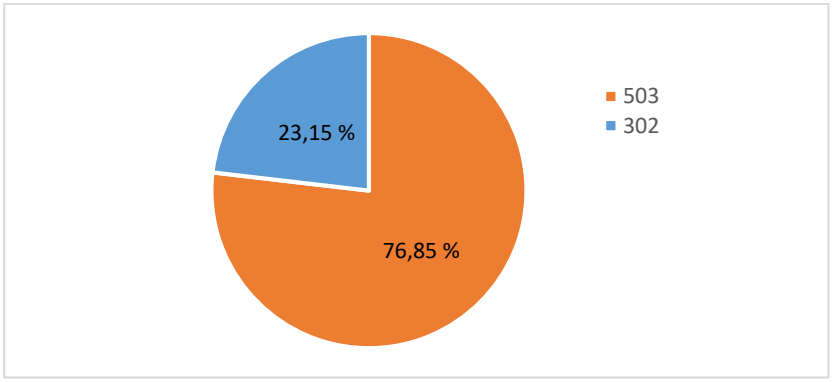
KUVIO 43. POST-tulvan statuskoodit 500 yhteydellä ja 1 s lähetystiheydellä POST-tulvassa



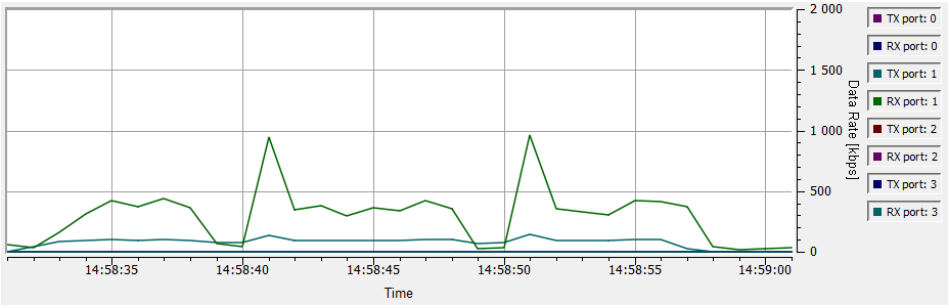
KUVIO 44. POST-tulvan kaistankäyttö 500 yhteydellä ja 1 s lähetystiheydellä

30 sekunnin lähetystiheydellä maksimivastausaika on 80 sekuntia (KUVIO 45). Ainoastaan 23,15 % pyynnöistä onnistuu. 503 virhekoodi johtuu ”SQLSTATE [HY000] [1040] Too many connections” – virheestä, tällöin MySQL-tietokantaan on muodostettu liian monta yhteyttä, joten sivusto ei voi käsitellä pyyntöä. (KUVIO 46.) Rugen maksimi lähetyskaistan käyttö on noin 100 kbps ja maksimi vastaanotto-kaistan käyttö on noin 1 000 kbps (KUVIO 47).

KUVIO 45. POST-tulvan vastausaika 500 yhteydellä ja 30 s lähetystiheydellä

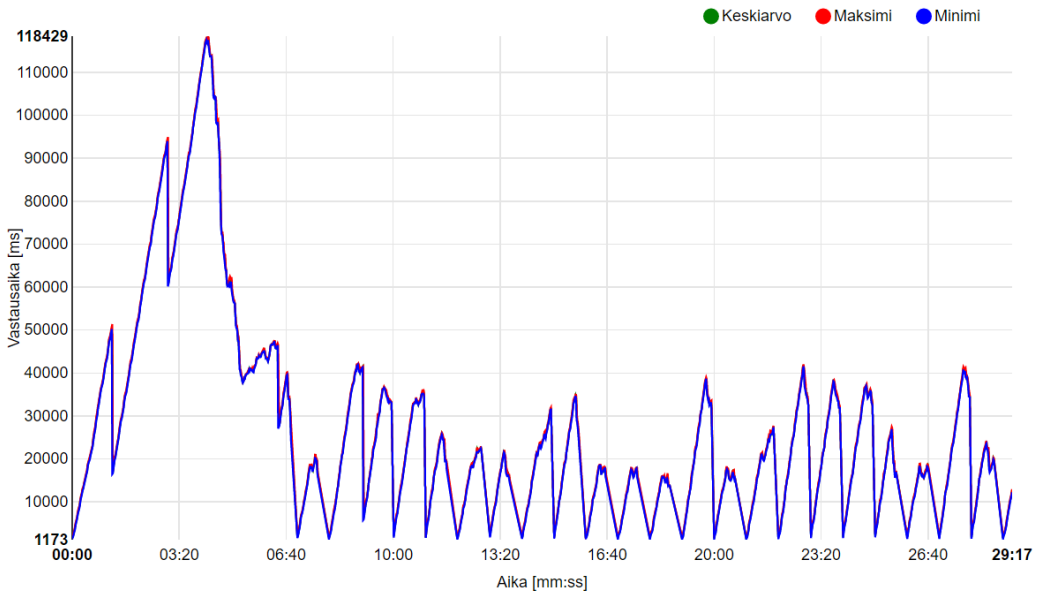


KUVIO 46. POST-tulvan statuskoodien jakautuma 500 yhteydellä ja 30 s lähetystiheydellä

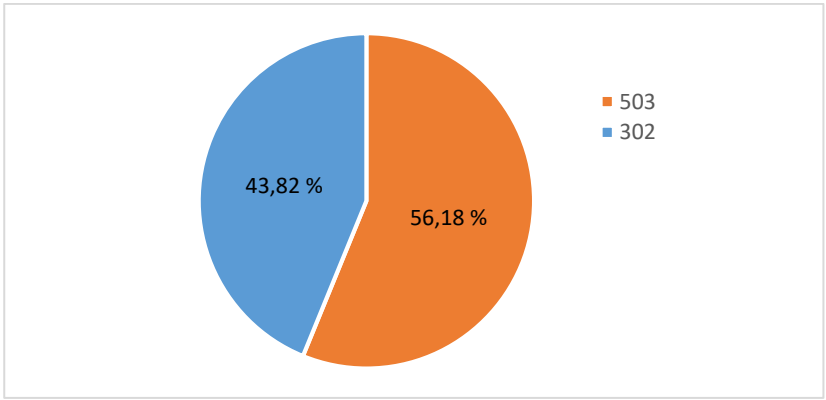


KUVIO 47. POST-tulvan kaistankäyttö 500 yhteydellä ja 30 s lähetystiheydellä

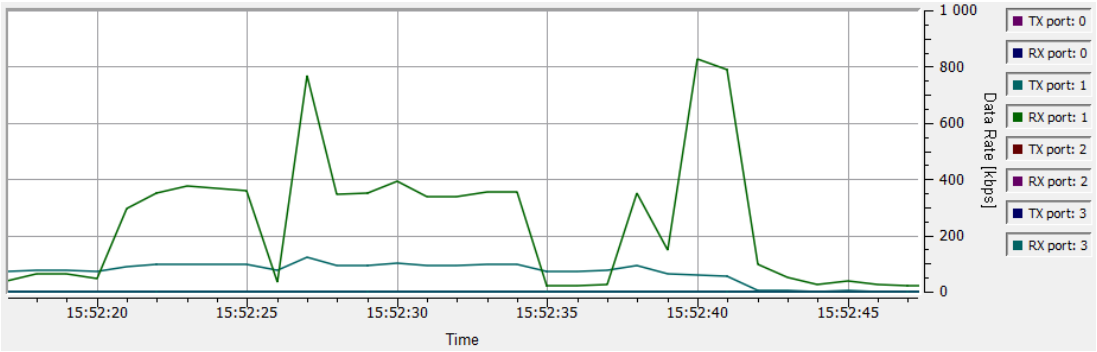
60 sekunnin lähetystiheydellä 56,18 % pyynnöistä hylätään (KUVIO 49). Vastausaika on 4 minuutin kohdalla 120 sekuntia ja samalla virhekoodien määrä alkaa nousta, josta vastausajan putoaminen johtuu. Tämän jälkeen vastausaika on 1 – 40 s. (KUVIO 48.) Rugen lähetyskaistan käyttö on alle 200 kbps ja vastaanotto kaistan käyttö vaihtelee 0 – 800 kbps välillä (KUVIO 50).



KUVIO 48. POST-tulvan vastausaika 500 yhteydellä ja 60 s lähetystiheydellä



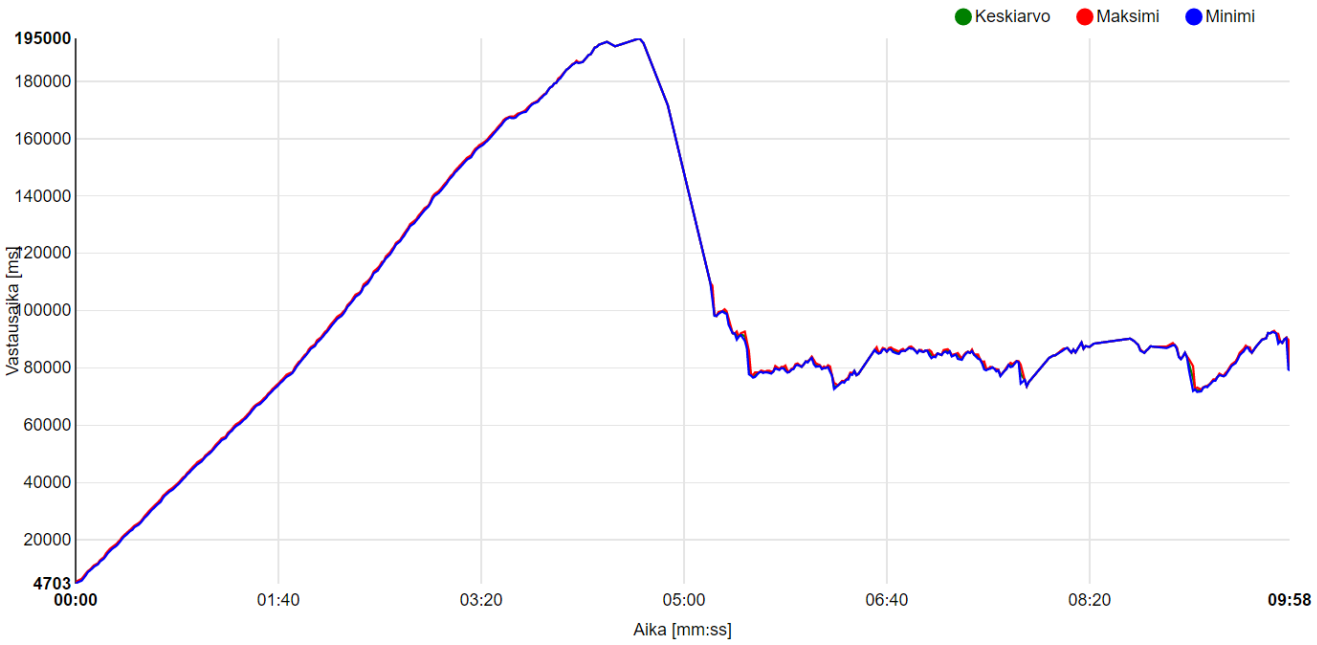
KUVIO 49. POST-tulvan statuskoodien jakautuma 500 yhteydellä ja 60 s lähetystiheydellä



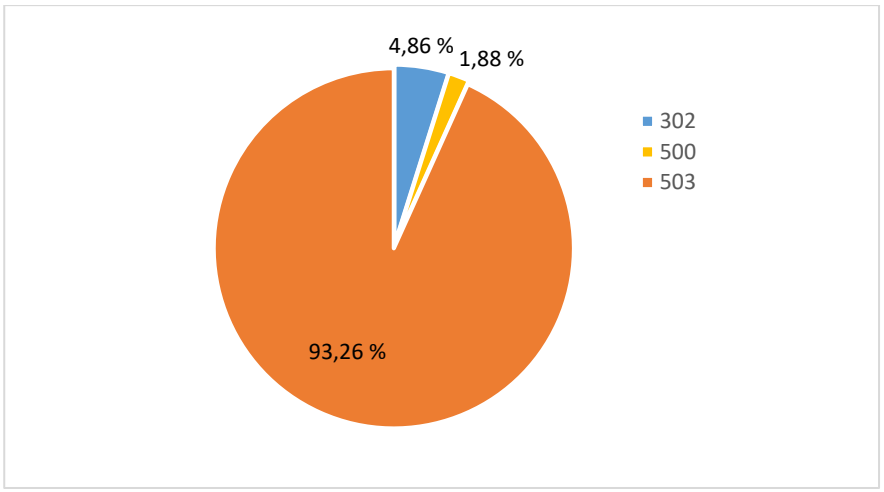
KUVIO 50. POST-tulvan kaistan käyttö 500 yhteydellä ja 60 s lähetystiheydellä

9.3.2 1 000 yhteyttä

1 000 yhteydellä ja yhden sekunnin lähetystiheydellä maksimivastausaika on 195 sekuntia (KUVIO 51). Palvelin käsittelee 4,86 % pyynnöistä (KUVIO 52).

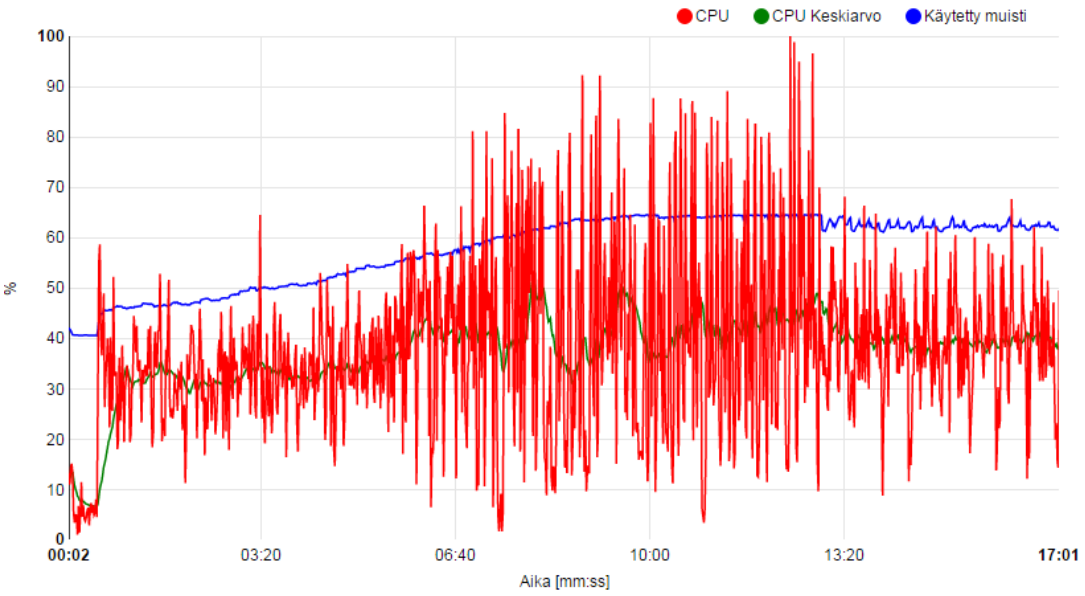


KUVIO 51. POST-tulvan vastausaika 1 000 yhteydellä ja 1 sekunnin lähetystiheydellä

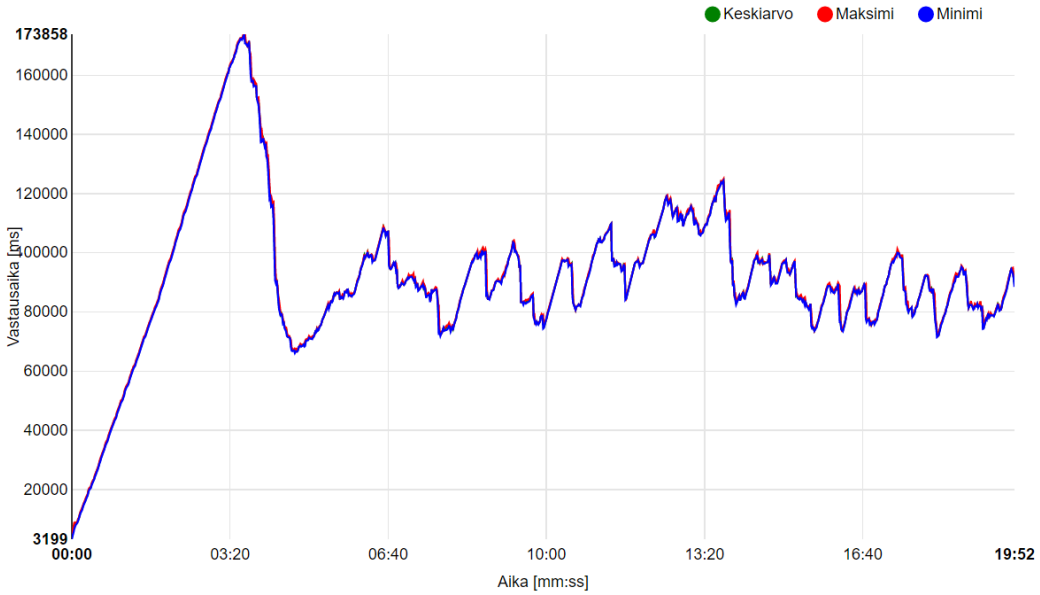


KUVIO 52. POST-tulvan statuskoodien jakautuma 1 000 yhteydellä ja 1 s lähetystiheydellä

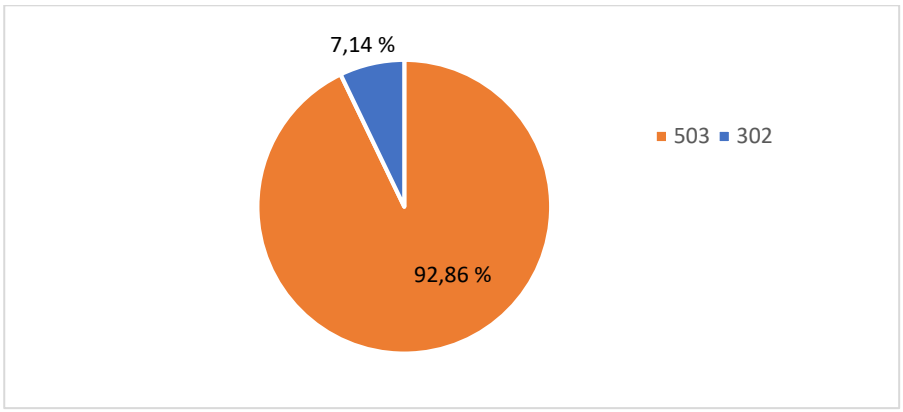
30 sekunnin lähetystiheydellä CPU:n kuormitus vaihtelee paljon ja keskusmuistin käyttö nousee noin 20 % hyökkäyksen aikana (KUVIO 53). Maksimivastausaika on noin 174 sekuntia (KUVIO 54). 30 sekunnilla palvelin käsittelee 7,14 % pyynnöistä (KUVIO 55). 30 sekunnilla rugen lähetyskaistan käyttö on noin 250 kbps ja vastaanottokaistan käyttö vaihtelee 0 – 1 500 kbps välillä (KUVIO 56). 60 sekunnin lähetystiheydellä maksimivastausaika on noin 140 sekuntia (KUVIO 57). 60 sekunnilla tämä on 24,5 %, mikä on 3,43 kertaa enemmän kuin 30 sekunnin lähetystiheydellä (KUVIO 58). 60 sekunnilla vastaanotto kaistan käyttö on välillä myös yli 1 500 kbps, mutta se on selvästi pienempi suuremman osan ajasta (KUVIO 59).



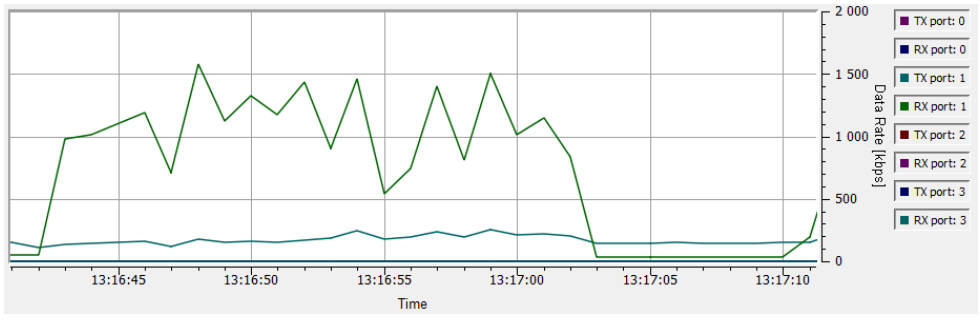
KUVIO 53. POST-tulvan resurssien käyttö 1 000 yhteydellä ja 30 s lähetystiheydellä



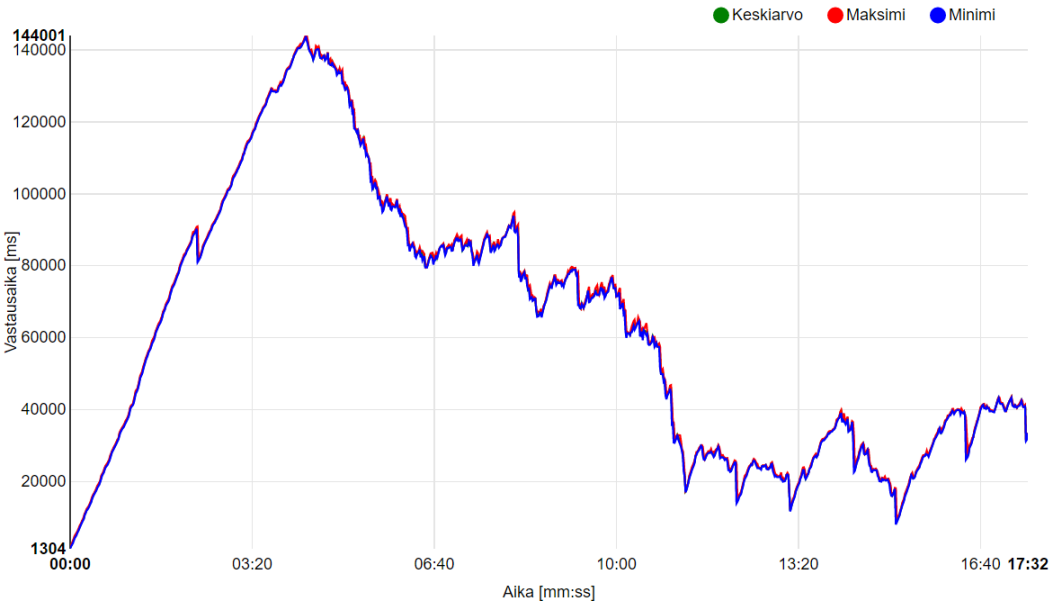
KUVIO 54. POST-tulvan vastausaika 1 000 yhteydellä ja 30 s lähetystiheydellä



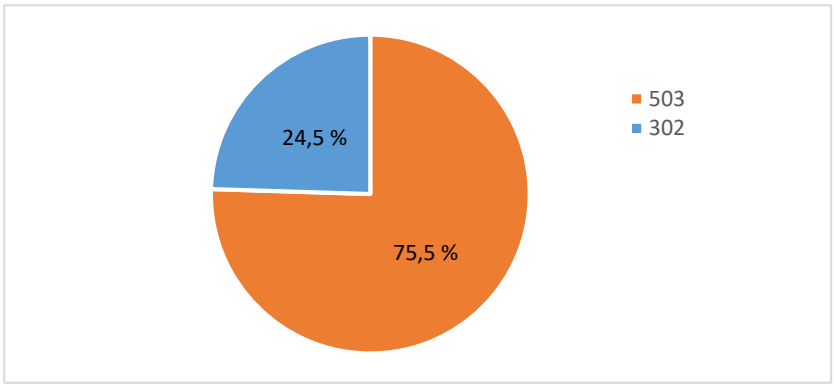
KUVIO 55. POST-tulvan statuskoodien jakautuma 1 000 yhteydellä ja 30 s lähetystiheydellä



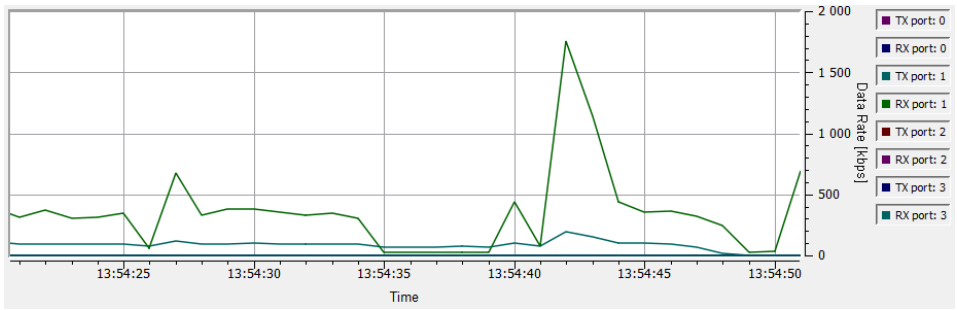
KUVIO 56. POST-tulvan kaistan käyttö 1 000 yhteydellä ja 30 s lähetystiheydellä



KUVIO 57. POST-tulvan vastausaika 1 000 yhteydellä ja 60 s lähetystiheydellä



KUVIO 58. POST-tulvan statuskoodien jakautuma 1 000 yhteydellä ja 60 s lähetystiheydellä



KUVIO 59. POST-tulvan kaistan käyttö 1 000 yhteydellä ja 60 s lähetystiheydellä

9.4 Hitaat hyökkäykset

Kuviossa 60 on hitaan hyökkäyksen rakenne. Ensin avataan TCP-yhteys ja lähetetään varsinainen pyyntö (HTTP_REQ). Pyynnön lopussa on ainoastaan yksi rivinvaihto, joten palvelin odottaa lisää dataa. Seuraavaksi lähetetään silmukassa seuraava pyyntö (HTTP_REQ_2).

	1	2	3	4	5	6	7	8	9	10	11
OFFSET [us]		1									
ITEM	START	TCP/SYN	INIT_HTTP_CTR	TCP/SYN_ACK							
OFFSET [us]					1	1					
ITEM				SYN/ACK	TCP/ACK	HTTP_REQ	LOOP				
OFFSET [us]											
ITEM							Timeout	INC_HTTP_CTR	HTTP_CTR<LIMIT		
OFFSET [us]										1	
ITEM									TRUE	HTTP_REQ_2	LOOP
OFFSET [us]											
ITEM									FALSE	START	

KUVIO 60. Hitaan hyökkäyksen rakenne

9.4.1 Hidas POST

Tässä käytetään tunnuksen luonti päätepistettä. Kuviossa 61 on ensimmäinen pyyntö ja toisessa pyynnössä on ainoastaan runko-osa. Tässä tapauksessa ”x=y”, sisällöllä ei ole väliä. Palvelin vastaa ”400 Bad Request” -virheilmoituksella. 400 statuskoodi viittaa väärään syntaksiin. Oletan, että tämä johtuu siitä, kun IIS ei lue runko-osaa osana edellistä pyyntöä.

```
POST http://magento.lan:11104/index.php/customer/account/createpost/ HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Host: magento.lan:11104
Cookie: frontend=cenr3du2ek3k7cv1n22i8hvmq5
Content-Length: 24
```

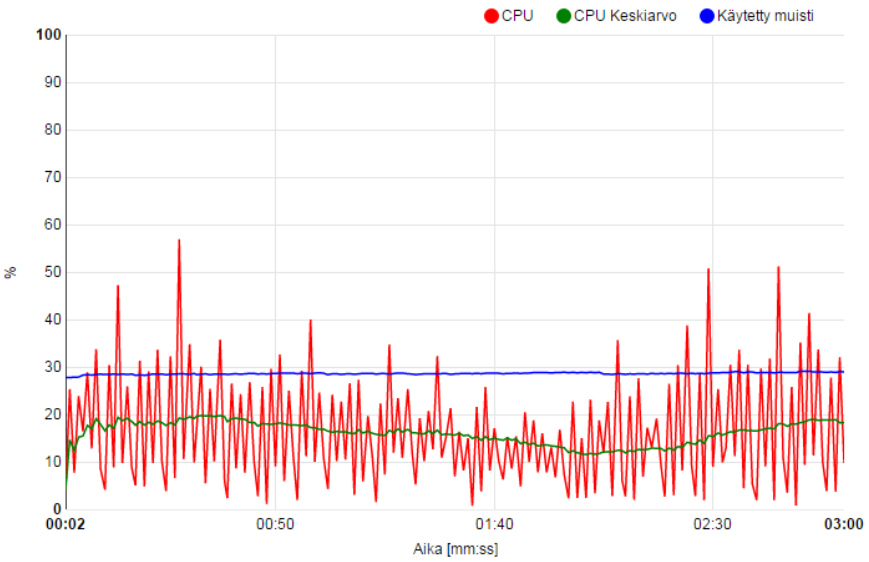
KUVIO 61. Hidas POST -pyyntö

9.4.2 Hidas GET

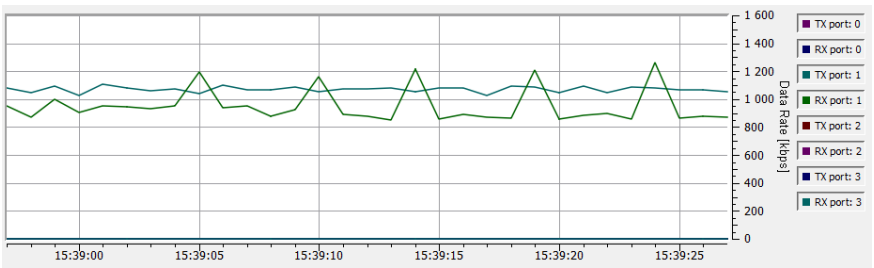
Kuviossa 62 on ensimmäinen pyynnön osa. Seuraavassa pyynnön osassa on ainoastaan ”Pragma: no-cache” -otsikko. Testi tehdään 10 000 yhteydellä ja lähetystiheys on 10 sekuntia. Hyökkäys vaikuttaa hieman prosessorin kuormitukseen eikä vaikutan paljon keskusmuistin käyttöön (KUVIO 63). Sivuston vastausaikaan tämä hyökkäys ei vaikuta. Lähetyskaistan käyttö on noin 1 100 kbps ja vastaanotto-kaistan käyttö on 900 - 1 200 kbps (KUVIO 64).

```
GET http://magento.lan:11104/index.php/ HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: fi-FI
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
Host: magento.lan:11104
```

KUVIO 62. Hidas GET -pyyntö



KUVIO 63. Resurssien käyttö hidas GET -hyökkäyksessä

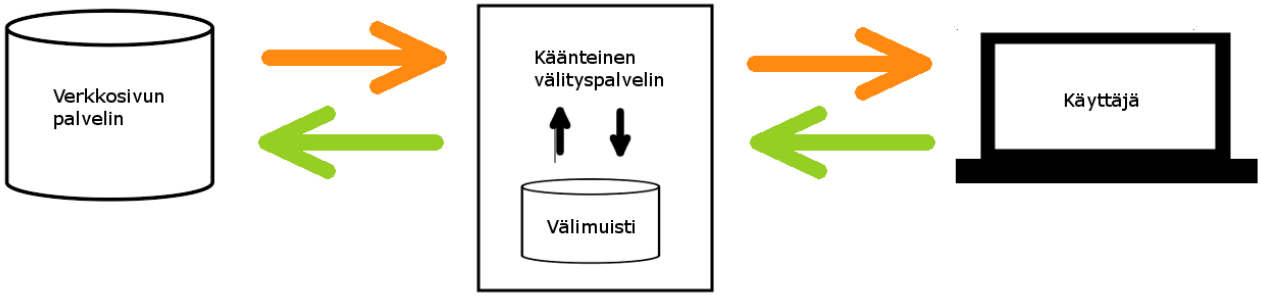


KUVIO 64. Kaistankäyttö Hidas GET -hyökkäyksessä

10 PALVELUNESTOHYÖKKÄYKSIIN VARAUTUMINEN

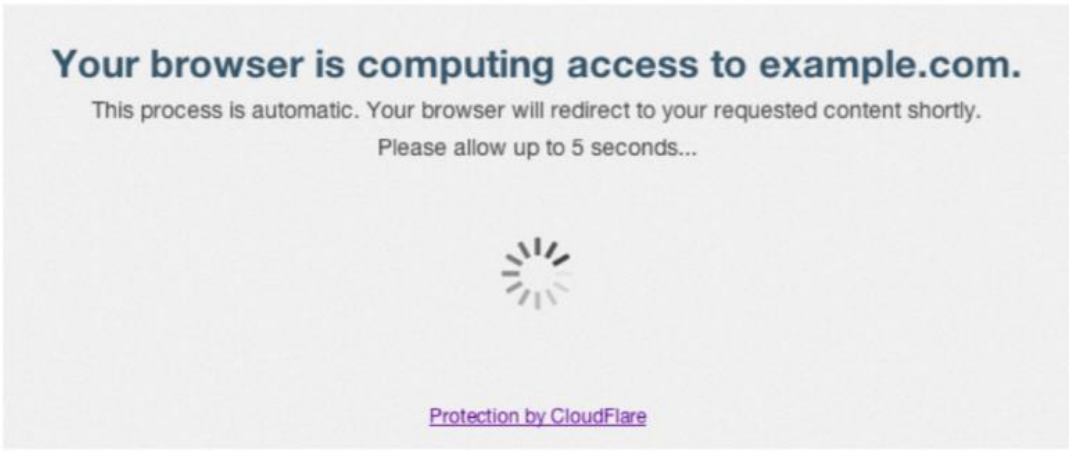
Palvelin kannattaa ylimitoittaa suhteessa käyttäjämäärään, koska tällöin on enemmän resursseja varattuna palvelunestohyökkäys tilanteeseen. Myös palvelunestohyökkäyksen toteuttaminen palvelimella kannattaa tehdä, koska tällöin saadaan selville mistä palvelimen toiminnan hidastuminen johtuu ja mitä sivuvaikutuksia sillä on. Tässä tapauksessa tämän aiheutti resurssien puute, CGI jonon raja ja yhtäaikaisten tietokantayhteyksien määrä. Näiden arvoja pitää muuttaa, jotta saadaan käytettyä palvelimen resurssit mahdollisimman tehokkaasti. Jos verkkosivu ei käytä HEAD-metodia, sen voi ottaa pois käytöstä, niin hyökkääjä ei voi käyttää sitä hyväksi.

Kannattaa myös käyttää käänteistä välityspalvelinta, kuten Nginx ja Varnish. Tällöin kaikki liikenne ohjataan käänteisen välityspalvelimen kautta. Nginx ja Varnish kykenevät toimimaan välimuistipalvelimena, eli ne voivat tallentaa staattisia ja jossain määrin dynaamisia sivuja. Tällöin itse verkkosivun palvelimen kuormitus vähenee, koska sivu voidaan hakea välimuistista. (KUVIO 65.)



KUVIO 65. Käänteinen välityspalvelin

Myös muita käänteisiä välityspalvelimia on olemassa, kuten Cloudflare. Cloudflare pyrkii myös suodattamaan kuljetus- ja sovelluskerroksen haitallisen liikenteen. Cloudflaren mukaan reaaliolosuhteissa Cloudflare vähentää haitallista HTTP-liikennettä 90 %. Cloudflarella on myös I'm Under Attack Mode (IUAM) -ominaisuus (KUVIO 66). Sivuston ylläpitäjä voi laittaa tämän päälle, jos sivusto on hyökkäyksen kohteena. Se pyytää selainta vastaamaan erilaisiin testeihin. Testit käyttävät Javascript-ohjelmointikieltä ja evästeitä. (Cloudflare 2017, 6.)



KUVIO 66. Cloudflaren IUAM-suojaus (Cloudflare 2017, 6)

11 JOHTOPÄÄTÖKSET JA POHDINTA

Opinnäytetyön tavoitteena oli toteuttaa sovelluskerroksen palvelunestohyökkäys Magento-verkkokauppa-alustaan, joka asennettiin IIS-palvelimelle käyttämällä Rugged Tooling Ruge -työkalua ja selvittää, kuinka paljon nämä vaikuttavat palvelimen toimintaan ja miten näitä hyökkäyksiä voidaan torjua. Kaikki hyökkäykset, joita testasin, käyttävät HTTP-protokollaa. Testasin Keep-Dead-, hitaita- ja tulvitushyökkäyksiä.

Keep-Dead -hyökkäys suoritettiin tuotehaku päätepisteeseen. Prosessorin kuormitus nousee 100 % ja keskusmuistin käyttö nousee hieman hyökkäyksen alussa ja sen jälkeen pysyy samana. Vastausaika voi nousta jopa 120 sekuntiin riippuen yhteyksien määrästä ja lähetystiheydestä. Koska Keep-Dead käyttää HEAD-metodia, hyökkääjän vastaanottokaistan käyttö on hyvin pieni.

POST-tulvahyökkäys suoritettiin gallup-päätepisteeseen. Prosessorin kuormitus ei noussut yhtä korkealle kuin Keep-Dead -hyökkäyksen tapauksessa tuotehaku päätepisteeseen. Testauksessa huomasin, että keskusmuistin käyttö nousi noin 20 %, jonka jälkeen se pysyi samana. Tästä hyökkäyksestä huomattiin myös, että verkkosivun ja sen tietokannan välinen yhteys voi ruuhkaantua. Tällöin palvelin ei voi käsitellä pyyntöjä.

GET-tulvahyökkäys suoritettiin Magento-logon resurssiin ja etusivuun. Logon tapauksessa resurssien käyttö ja vastausaika ei noussut paljon. Etusivun tapauksessa vastausaika nousi 70 sekuntiin ja prosessorin kuormitus oli lähes koko ajan 100 %.

Jos hyökkäyksen kohteena on dynaaminen sivu, kuten Keep-Dead, POST- ja GET-tulvahyökkäyksessä etusivuun ja yhteyksien määrä on yli 1 000, niin osalle yhteyksistä tulee ”FastCGI queue limit has been exceeded” -virheilmoitus, eli tätä pyyntöä ei voida käsitellä, koska jono on täysi. 2 000 yhteyden Keep-Dead -testistä huomasin, että tämä tulee noin 84 % pyynnöistä. Hyökkäyksien kaistankäyttö oli myös niin pieni, että tässä tapauksessa hyökkäys voitaisiin toteuttaa yhdestä tyypillisestä kuluttajaliittymästä. Testatut hitaat hyökkäykset eivät näyttäisi toimivan IIS-palvelinta vastaan.

Sovelluskerroksen palvelunestohyökkäyksiä voidaan ehkäistä käyttämällä käänteistä välityspalvelinta, jossa on välimuistituki. On myös olemassa palveluita, jotka välimuistina toimimisen lisäksi pyrkivät suodattamaan haitallisen liikenteen, kuten Cloudflare. Myös palvelimen asetuksia muuttamalla voidaan

kasvattaa sietokykyä, kuten nostamalla FastCGI jonon raja-arvon korkeammaksi, jos vain palvelimella on riittävästi resursseja käytettävissä.

Työssä tuli vastaan muutamia ongelmia. Jouduin vaihtamaan palvelinkoneen aliverkkomaskin ”255.255.255.0”:sta ”255.255.0.0”:n, jotta sain palvelimen vastaanottamaan tietoliikennettä laajemmasta IP-osoitealueesta. Jouduin myös muuttamaan mistä IP-osoitteesta palvelin kuuntelee, koska palvelimessa on kaksi verkkokorttia, joilla on eri IP-osoite.

Rugelle on olemassa laaja valikoima valmiiksi tehtyjä ohjelmia eri palvelunestohyökkäyksien toteuttamiseen. HTTP-protokollan palvelunestohyökkäyksiin ei kuitenkaan löytynyt valmiiksi tehtyjä ohjelmia, joten tein omat. HTTP-vastauksen lukeminen ei ollut ainakaan työn teko hetkellä mahdollista, joten en pystynyt testaamaan sivuston osia, jotka vaativat Cross-Site Request Forgery (CSRF) -tokenin. Haasteelliseksi osoittautui myös silmukan luonti, jotta sain lähetettyä monta HTTP-pyyntöä saman yhteyden sisällä. Kokeilin myös suorittaa hyökkäyksiä pienemmällä lähetystiheydellä ja suuremmalla yhteyksien määrällä, mutta tämä johti Rugen kaatumiseen.

Opinnäytetyö oli erittäin opettava ja kiinnostava. Opin työn aikana käyttämään Rugea ja opin paljon uutta TCP- ja HTTP-protokollista. Opin myös palvelunestohyökkäyksien toiminnasta, toteutuksesta ja torjunnasta paljon. Työ kesti monta kuukautta ja varsinkin testauksien laatiminen ja toteuttaminen kestivät kauan. Työtä voitaisiin jatkokehittää testaamalla muitakin kuin HTTP-protokollan palvelunestohyökkäyksiä ja testaamalla, kuinka paljon erilaiset torjuntamenetelmät auttavat käytännössä.

LÄHTEET

RFC 2616. HTTP/1.1. 1999. Saatavissa: https://tools.ietf.org/html/rfc2616. Viitattu 23.9.2016.

Microsoft. The OSI Model's Seven Layers Defined and Functions Explained. 2014. Saatavissa: https://support.microsoft.com/en-us/kb/103884. Viitattu 25.9.2016.

Ecommerce Usage Statistics. Saatavissa: http://trends.builtwith.com/shop.Viitattu 26.9.2016.

September 2016 Web Server Survey. Saatavissa: https://news.netcraft.com/archives/2016/09/19/september-2016-web-server-survey.html. Viitattu 27.9.2016.

Richard, W. S. 1994. TCP/IP Illustrated, Volume 1. Saatavissa: http://www.cs.new-paltz.edu/~pletcha/NET_PY/the-protocols-tcp-ip-illustrated-volume-1.9780201633467.24290.pdf. Viitattu 2.10.2016.

Kozierok, C. M. 2005. The TCPIP Guide. Saatavissa: https://doc.lagout.org/network/The%20TCPIP%20guide.pdf. Viitattu 2.10.2016.

Rintala M. 2001. TCP-protokolla. Saatavissa: http://mrin.mbnet.fi/paattotyo/tcp.html. Viitattu 3.10.2016.

Keep-Alive DoS Script. 2011. Saatavissa: https://web.archive.org/web/20160312120230/http://www.esrun.co.uk/blog/keep-alive-dos-script/. Viitattu 8.10.2016.

Pillai, S. 2013. SLOWLORIS: HTTP DOS (Denial Of Service) attack and prevention. Saatavissa: http://www.slashroot.in/slowloris-http-dosdenial-serviceattack-and-prevention. Viitattu 8.10.2016

Dantas, Y. G., Nigam, V. & Fonseca, I. 2014. A Selective Defense for Application Layer DDoS Attacks. Federal University of Para´ıba, João Pessoa. IEEE Xplore. Viitattu 9.10.2016.

Shekyan, S. 2016. Slowhttptest. Saatavissa: https://github.com/shekyan/slowhttptest/wiki. Viitattu 18.10.2016

Steam. 2015. Saatavissa: http://store.steampowered.com/news/19852/. Viitattu 28.2.2017.

CloudFlare. Cloudflare advanced DDoS protection. 2017. https://www.cloudflare.com/media/pdf/cloudflare-whitepaper-ddos.pdf. Viitattu 2.3.2017.

Viestintävirasto. 2016. Palvelunestohyökkäykset ovat internetin arkipäivää. Saatavissa: https://www.viestintavirasto.fi/kyberturvallisuus/tietoturvanyt/2016/04/ttn201604291231.html.Viitattu: 14.3.2017.

Osanaiye O., Choo K.R. & Dlodlo M. 2015. Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. Elsevier. Journal of Network and Computer Applications. Viitattu 24.4.2017.