jamk.fi

Sovelluskerroksen DoS-hyökkäykset ja niiden torjunta

Niko Pahajoki

Opinnäytetyö
Toukokuu 2014

Tietotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) Pahajoki, Niko	Julkaisun laji Opinnäytetyö	Päivämäärä 15.05.2014
	Sivumäärä 62	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
Sovelluskerroksen DoS-hyökkäykset ja niiden torjunta

Koulutusohjelma
Tietotekniikan koulutusohjelma

Työn ohjaaja(t)
Saharinen, Karo
Rantonen, Mika

Toimeksiantaja(t)
JYVSECTEC
Niemelä Antti

Tiivistelmä

Opinnäytetyön toimeksiantaja oli JYVSECTEC, joka on Jyväskylän ammattikorkeakoulun kyberturvallisuushanke. Opinnäytetyön tavoitteena oli tutustua sovelluskerroksen palvelunestohyökkäyksiin, niiden vaikutuksiin sekä niiden torjuntaan tai vaikutusten pienentämisen ja tehdä ympäristö, jossa voidaan testata näitä hyökkäyksiä suojaamattomiin ja suojattuihin kohteiseen. Sovelluskerroksen (OSI-mallin taso 7) hyökkäykset ovat yleistymässä, koska nämä hyökkäykset ovat monesti tehokkaampia kuin perinteisemmät hyökkäykset ja vaativat vähemmän resursseja hyökkääjältä.

Ympäristöön luotiin kolme kohdetta, joista yksi oli suojaamaton, ja kaksi muuta eri tavoin suojattuja. Kohteisiin asennettiin Apache-webpalvelin ja Wordpress-websovellus.

Suojaamattoman kohteen Apache-palvelimeen pääsy pystyttiin estämään Slowloris-hyökkäyksellä, jossa kyselyitä lähetetään hitaasti, jolloin yhteyspaikat jäävät varatuksi hyökkääjälle. Kohteessa 2 tätä vastaan suojauduttiin käyttämällä Reqtimeout ja ModSecurity-moduuleita. Kohteessa 3 suojauduttiin käyttämällä nginx-webpalvelinta käänteisenä välityspalvelimena Apachelle.

Suojaamattoman kohteen Wordpressiin hyökättiin Keep-Dead-hyökkäyksellä, jolla saatiin estettyä sen toiminta. Hyökkäystä kokeiltiin myös suoraan webpalvelimeen, mutta se ei vaikuttanut toimintaan. Kohteessa 2 tätä vastaan suojauduttiin rajoittamalla yhdestä IP-osoitteesta tulevia kyselyjä sekä PHP:n välimuistirat-kaisulla. Kohteessa 3 suojauduttiin tätä vastaan lisäämällä nginx-webpalvelimeen välimuisti, mikä vä-hensi Apachelle ohjautuvien kyselyiden määrää.

Avainsanat (asiasanat)
Palvelunestohyökkäys, Apache, nginx, Slowloris

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Pahajoki, Niko	Bachelor´s Thesis	15.05.2014
	Pages 62	Language Finnish
		Permission for web publication (X)

Title
Application-level DoS-attacks and migitation

Degree Programme
Information Technology

Tutor(s)
Saharinen, Karo
Rantonen, Mika

Assigned by
JYVSECTEC
Niemelä Antti

Abstract

The bachelor's thesis was assigned by JYVSECTEC, a cybersecurity project administered by JAMK University of Applied Sciences. The goal of this thesis was to research the application-level denial of service attacks, effects of these attacks and mitigation techniques and create a test environment with the ability to test these attacks on protected and unprotected targets. Application-level attacks (OSI Layer 7) are increasing in popularity, since these attacks can have a more significant effect on a target than more conventional attacks while requiring less resources from the attacker.

The testing environment had three servers, one of which was unprotected and two were protected using different methods. The target servers had an Apache webserver and Wordpress web application installed.

The unprotected targets were attacked with Slowloris type attack, where requests are sent slowly, which results in connection slots tied to the attacker. Server 2 was protected using Reqtimeout and ModSecurity modules and Server 3 using nginx as frontend webserver for Apache.

Wordpress in the unprotected server was attacked using Keep-Dead attack, which was able to prevent Wordpress from working. This attack was also tried directly to the server, however, it did not work. Server 2 was protected using request limits and APC opcode and object cache. Server 3 was protected by adding caching to nginx frontend, which resulted in less queries to Apache.

Keywords
DOS, Denial of Service, Apache, nginx, Slowloris

Miscellaneous

Sisältö

Kuviot

Taulukot

Lyhenteet

CIA	Confidentiality Integrity Availability
CPU	Central Processing Unit
DNS	Domain Name System
DOS	Denial of Service
DDOS	Distributed Denial of Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
NAT	Network Address Translation
NTP	Network Time Protocol
MITM	Man-in-the-Middle
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PHP	PHP: Hypertext Preprocessor
RGCE	Realistic Global Cyber Environment
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
XSS	Cross Site Scripting
WAF	Web Application Firewall

1 Lähtökohdat

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantaja oli JYVSECTEC (Jyväskylä Security Technology), joka on Jyväskylän ammattikorkeakoulun vuonna 2011 aloitettu kyberturvallisuushanke. JYVSECTEC ylläpitää ja kehittää RGCE-ympäristöä (Realistic Global Cyber Environment). (Jyvsectec 2013.)

RGCE-ympäristö on tehty internetin kaltaiseksi sekä palveluiltaan että rakenteeltaan käyttämällä sekä fyysisiä että virtualisoituja ratkaisuja. Tämä mahdollistaa erilaisten tietoturvaratkaisujen testaamisen internetistä eristettynä, jolloin muille palveluille ei aiheudu vaaratilannetta testaamiten seurauksena. (Jyvsectec 2013.)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli tutkia ja testata palvelunestohyökkäyksiä sovelluskerroksella. ja tapoja tunnistaa, torjua tai pienentää niiden vaikutusta, sekä toteuttaa ympäristö, jossa voidaan testata näitä hyökkäyksiä ja tarkastella niiden vaikusta suojaamattomiin ja suojattuihin kohteisiin.

2 Tietoturva ja palvelunestohyökkäykset

2.1 Tietoturva

Tietoturvalla tarkoitetaan tiedon turvaamista luvattomalta pääsyltä, käytöltä, häirinnältä, muokkauksilta, kopioinnilta tai tuhoamiselta, voidaan jakaa kolmeen osa-alueeseen CIA-mallin mukaan:

1. Luottamuksellisuus (confidentiality): Tietoa voivat käsitellä vain ne joilla on siihen tarvittavat oikeudet.
2. Eheys (intergrity): Tieto ei saa muuttua tahattomasti esimerkiksi levyrikon, eikä tahallisesti esimerkiksi hyökkäyksen seurauksena.
3. Saatavuus (availability): Tieto on saatavilla silloin kuin sitä tarvitaan. (Tipton & Krause 2007.)

2.2 Palvelunestohyökkäykset

Palvelunestohyökkäyksillä pyritään estämään tiedon saatavuus eli ne kohdistuvat saatavuuteen. Tietoturva on yhä tärkeämpää, koska suuri osa palveluista on siirtynyt internettiin. (Tipton & Krause 2007.)

Palvelunestohyökkäys (Denial of Service Attack, DoS) on hyökkäys, jolla pyritään estämään pääsy palveluun ja siten estämään siellä olevan tiedon saatavuus. Tähän kuuluvat myös fyysiset uhat, mutta käytännössä lähes kaikki hyökkäykset tapahtuvat verkon yli. (Prowell, Kraus & Borkin 2010.)

Haktivismi on internetissä tapahtuvaa aktivismia, jolla pyritään vaikuttamaan johonkin tahoon, yleensä kohteeseen. Esimerkiksi Anonymous-ryhmän palvelunestohyökkäykset Paypallia, Visaa ja Mastercardia vastaan näiden lopetettua lahjoitusten välittämisen Wikileaksille. (Fantz & Shuber 2010.)

Palvelunestohyökkäys voidaan tehdä myös kiristykseksi tai kaupallisista syistä esimerkiksi kilpailijaa vastaan. Kiristystapauksissa hyökkäystä jatketaan, kunnes kohde suostuu maksamaan lunnaat, tai jos kohde onnistuu torjumaan hyökkäyksen, jolloin kohteelle ei jää syytä maksaa lunnaita. Kiristyksen kohteena ovat monesti nettikasinot, joissa liikkuu paljon rahaa ja lyhyet katkokset voivat aiheuttaa asiakkaiden siirtymisen toiseen palveluun. Palvelunestohyökkäyksen kohteeksi voi joutua käytännössä mikä tahansa palvelu ja myös syistä, jotka eivät välttämättä suoraan liity itse palveluun, esimerkiksi näyttämisenhalusta tai kokeilumielessä tehty hyökkäys. (Leyden 2013.)

Palvelunestohyökkäykset voidaan luokitella OSI-mallin mukaan fyysisen kerroksen, verkkokerroksen, kuljetuskerroksen ja sovelluskerroksen hyökkäyksiin. Nämä eroavat toisistaan hyökkäystavoilta, vaikutuksiltaan ja torjunnalta. Fyysisen kerroksen palvelunestohyökkäyksiin luokitellaan esimerkiksi kaapelin katkaiseminen. Hankalin toteuttaa ja siten harvinaisin, mutta esimerkiksi merikaapeleita on yritetty katkaista. Tällöin kohteena on koko maan ulkomaanliikenne, joka voi hidastua tai pahimmillaan katketa kokonaan. (Thomson 2013.)

Verkkokerroksen palvelunestohyökkäyksissä kohteeseen hyökätään suurella liikennemäärällä. Helppo toteuttaa, koska kohteessa ei tarvitse olla haavoittuvaa palvelua,

vaan riittää hyökkääjään liikenne saapuu kohteeseen. Jotta hyökkäys olisi tehokas, hyökkääjältä vaaditaan enemmän kaistaa, kuin mitä kohteella on. (Mehmud 2011, 2.)

Verkkokerroksen hyökkäyksiin kuuluu esimerkiksi ICMP-tulvitus (ICMP Flood). Tulvitushyökkäyksissä (Flood attack) kohde yritetään hukuttaa suureen pyyntö- tai liikennemäärään. ICMP-tulvituksessa kohteeseen lähetetään ICMP Echo Request-pyyntöjä. Kohteen lamaannuttamiksesi hyökkääjällä täytyy olla käytettävissä enemmän kaistaa, kuin mitä kohteella on. (Prowell, Kraus & Borkin. 2010.)

Kuljetuskerroksen hyökkäyksiin kuuluu esimerkiksi SYN-tulvitus, jossa hyödynnetään TCP-protokollan kolmivaiheista kättelyä:

1. Asiakas lähettää avauspyynnön (SYN)
2. Palvelin kuittaa avauspyynnön (SYN+ACK)
3. Asiakas kuittaa takaisin (ACK)

SYN-tulvituksessa lähetetään väärennetty IP-osoite ja palvelin ei saa vastapuolelta kuittausta. Tällöin yhteys jää puoliavoimeksi (half-open), kun palvelin jää odottamaan vastausta. Palvelimen täytyy olla tapa tietää asiakkaan lähettäneen avauspyynnön vastatakseen kolmannen vaiheen kuittauksen. (Tipton & Krause 2007.)

Yhteistä verkosta tuleville hyökkäyksille on niiden pyrkimys kuluttaa palvelulta sen resurssit, jonka seurauksena resursseja ei jää oikeiden palvelimiseen.

Nykyään palvelut ovat monesti riippuvaisia myös toisista palveluista. Esimerkiksi käytännössä kaikki internetissä olevat palvelut käyttävät domain-nimiä, jolloin tarvitaan nimipalvelin muuttamaan domain IP-osoitteeksi, jolloin voidaan suorittaa palvelunestohyökkäys nimipalvelimeen. Muita riippuvaisuuksia voivat olla esimerkiksi kolmannen osapuolen tarjoamat maksupalvelut, johon suoritettu palvelunestohyökkäys estää maksutapahtumat. (Prowell ym. 2010.)

2.3 Hajautetut palvelunestohyökkäykset

Nykyään lähes kaikki palvelunestohyökkäykset tulevat useasta lähteestä, koska yhdestä lähteestä tuleva palvelunestohyökkäys on helppo torjua, eikä se välttämättä

edes häiritse palvelun toimintaa. Tällöin puhutaan hajautetusta palvelunestohyökkäyksestä (Distributed Denial of Service, DDoS). (Mehmud 2011, 3-4.)

Hajautettuihin palvelunestohyökkäyksiin käytetään bottiverkkoja (botnet). Bottiverkot ovat useista orjakoneista koostuvia verkkoja. Bottiverkoissa orjakoneet (zombie) toteuttavat ohjauspalvelimelta (Command & Control) saamiansa käskyjä. Palvelunestohyökkäysten lisäksi bottiverkkoja voidaan käyttää esimerkiksi roskapostin lähettämiseen. (Zhang ym. 2011.)

Ohjauspalvelin voi olla keskitetty, jolloin käskyt saadaan yhdeltä tai useammalta palvelimelta, mutta erityisesti uudemmat bottiverkot ovat P2P-periaattella toimivia. P2P-bottiverkoissa orjakoneet muodostavat yhteyksiä toisiinsa ohjeiden vaihtoa varten. Tämän takia bottiverkon sulkeminen ohjauspalvelimen sulkemalla ei ole enää mahdollista. (Dae-il ym. 2009.)

Yleensä bottiverkot koostuvat haittaohjelmilla kaapatuista koneista, mutta selainten kehittyminen on avannut uusia mahdollisuuksia toteuttaa hyökkäyksiä käyttämällä Javascriptiä, jonka avulla hyökätään kohteeseen. (Grossman & Johansen 2013; Zhang ym. 2011.)

3 Sovelluskerroksen palvelunestohyökkäykset

3.1 Yleisesti

OSI-mallin sovelluskerroksen palvelunestohyökkäyksissä (Application Level Denial of Service) hyödynnetään ohjelmistosta löytyviä virheitä tai konfiguraatiovirheitä palvelunestohyökkäyksen toteuttamisessa. Nämä ovat monesti tehokkaampia ja vaativat hyökkääjältä vähemmän resursseja kuin perinteisemmät verkkotason tulvitushyökkäykset. Toisaalta kohteeltakin vaaditaan haavoittuva palvelu, jota vastaan hyökätään. Riippuen hyökkäyksestä vaikutus voi kohdistua koko palvelimeen tai vain yhteen palveluun, jolloin muut ovat vielä käytettävissä. (Durcekova ym. 2012.)

Yksinkertaisimmillaan sovelluskerroksen hyökkäys voi olla HTTP-kyselytulvitus, jossa lähetetään kelvollisia HTTP-kyselyitä. Koska kyselyt ovat kelvollisia, palomuurit eivät voi helposti erottaa aitoja ja hyökkäykseen kuuluvia kyselyjä. Hyökkäyksissä pyritään

kuluttamaan palvelimelta rajallisia resursseja esimerkiksi CPU-aikaa tai muistia (Durcekova ym. 2012.).

Peilaushyökkäyksessä (Reflection attack / Amplification attack) lähetetään kysely UDP-protokollaa käyttävään palveluun käyttäen väärennettyä lähdeosoitetta. Tämä on mahdollista, koska UDP on tilaton protokolla eli se ei vaadi TCP:n tapaan yhteyden muodostusta. Peilaushyökkäyksessä kohteeseen lähetetään protokollan kannalta toimiva kysely, joka tuottaa mahdollisimman ison vastauksen. Peilaushyökkäystä voidaan käyttää tukkimaan kohteen ulospäin lähtevä yhteys tai hyökkäämään kolmannen osapuolen palveluun. Hyökkäyksessä käytetään sellaisia komentoja, jotka tuottavat pienellä kyselyllä suuren vastauksen ja eivät varmista lähettäjää. Esimerkiksi, jos 50 tavun kysely tuottaa 500 tavun vastauksen, hyökkääjä saa kasvatettua volyymin 10-kertaiseksi eli hyökkääjän lähettäessä kyselyjä 10 megabitin nopeudella, pavelin lähettää vastauksia uhrille 100 megabitin nopeudella. (DNS Amplification Attacks 2013.)

Arbor Networksin tutkimuksen mukaan 24 % hyökkäyksistä on sovelluskerroksen-hyökkäyksiä ja vastaajista 82 % on hyökätty eri HTTP-protokollan hyökkäyksillä, 77 % on hyökätty DNS-protokollan hyökkäyksillä ja 54 % HTTPS-protokollan hyökkäyksillä. (Arbor Networks 2013.)

3.2 Webpalvelimet

3.2.1 Hyökkäystavat

Webpalvelimet käyttävät pääosin HTTP-protokollaa sivujen siirtämiseen. Sen rinnalla on kuitenkin käytössä myös muita harvinaisempia protokollia, joita voidaan käyttää joissakin tapauksissa, mutta varsinaisia korvaajia ne eivät ole.

HTTP:n rinnalla on olemassa Googlen kehittämä SPDY, joka on muokattu versio HTTP-protokollasta. SPDY:n pyrkimys on vähentää kaistankäyttöä kaistan käyttöä. Esimerkiksi otsikot lähetetään pakattuna. Se on tällä hetkellä kuitenkin erittäin harvinainen ja käytössä alle prosentissa websivuista. HTTP:n version 2.0 työversio perustuu SPDY-protokollaan, joilloin nämä parannukset ovat mahdollisesti tulossa HTTP-protokollaan. (Usage of SPDY for websites 2014; Nottingham 2012.)

WebSocket on websovellusten tapa siirtää tietoa molempiin suuntiin, ja se on tarkoitettu esimerkiksi reaaliaikasovelluksiin. Websocket-yhteyden muodostukseen käytetään HTTP-protokollan mukaista kyselyä, jossa on tarvittavat otsikot, mikä mahdollistaa saman portin käyttämisen sekä HTTP- että WebSocket-yhteyksille. (Fette & Melnikov 2011.)

HTTP-protokollassa on määritelty useampi metodi, joilla on oma tarkoituksensa. Tärkeimmät näistä ovat GET ja POST. GET-metodia käytetään sivun hakemiseen palvelimelta. Kuviossa 1 esimerkki tyypillisestä sivulatauksessa, jossa on käytetty GET-metodia. HEAD-metodi toimii kuten GET-metodi, mutta varsinaista sisältöä ei palauteta, ainoastaan HTTP-otsikot, mikä säästää kaistaa, jos varsinaista tiedostoa ei tarvita. (Fielding ym. 1999.)

```
Hypertext Transfer Protocol
⊞ GET / HTTP/1.1\r\n
  Accept: text/html, application/xhtml+xml, */*\r\n
  Accept-Language: en-US,en;q=0.7,fi;q=0.3\r\n
  User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0)\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Connection: Keep-Alive\r\n
  Host: www.iltasanomat.fi\r\n
  \r\n
```

Kuvio 1. Esimerkki HTTP GET-pyynnöstä

POST-metodia käytetään lomaketietojen tai tiedostojen siirrossa palvelimelle. Lomake voi olla esimerkiksi palautelomake. Lomaketietoja voidaan kuitenkin siirtää myös GET-metodissa, jos ne siirretään osana osoitetta, esimerkiksi hakulomakkeet, tämä mahdollistaa linkittämisen sivuihin ja siten myös hakukoneella löytymisen. Kuviossa 2 on esimerkki HTTP POST-pyynnössä, jossa on mukana lomaketietoja. (Fielding ym. 1999.)

```
⊞ POST / HTTP/1.1\r\n
  Host: 192.168.1.46\r\n
  User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:26.0) Gecko/20100101 Firefox/26.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: fi-fi,fi;q=0.8,en-us;q=0.5,en;q=0.3\r\n
  Accept-Encoding: gzip, deflate\r\n
  DNT: 1\r\n
  Referer: http://192.168.1.46/\r\n
  Connection: keep-alive\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
⊞ Content-Length: 11\r\n
  \r\n
  [Full request URI: http://192.168.1.46/]
  [HTTP request 1/1]
  [Response in frame: 2]
Line-based text data: application/x-www-form-urlencoded
  a=1&b=2&c=3
```

Kuvio 2. Esimerkki HTTP POST-kyselystä

HTTP-kyselyihin voidaan määritellä otsikoita, jotka erotellaan rivivaihdolla ja viimeisen otsikon jälkeen lähetetään vielä uusi rivivaihto, jotta palvelin tietää kyselyn päättyneen ja voi siirtyä vastaamaan siihen. Otsikoilla kerrotaan palvelimella lisätietoja pyynnöstä. Ainoastaan sivuston DNS-nimi (Host) on pakollinen, tämä mahdollistaa useamman eri sivuston samassa IP-osoitteessa. (Fielding ym. 1999.)

Pakollisen Host-otsikon lisäksi selaimet yleensä kertovat ainakin tuetut tiedostomuodot (Accept), halutut kielet (Accept-Language), selaimen (User-Agent), tuetut pakkausmuodot (Accept-Encoding). (Fielding, ym. 1999.)

Tietojen lähetyksessä POST-metodilla määritellään näiden lisäksi myös tietojen lähetysmuoto (Content-Type) ja tietojen pituus (Content-Length). (Fielding ym. 1999.)

Hitaissa hyökkäyksissä (slow rate attack) kulutetaan palvelimen resursseja pitkittämällä yhteyttä. Yhteyden pitkittäminen onnistuu esimerkiksi pidentämällä kyselyä ns. turhalla datalla, joka on kuitenkin protokollan mukaista ja lähettämällä se palvelimelle mahdollisimman hitaasti eli juuri ennen kuin palvelin katkaisisi yhteyden. Tästä syystä hyökkäys ei vaadi hyökkääjältä suurta kaistaa, kuten esimerkiksi tulvitukseen perustuvat hyökkäykset. (Hansen 2009.)

Yhteyksien kirjaus logiin tapahtuu yhteyden päätyttyä, joten hyökkäyksen aikana ei logiin tule merkintöjä. Jos hyökkääjä lopettaa yhteyden ajoissa hyökkäyksestä jää samanlainen merkintä kuin tavallisesta onnistuneesta yhteydestä (200 OK). Hyökkääjä voi testata palvelimen aikakatkaisu ajan, jolloin hyökkäys voidaan mitoittaa kohteelle oikein, jolloin virheitä tulee mahdollisimman vähän. (Hansen 2009.)

Slowloris-hyökkäyksessä käytetään hyväksi mahdollisuutta määritellä otsikoita ennalta määrittämättömiä otsikoita. Hyökkäyksessä palvelimelle lähetetään mahdollisimman harvoin uusi otsikko, jotta palvelin ei aika katkaise yhteyttä. Tätä jatketaan, kunnes palvelin katkaisee yhteyden. Näin saadaan varattua palvelimelta yhteyksiä mahdollisimman pitkäksi aikaa. Koska hyökkäyksessä ei juuri lähetetä dataa, voi hitaallakin yhteydellä avata satoja yhteyksiä. Slowloris-hyökkäys ei ole riippuvainen käytetystä metodista, mutta joissakin tapauksissa saattaa vaikuttaa hyökkäyksen toimivuuteen, FreeBSD mahdollistaa HTTP-kyselyiden puskuroimisen Kernel-tasolla. Toiminto rajoittuu kuitenkin salaamattomiin yhteyksiin, GET ja HEAD-metodeihin,

jolloin hyökkäys onnistuu esimerkiksi HTTPS yli tai käyttämällä POST-metodia. (Hansen 2009; FreeBSD Project 2000.)

Toiminnaltaan tätä voidaan verrata SYN-tulvitus hyökkäykseen, jossa jätetään TCP-yhteyden avaus kesken ja Slowloris-hyökkäyksessä avataan TCP-yhteys, mutta ei lähetetä täydellistä HTTP-kyselyä.

Slow HTTP POST -hyökkäys (myös RUDY tai r-u-dead-yet) on toinen HTTP-protokollaan perustuva hidas hyökkäys, joka kohdistuu POST-metodia käyttäviin lomakkeisiin. Hyökkäyksessä lähetetään otsikot normaalisti, mutta kyselyyn liitetään mahdollisimman pitkä lomaketieto. Apache sallii oletusasetuksilla kahden gigatavun lomaketiedot, jolloin yhteys saadaan varattua hyvin pitkäksi aikaa. (Onn Chee & Brennan 2010; Apache HTTP Server Version 2.4 Documentation 2014.)

Kuten otsikot, myös lomaketiedot voidaan lähettää hitaasti, jolloin palvelin joutuu odottamaan kauemmin ja hyökkääjän ei tarvitse kuluttaa kaistaa hyökkäyksen toteuttamiseen, joka tässäkin tapauksessa mahdollistaa satojen yhteyksien avaamisen yhdestä lähteestä. Riippuen palvelimen asetuksista, tämä voi edellyttää olemassa olevaa POST-metodia käyttävää lomaketta. (Onn Chee & Brennan 2010.)

Molemmissa hyökkäyksissä pyritään pitämään palvelimeen mahdollisimman monta avointa yhteyttä, jolloin oikeille käyttäjille ei jää vapaita yhteyksiä ja ne joutuvat jonoon odottamaan. Tämän takia ne toimivat hyvin webpalvelimiin, jotka rajoittavat yhteyksien määrää. (Onn Chee & Brennan 2010.)

Säikeistetyissä käsitellään yhtä yhteyttä säiettä kohden, tämä rajoittaa käsiteltävien yhteyksien määrän säikeiden määrään. Säikeiden määrää rajoittaa puolestaan palvelimen muut resurssit, kuten keskusmuisti. Jos yhteyksien määrää ei rajoiteta, tämä mahdollistaa palveluneston kuluttamalla palvelimen muisti, jolloin palvelin joutuu lopettamaan prosesseja tai siirtämään osa muistin sisällöstä swap-muistiin. (Erb 2012.)

Esimerkiksi suosittu Apache perustuu säikeisiin ja yhteyksien määrää rajoittaa asetuksissa sallittu säikeiden ja/tai prosessien määrä. Ylimenevät yhteydet jäävät jonoon ja tämä näkyy käyttäjälle hitautena tai selaimen aikakatkaisuvirheenä, riippuen kuinka kauan yhteydet joutuvat jonossa odottamaan. (Onn Chee & Brennan 2010; Apache HTTP Server Version 2.4 Documentation 2014.)

Hyökkäyksellä ei ole läheskään yhtä suurta vaikutusta tapahtumapohjaisiin (event-driven) webpalvelimiin. Näissä kussakin säikeessä käsitellään vuorotellein useampaa yhteyttä ja säikeitä on vain muutamia, esimerkiksi yksi jokaista prosessorin ydintä kohden. Esimerkiksi nginx on tapahtuma-pohjainen.

Slow Read hyökkäyksessä palvelimelta ladataan iso sivu (ei mahdu lähettäjän TCP-puskuriin), tässä voidaan hyödyntää HTTP-Pipelining ominaisuutta, mikä sallii samaan aikaan useiden sivujen pyytämisen, ennen edellisen kyselyn valmistumista. (Shekyan 2013.)

SlowHTTPTest on testausohjelma, jolla voidaan testata sekä Slowloris- että RUDY-hyökkäystä. Näiden lisäksi tuettuina on Slow Read ja Apache Range Header-hyökkäykset. SlowHTTPTest ei muodosta uusia yhteyksiä niiden päätyttyä, joten sillä voidaan mitata helposti kauanko kestää, että palvelin katkaisee yhteydet. (Shekyan. 2013.)

Slowloris-ohjelma ei anna minkäänlaista tilastotietoa, mutta sitä voidaan käyttää varsinaisen hyökkäyksen simuloimiseen ja testata kuinka palvelin siitä selviää. Tällöin mittaamiseen voidaan käyttää esimerkiksi Apache Benchmark –ohjelmaa, jolla mitataan sivun latausaikoja.

3.2.2 Suojautuminen

Apachen tilasivulla Slowloris ja POST DOS -hyökkäyksissä käytetyt yhteydet näkyvät lukutilassa (R). Kuviossa 3 on esimerkki, jossa on 50 lukutilassa olevaa yhteyttä. Tästä

ei kuitenkaan suoraan voida sanoa ovatko ne hyökkäystä vai oikeita käyttäjiä. Seuraamalla tilaa voi olla mahdollista selvittää tämä, mutta tämä on epätarkkaa.

PID	Connections		Threads		Async connections		
	total	accepting	busy	idle	writing	keep-alive	closing
1321	25	yes	25	0	0	0	0
1323	24	yes	25	0	0	0	0
1379	1	yes	1	24	0	1	0
Sum	50		51	24	0	1	0

RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRW_____
_____.....
.....

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Kuvio 3. Esimerkki Apachen tilasivusta hyökkäyksen aikana

Hyökkäys voidaan tunnistaa seuraamalla yhteyksiä ja logia. Logi kirjoitetaan vasta yhteyden päätyttyä. Suurella yhteysmäärällä logiin pitäisi tulla jatkuvasti merkintöjä, mutta jos niitä tulee vähän (verrattuna normaalitilanteeseen samalla yhteysmäärällä) voi kyseessä olla Slowloris tai muu hidashyökkäys.

Hyökkäys voidaan havaita myös yhdestä IP-osoitteesta tulevalla suurella yhteysmäärällä mutta tämä voi olla myös internetpalveluntarjoajan NAT tai proxy, jonka kautta sivulla on todellisuudessa useampi käyttäjä. Tämän takia yhteysmäärään pohjautuva esto voi estää myös oikeiden käyttäjien pääsyn sivustolle, jos raja asetetaan liian pieneksi. Jos tarkastellaan vain aikakatkaisuun päättyneitä yhteyksiä hyökkääjä voi ohittaa tunnistuksen mitoittamalla yhteyden oikein, jolloin logiin tulee vain onnistuneita sivulatauksia. (Giobbi 2009.)

Apachen moduulit

Apachelle löytyy useita moduuleja, joita voidaan käyttää Slowloris ja HTTP POST DOS-hyökkäysten torjumiseen.

Apachessa yhteysmäärä rajoitetaan asetustiedostossa, tätä rajoitusta voidaan nostaa, jolloin yhteyksiäkin voi olla enemmän. Tässä on kuitenkin rajana palvelimen resurssit, erityisesti muisti. (Apache HTTP Server Version 2.4 Documentation 2014.)

ModSecurity on avoimen lähdekoodin Web Application Firewall. ModSecurity perustuu sääntöihin, joita on saatavissa sekä ilmaisia että maksullisia. Säännöillä voidaan rajoittaa lukutilassa olevien yhteyksien määrää ja siten rajoittaa hitaiden hyökkäysten vaikutusta. Lisäksi sitä voidaan käyttää myös muiden uhkien, kuten erilaiset XSS- ja SQL-injektiot, torjuntaan ja se on saatavilla myös nginx- ja IIS-webpalvelimille. Sisäänpäin tulevien pyyntöjen lisäksi voidaan tarkistuksia kohdistaa myös ulospäin lähtevään vastaukseen. (ModSecurity 2014.)

Apchen 2.2.15 versiosta alkaen mukana toimitettavalla mod_reqtimeout-moduulilla voidaan asettaa erilliset aikarajat otsikolle ja rungolle. Raja voidaan asettaa myös dynaamisesti, jolloin raja kasvaa määrättyyn rajaan asti vastaanotettujen tavujen mukaan, joka rajoittaa minimi nopeuden. Oletuksena tämä moduuli ei ole kuitenkaan käytössä ennen versiota 2.3.14. (Apache HTTP Server Version 2.4 Documentation 2014.)

Käänteinen välityspalvelin

Yksi tapa suojautua on käyttää käänteisenä välityspalvelimena esimerkiksi nginx-palvelinta tai jotakin muuta palvelinta johon hyökkäys ei toimi. Käänteinen välityspalvelin lähettää yhteydet varsinaiselle palvelimelle, kun ne valmistuvat, jolloin hyökkäys ei vaikuta taustalla olevaan Apache-palvelimeen. Tästä voi olla myös muita etuja, koska näin voidaan toteuttaa myös kuormantasaus, joka mahdollistaa kapasiteetin lisäämisen.

Nginx voi toimia myös palomuurina websovelluksille asentamalla siihen esimerkiksi ModSeucrity-websovelluspalomuuri.

3.3 Verkkosivut / Websovellukset

3.3.1 Yleisesti

Webpalvelimella on usein myös verkkosivut, jotka sisältävät monesti esimerkiksi PHP-skriptillä tuotettua dynaamista sisältöä. Hyökkääjä voi käyttää näitä apuna hyökkäyksen toteutuksessa. Tavallisen staattisen sisällön lataaminen palvelimelta kyselytulvituksella erityisesti, kun käytetään nginx tai muuta hyvin suurelle yhteysmäärälle soveltuvaa webpalvelinta, ei ole tehokasta.

Websovellusten taustalla on useasti tietokantapalvelin, josta haetaan tiedot. Tällöin hyökkääjällä on kolme mahdollista kohdetta: webpalvelin, websovellus ja websovelluksen kautta myös tietokantaan. Tietokantaan hyökkäämisen voi mahdollistaa esimerkiksi sivustolta löytyvä hakutoiminto tai muu raskastoiminto.

Verkkosivut mahdollistavat myös uusia tapoja toteuttaa hyökkäys. Selaimet ovat kehittyneet, ja Javascriptillä voi tehdä monimutkaisiakin sovelluksia, jotka toimivat nopeasti. Tämä mahdollistaa myös palvelunestohyökkäyksen toteuttamisen käyttämällä Javascriptin ja selaimen ominaisuuksia lataamaan sivua taustalla, jolloin käyttäjästä voi tulla hyökkääjä vierailemalla saastuneella sivulla. Saastuneelle sivustolle koodi voi päätyä esimerkiksi mainoskoodiin piilotettuna tai erilaisten tietoturva-aukkojen takia (esimerkiksi XSS). Käyttäjä suorittaa koodia, kunnes selain suljetaan tai sivusto suljetaan, minkä takia hyökkäys on sitä tehokkaampi, mitä pidempään sivulla ollaan. (Clark 2013; Atias 2014)

Websovellusten tietoturva puutteet tai ominaisuudet voivat mahdollistaa niiden käyttämisen hyökkäyksessä. Esimerkiksi WordPressin pingback-toimintoa voidaan käyttää tähän tarkoitukseen. (Moore 2014.)

3.3.2 Kyselytulvitus

Kyselytulvituksessa (Request Flood) ladataan palvelimelta sivuja mahdollisimman nopeasti. Käytännössä kohteena on oltava websovellus tai muu dynaaminen sivu, jotta sen avulla saadaan palvelimen prosessorikäyttöä nostettua. Jos kyseessä on staattinen tai kevyt websovellus, kyselytulvitusta tehokkaampaa on käyttää jotakin muuta hyökkäystä, kuten esimerkiksi Slowloris ja RUDY-hyökkäykset.

Kyselytulvituksen voi toteuttaa Javascriptillä, jolloin se on mahdollista upottaa nettisivulle, jolloin sivun käyttäjistä tulee hyökkääjiä. Jos sivustolla on tarpeeksi käyttäjiä, jotka pysyvät tarpeeksi kauan, tämä toteutuu esimerkiksi sivuilla joilla on videoita. (Clark 2013; Atias 2014)

3.3.3 Keep-Dead

HTTPn Keep-Alive toiminto sallii useamman perättäisen kyselyn samassa TCP-yhteydessä. Ilman Keep-Alivea, jokainen sivuun linkitetty resurssi, jouduttaisiin lataamaan uudessa TCP-yhteydessä. Keep-Aliven kanssa nämä voidaan lähettää samassa yhteydessä, jolloin säästytään uuden TCP-yhteyden avaukselta. (Fielding ym. 1999.)

Keep-Dead on kyselytulvitushyökkäys, jossa käytetään HTTP:n HEAD-metodia, jolla haetaan pelkät otsikot ilman sisältöä. Hyökkääjälle tästä on se etu, että vastaus on pieni, jolloin vastaukset eivät tuki hyökkääjän kaistaa. Keep-Aliven ansioista jokaiselle kyselylle ei taritse luoda uutta yhteyttä, jonka seurauksena kyselyitä voidaan lähettää nopeammin ilman uuden TCP-yhteyden muodostusta. Kohteena voidaan käyttää websovelluksia esim. PHP-skriptiä, jolloin palvelin ajaa sen, joka puolestaan kuluttaa palvelimelta resursseja. (Esrun 2011; Fielding, ym. 1999.)

3.3.4 Suojautuminen

Websovellusten suorituskykyä voidaan parantaa käyttämällä erilaisia välimuisti ratkaisuja.

PHP on tulkattavakieli eli se käännetään aina ajettaessa, tähän voidaan käyttää erilaisia välimuistiratkaisuja, jotka ottavat talteen tulkatun koodin ja käyttävät sitä seuraavilla sivulatauksilla, jolloin sitä ei tarvitse enää kääntää. Tämä ei sovelluksen osalta vaadi erityistoiminpiteitä. (OPCache 2014.)

Lisäksi voidaan käyttää ratkaisuja, jotka tallentavat osan sivusta, esimerkiksi memcached. Tämä edellyttää, että sovellus tukee näitä toimintoja ja on konfigurointi käyttämään niitä. (Memcached 2014.)

Erilaisilla websovellus palomuureilla voidaan rajoittaa sivun latauksia, etenkin raskaita, kuten hakusivua, ei voi ladata useasti samasti IP-osoitteesta. (ModSecurity 2014.)

3.4 TLS/SSL-palvelunestohyökkäys

Internetissä käytettävät protokollat ovat monesti salaamattomia esimerkiksi HTTP. Tämä mahdollistaa tiedon salakuuntelun ja muuttamisen matkalla, tätä varten on kehitetty salattuja protokollia ja TLS (Transport Layer Security) ja sen edeltäjä SSL (Secure Sockets Layer), joiden avulla voidaan varmistaa estää salakuuntelu ja tiedon muuttaminen. (Dierks. & Rescorla. 2008.)

TLS/SSL-yhteys muodostetaan olemassa olevan yhteyden (esimerkiksi TCP-yhteys) päälle eli se ei ole erillinen verkkoprotokolla. SSL-yhteyden sisällä siirretään varsinaisen sovelluskerroksen protokollan (esimerkiksi HTTP-liikennettä). (Dierks & Rescorla. 2008.)

TLS-yhteyden aluksi asiakas lähettää ClientHello-viestin, jossa palvelimelle se kertoo palvelimelle mm. tuetut Chiper Suitet, joka määrää käytettävän avaimen vaihto algoritmin, salausalgoritmin ja todennusalgoritmin) ja numeron, jota käytetään myöhemmin salausavaimen luonnissa. Palvelin vastaa tähän ServerHello-viestillä, jossa se ilmoittaa mm. valitun algoritmin ja satunnaisnumeron, jota käytetään salausavaimen luonnissa. Tämän jälkeen palvelin lähettää asiakkaalle sertifikaatin, jolla varmistutaan, ettei kyseessä ole esimerkiksi MITM-hyökkäys. Tämä jälkeen palvelin lähettää ServerHelloDone-viestin. (Dierks & Rescorla 2008.)

Seuraavaksi suoritetaan avainvaihto, jossa asiakas lähettää algoritmista riippuen palvelimelle avaimen salattuna palvelimen julkisella avaimella. Tämän perusteella ja aiemmin vaihdettujen satunnaisnumeroiden perusteella lasketaan lopullinen avain. Tämän jälkeen asiakas ja sitten palvelin ilmoittavat toisilleen aloittavansa salauksen ChangeCipherSpec-viestillä. (Dierks & Rescorla 2008.)

TLS-yhteyden muodostus on laskennallisesti raskain osa, koska erityisesti avaimen vaihtoon käytetyt algoritmit ovat raskaita. Tämä varmistaa sen, ettei salausta voi purkaa laskennallisesti järjellisessä ajassa. (Dierks & Rescorla 2008.)

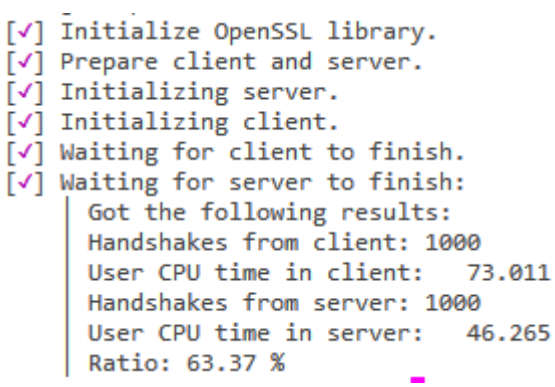
Toinen yleisesti käytetty protokolla on SSH, jota käytetään erityisesti etähallintaan, mutta sitä voidaan käyttää myös yhteyksien tunnelointiin. (Ylönen & Lonvick 2006.)

Koska operaatiot ovat raskaita, tämä mahdollistaa myös palvelunestohyökkäyksen avaamalla palvelimelle yhteyksiä. Riippuen käytetyistä algoritmista palvelimella käytetty laskentateho voi olla yli 20-kertainen asiakkaakseen nähden, mutta joillakin algoritmeilta hyökkääjältä vaadittu laskentateho voi olla suurempi (kuvio 4).

```
[✓] Initialize OpenSSL library.
[✓] Prepare client and server.
[✓] Initializing server.
[✓] Initializing client.
[✓] Waiting for client to finish.
[✓] Waiting for server to finish:
    | Got the following results:
    | Handshakes from client: 1000
    | User CPU time in client:    0.451
    | Handshakes from server: 1000
    | User CPU time in server:    9.754
    | Ratio: 2159.46 %
```

Kuvio 4. Yhteyden muodostukseen kulunut aika RSA-algoritmilla

Jotkin algoritmit voivat olla palvelimen kannalta edullisempia, kuin hyökkääjän kannalta. Esimerkiksi Diffie-Hellman (Kuvio 5) on raskaampi asiakaspuolella kuin palvelin puolella, näissä tapauksissa hyökkääjä voi jättää laskutoimitukset suorittamatta ja lähettää palvelimelle esimerkiksi satunnaisdataa. Palvelin joutuu kuitenkin tarkistamaan ja toteamaan, ettei data ole kelvollista, mutta tämä ei kuitenkaan yhtä tehokasta, kuin laskennallisesti kelvollisilla arvoilla yhteyden muodostus. (Sukalp 2012.)



Kuvio 5. Yhteyden muodostukseen kulunut aika Diffie-Hellman algoritmilla

Jos palvelin sallii asiakkaalta avaimen vaihdon (SSL Renegotation) hyökkääjä voi käyttää tätä hyökkäyksessä, jolloin hyökkääjän ei tarvitse avata uutta TCP-yhteyttä vaan se voi käyttää yhtä yhteyttä, jossa neuvotellaan jatkuvasti uudet salausavaimet. (Sukalp 2012.)

Koska liikenne on salattua palvelinohjelmistolle asti, tämä vaikeuttaa palomuurin toimintaa, koska se ei voi tarkastella yhteyden sisältöä ja hyökkääjän on mahdollista toteuttaa esimerkiksi Slowloris-hyökkäys HTTPS-palvelimeen, vaikka palomuuri pystyisi tunnistamaan sen salaamattomasta. (Sukalp 2012.)

Suojautuminen

TLS/SSL-hyökkäyksien vaikutusta voidaan vähentää rajoittamalla samasta IP-osoitteesta tulevia SSL-yhteyksiä. Tässä on kuitenkin otettava huomioon internetpalveluntarjoajien proxy ja NAT-ratkaisut, joiden takia yhden IP-osoitteen takana voi olla useita käyttäjiä. (Sukalp 2012.; Giobbi 2009.)

Lisäksi voidaan valita tuetut salaustavat siten, että raskaimmat eivät ole käytettävissä ja asiakkaalta tulevia SSL Renegotation-pyyntöjä ei sallita. (Sukalp 2012.)

3.5 DNS

3.5.1 DNS:n toiminta

DNS-palvelua käytetään selvittämään domain-nimen IP-osoite. Käytännössä kaikki palvelut internetissä ovat domain-nimen takana.

Kun käyttäjä avaa verkkosivun (esim. google.fi), täytyy selvittää tähän liittyvä IP-osoite. IP-osoitteen selvittämiseksi lähetetään käytetetään DNS Resolveria. Jos DNS-Resolverilla ei ole välimuistissa domainin tietoja, suoritetaan DNS-rekursio, jossa selvitetään domainista vastaava nimipalvelin.

DNS Resolver voi palauttaa välimuistissa olevan IP-osoitteen. Jos osoitetta ei ole välimuistissa, suoritetaan DNS-rekursio, jossa haetaan nimipalvelin tasokerrallaan.

DNS-rekursio on monivaiheinen operaatio, jossa selvitetään kohdepalvelimen IP-osoite, esimerkiksi google.fi. DNS-rekursio aloitetaan ylimmältä tasolta eli juurinimipalvelin, jolta saadaan seuraavan tason nimipalvelimet (tässä tapauksessa fi-päätteen nimipalvelimet). Tämä voidaan todeta komennolla `dig google.fi @i.root-servers.net. +norec`. (ks. kuvio 6)

```
; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> google.fi @i.root-servers.net. +norec
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35293
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 9, ADDITIONAL: 17

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;google.fi.                     IN      A

;; AUTHORITY SECTION:
fi.                     172800  IN      NS      g.fi.
fi.                     172800  IN      NS      b.fi.
fi.                     172800  IN      NS      i.fi.
fi.                     172800  IN      NS      d.fi.
fi.                     172800  IN      NS      c.fi.
fi.                     172800  IN      NS      f.fi.
fi.                     172800  IN      NS      e.fi.
fi.                     172800  IN      NS      a.fi.
fi.                     172800  IN      NS      h.fi.

;; ADDITIONAL SECTION:
a.fi.                   172800  IN      A       193.166.4.1
a.fi.                   172800  IN      AAAA    2001:708:10:53::53
b.fi.                   172800  IN      A       194.146.106.26
b.fi.                   172800  IN      AAAA    2001:67c:1010:6::53
c.fi.                   172800  IN      A       156.154.100.26
c.fi.                   172800  IN      AAAA    2001:502:ad09::26
d.fi.                   172800  IN      A       77.72.229.253
d.fi.                   172800  IN      AAAA    2a01:3f0:0:302::53
e.fi.                   172800  IN      A       194.0.1.14
e.fi.                   172800  IN      AAAA    2001:678:4::e
f.fi.                   172800  IN      A       87.239.127.198
f.fi.                   172800  IN      AAAA    2a00:13f0:0:3::aaaa
g.fi.                   172800  IN      A       156.154.101.26
h.fi.                   172800  IN      A       156.154.102.26
i.fi.                   172800  IN      A       156.154.103.26
i.fi.                   172800  IN      AAAA    2001:502:2eda::26

;; Query time: 34 msec
;; SERVER: 192.36.148.17#53(192.36.148.17)
;; WHEN: Wed Mar 05 16:59:15 EET 2014
;; MSG SIZE  rcvd: 522
```

Kuvio 6. Juurinimipalvelimen vastaus kyselylle google.fi -domainista

Vastauksen perusteella kyselyä jatketaan jollekin seuraavan tason nimipalvelimelle, esimerkiksi g.fi (156.154.101.26), josta saadaan kuvion 7 mukainen vastaus.

```
; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> google.fi @g.fi. +nore
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25103
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;google.fi.                     IN      A

;; AUTHORITY SECTION:
google.fi.              21600   IN      NS      ns4.google.com.
google.fi.              21600   IN      NS      ns3.google.com.
google.fi.              21600   IN      NS      ns2.google.com.
google.fi.              21600   IN      NS      ns1.google.com.

;; Query time: 62 msec
;; SERVER: 156.154.101.26#53(156.154.101.26)
;; WHEN: Wed Mar 05 17:22:17 EET 2014
;; MSG SIZE  rcvd: 120
```

Kuvio 7. g.fi –nimipalvelimen vastaus kyselylle google.fi -domainista

Seuraavaksi kyselyohjataan jollekin nimipalvelimista, esimerkiksi ns1.google.com. Tässä vaiheessa joudutaan selvittämään nimipalvelimen IP-osoite samalla tavalla, koska sitä ei ollut vastauksessa.

Tässä tapauksessa ns1.google.com on viimeinen nimipalvelin ja antaa varsinaisen vastauksen kyselyyn. (Tässä tapauksessa oletus A eli IPv4-osoite). Muita tietueita ovat esimerkiksi AAAA (IPv6-osoite), NS (nimipalvelin) ja MX (sähköpostipalvelin). ANY-kysely palauttaa kaikki domainin tietueet.

Kentän TTL (time-to-live) kertoo elinajan, jonka jälkeen tieto poistetaan välimuistista ja kysytään uudelleen. Kuviosta 8 nähdään, että Google on valinnut ajaksi 5 minuuttia (300 sekuntia).

```
; <<>> DiG 9.9.3-rpz2+rl.13214.22-P2-Ubuntu-1:9.9.3.dfsg.P2-4ubuntu1.1 <<>> google.fi @ns1.google.com +norec
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 10422
;; flags: qr aa; QUERY: 1, ANSWER: 16, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.fi.                     IN      A

;; ANSWER SECTION:
google.fi.              300     IN      A       193.229.108.55
google.fi.              300     IN      A       193.229.108.45
google.fi.              300     IN      A       193.229.108.49
google.fi.              300     IN      A       193.229.108.25
google.fi.              300     IN      A       193.229.108.30
google.fi.              300     IN      A       193.229.108.24
google.fi.              300     IN      A       193.229.108.59
google.fi.              300     IN      A       193.229.108.39
google.fi.              300     IN      A       193.229.108.44
google.fi.              300     IN      A       193.229.108.34
google.fi.              300     IN      A       193.229.108.40
google.fi.              300     IN      A       193.229.108.50
google.fi.              300     IN      A       193.229.108.20
google.fi.              300     IN      A       193.229.108.29
google.fi.              300     IN      A       193.229.108.54
google.fi.              300     IN      A       193.229.108.35

;; Query time: 72 msec
;; SERVER: 216.239.32.10#53(216.239.32.10)
;; WHEN: Wed Mar 05 17:25:52 EET 2014
;; MSG SIZE  rcvd: 283
```

Kuvio 8. ns1.google.com –nimipalvelimen vastaus kyselylly

Koska DNS-palvelu on kriittinen internetin toimivuudelle, on se myös hyökkääjille houkutteleva kohde. Kohteena voi olla esimerkiksi domain-nimen nimipalvelimet, jolloin hyökkäys vaikuttaa vain siihen ja mahdollisin muihin samoja nimipalvelimia käyttäviin domain-nimiin. (Mantripragada 2013)

TTL-ajalla on vaikutusta palvelunestohyökkäyksien toimintaan. Pitkällä TTL-ajalla vastaus pysyy välimuistissa kauemmin ja muutokset tulevat voimaan hitaammin, jos esimerkiksi IP-osoite joudutaan vaihtamaan palveluestohyökkäyksen seurauksena. Jos palvelunestohyökkäys saa kaadettua DNS-palvelimet pidemmäksi aikaa kuin TTL, DNS-tiedot poistuvat välimuistista ja jos mikään nimipalvelin ei vastaa, ei uusia tietojakaan saada, jolloin käyttäjät eivät pääse sivustolle, vaikka itse webpalvelin olisikin käytettävissä. (Mantripragada 2013)

3.5.2 DNS-peilaushyökkäys

DNS mahdollistaa kyselyt UDP-protokollalla, joka on tilaton eikä vaadi yhteyden muodostusta ja on siten altis lähettäjän väärentämiselle. Vastaus lähetetään UDP:n yli kun se on alle 512 tavua tai käytössä on DNSSEC, jolloin UDP:n yli voidaan siirtää 4 kilotavuun asti. (Damas, Graff & Vixie 2013.)

DNS-peilaushyökkäyksessä hyödynnetään DNS-tuottamia suuria vastauksia, jolloin hyökkääjä saa vahvistettua hyökkäyksen tehoa. Kuten kuviosta 9 nähdään 67 tavun kysely voi tuottaa vastauksen, joka on 529 tavua. Vastaus on tässä tapauksessa kahdeksan kertainen kyselyyn nähden. Hyökkäyksissä voidaan saada jopa 60 kertainen vahvistus, kun käytetään domainia, mikä antaa 4 kilotavun vastauksen maksimi vastauskoko, kun käytetään UDP-protokollaa. 60 kertaisella vahvistuksella, kun hyökkääjä lähettää megabitin kaistalla kyselyjä DNS-palvelin lähettää uhrille vastauksia 60 megabitin vauhdilla. (DNS Amplification Attacks 2013; Mantripragada 2013.)

```
Protocol  Length  Info
DNS         67 Standard query 0x0003  ANY isc.org
DNS        529 Standard query response 0x0003  RRSIG RRSIG
```

Kuvio 9. Esimerkki DNS-kyselystä ja vastauksesta

Suorissa DNS-peilaushyökkäyksissä pyritään tukkimaan ulospäin lähtevä kaista ja siten estämään palvelimelle saapuvat oikeat kyselyt. Kohteen kannalta tämä vastaa DNS-kyselytulvitusta, mutta koska lähdeosoite on väärennetty, vastaukset eivät palaa hyökkääjälle ja siten tuki hyökkääjän omaa kaistaa. (Mantripragada 2013.)

Epäsuorissa DNS-peilaushyökkäyksessä (DNS-Reflection) lähetetään ANY-kyselyjä avoimille DNS-Resolver palvelimille, joissa lähettäjäksi väärennetään hyökkäyksen kohde. ANY-kysely sisältää kaikki domainin-tiedot ja on siten pisin mahdollinen. Hajautetussa DNS-peilaushyökkäyksessä käytetään useita eri DNS-palvelimia, jolloin hyökkäystä saadaan kasvatettua yli yhden palvelimen kapasiteetin. (DNS Amplification Attacks 2013; Mantripragada 2013.)

3.6 NTP-peilaushyökkäys

NTP-peilaushyökkäyksessä hyödynnetään NTP (Network Time Protocol) eli aikapalvelimen monlist-toimintoa, joka palauttaa aikapalvelinta käyttävien koneiden tiedot ja siten mahdollistaa sen käyttämisen hyökkäyksessä ja/tai tukkimaan lähtevä kaista,

jolloin palvelimen muihin palveluihin yhdistäminen hidastuu tai estyy. NTP käytetään UDP-kuljetusprotokollaa. (NTP Amplification Attacks Using CVE-2013-5211 2014.)

Monlist-toiminto palauttaa tiedot (ks. kuvio 10) korkeintaan 600 aikapalvelinta käyttävästä koneista tai aikapalvelimen käyttämistä muista palvelimista. Lista voi olla suuri ja koska monlist-toiminnossa ei ole varmistusta, tämä mahdollistaa väärennetyn IP-osoitteen käytön, jos lähettäjän palveluntarjoaja ei tee pakettien suodatusta. (Prince 2014.)

```
remote address          port local address      count m ver rstr avgint  lstint
===============================================================================
91.189.94.4              123 192.168.1.40          497 4 4    1d0    856     154
62.237.86.238            123 192.168.1.40          499 4 4    1d0    853     291
85.23.204.192            123 192.168.1.40          498 4 4    1d0    854     334
86.60.160.70             123 192.168.1.40          499 4 4    1d0    853     336
83.145.237.222           123 192.168.1.40          500 4 4    1d0    851    1002
```

Kuvio 10. monlist-komennon tuloste

Vaikka kovin monella yrityksellä ei ole käytössä omaa aikapalvelinta, netissä on monia julkisia aikapalvelimia, joita hyökkääjä voi käyttää hyväkseen. NTP-peilaushyökkäyksillä on toteutettu jopa 400 gbps-hyökkäyksiä. Näin suurten hyökkäysten torjunta on erittäin vaikeaa. (Prince 2014.)

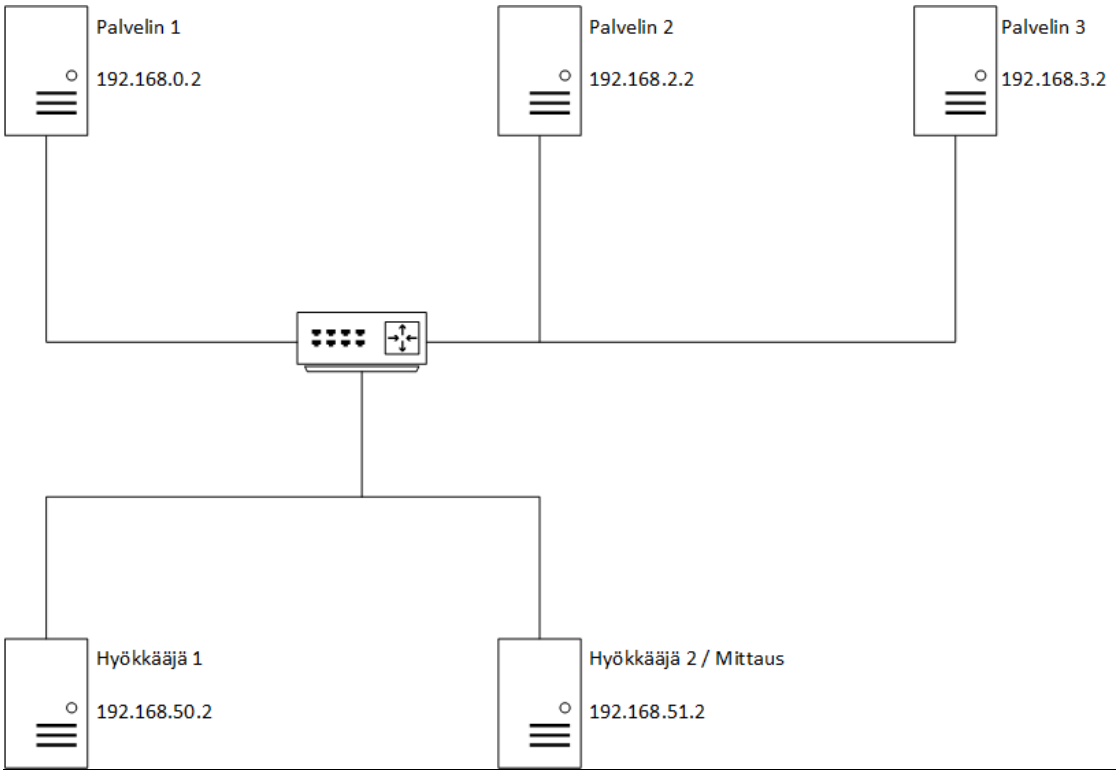
Tältä hyökkäykseltä voidaan suojautua päivittämällä aikapalvelin sellaiseen versioon, joka tukee vain turvallista mrunlist-toimintoa, jossa on lähettäjän varmistus. (Prince 2014.)

4 Toteutus

4.1 Ympäristö

Tehtävänä oli toteuttaa ympäristö, jossa voidaan testata hyökkäysten vaikutusta suojattuun ja suojaamattomaan koneeseen. Toteutuksessa keskityttiin erityisesti HTTP-protokollan hitaisiin hyökkäyksiin ja websovelluksiin kohdistuviin hyökkäyksiin, koska tilaajalla on käytössä DNS- ja NTP-hyökkäyksien testaukseen.

Toteutus tapahtui virtualisoidussa Vmware vCould-ympäristössä. Käyttöjärjestelmänä käytetään Ubuntu Linux 12.04 LTS-versiota. Palvelimet ovat verkossa kuvion 11 mukaisessa topologiassa. Reitittimenä toimii myös Ubuntuu 12.04 LTS-kone. Reitittimessä ei ole palomuuria tai vastaavaa, mikä voisi vaikuttaa tuloksiin.



Kuvio 11. Ympäristön verkkokaavio

Ympäristöön kuuluu kolme kohdetta. Ensimmäinen kohde on altis hyökkäyksille. Toisessa ja kolmannessa kohteessa käytetään eri suojaustapoja. Tämä mahdollistaa toiminnan vertaillun eri hyökkäysten alla kussakin kohteessa.

Kohteina olevilla koneilla on käytössä saman verran laitteistoresursseja, jolloin erot tulevat ainoastaan ohjelmistoista, mikä tekee tuloksista vertailukelpoisia, jolloin niistä nähdään miten eri muutokset vaikuttavat toimivuuteen.

Kohteisiin asennettiin Apache, PHP, MySQL liitteen 1 mukaisesti. Lisäksi Pavelimelle 3 asennettiin nginx liitteen 2 mukaisesti. SSL-testejä varten palvelimille luotiin SSL-serfikaatit liitteen 3 mukaisesti.

Ympäristössä on kaksi hyökkäyskonetta, joilla toisella voidaan suorittaa hyökkäys ja toisella tehdä mahdolliset mittaukset. Näissä on asennettuna graafinen käyttöliittymä, joka helpottaa käyttöä, kun tarvitaan useita samanaikaisia ohjelmia.

Hyökkäykset tapahtuvat Hyökkäys 1-koneelta. Tälle on asennettu SlowHTTPTest (liite 4), Keep-Dead (liite 5), Slowloris (liite 6) ja SSL Squeeze (liite 7). Lisäksi iftop (liite 8), jolla voidaan seurata hyökkäyksessä käytettyä kaistaa. Testitapauksia varten on tehty

skriptit, mikä helpottaa niiden suorittamista (ks. liite 9). Lisäksi käytettävissä on webkäyttöliittymä, jonka kautta voidaan ajaa SlowHTTPTest, tämä pyörii Apachen ja PHP:n päällä.

Mittaus tapahtuu toisella koneella (Hyökkäys 2), jolloin hyökkäys ei vaikuta epäsuorasti tuloksiin. Lisäksi tällä palvelimella on Squid-välityspalvelin, jota voidaan käyttää Hyökkäys 1-koneelta testauksessa.

Taulukko 1. Ympäristön virtuaalikoneet

Kone	IP-osoite	CPU	Muisti
Palvelin 1	192.168.0.2	1 vCPU	4 GB
Palvelin 2	192.168.2.2	1 vCPU	4 GB
Palvelin 3	192.168.3.2	1 vCPU	4 GB
Hyökkäys 1	192.168.50.2	1 vCPU	1 GB
Hyökkäys 2 / Mittaus	192.168.51.2	1 vCPU	1 GB

4.2 Webpalvelimen suojaus

4.2.1 Ympäristö

Webpalvelimeen hyökätään eri hitailla hyökkäyksillä ja verrataan näiden vaikutusta ja toimintaa. Testaustyökaluina käytettään SlowHTTPTest:iä.

Palvelimella 1 on Apache 2.2.22 ja PHP 5.3.10. Ubuntussa on uusille asennuksille käytössä asetukset, joissa on oletuksena mod_reqtimeout. Tämä on poistettu käytöstä komennolla ”a2dismod reqtimeout”. Palvelinta 1 käytetään testitapauksena suojaamattomasta kohteesta, joka mahdollistaa demoamisen muuttamatta asetuksia.

Palvelimella 2 on Apache 2.2.22 ja PHP 5.3.10. Tähän tehdään muutoksia asetuksiin ja yritetään suojata se hyökkäyksiltä.

Palvelimella 3 on nginx käänteisenä välityspalvelimena, jonka takana Apache 2.2.22 samoilla asetuksilla kuin palvelimella 1. Tässä tapauksessa nginx on portissa 80 ja Apache kuuntelee portissa 8080 ja vain paikallisessa IP-osoitteessa (127.0.1.1), koska Apacheen voisi hyökätä, jos se olisi julkisesti saatavilla ja hyökkääjä saisi tietoonsa sen käyttämän portin esimerkiksi porttiskannauksen avulla.

Hyökkäystyökaluina käytetään SlowHTTPTest, Keep-Dead, Slowloris, ja SSLSqueeze -työkaluja. Testitapauksia varten tehdään testiskriptit, jolloin ne voidaan ajaa helposti myöhemmin uudestaan.

Toimivuutta mitataan Hyökkäys 2 palvelimelta Apache Benchmarkilla. Apache Benchmark mittaa sivunlataus aikaa määrätyllä yhteysmäärällä (samanaikaiset sivulataukset) ja latausmäärällä (sivulatauksia yhteensä).

4.2.2 Testi 1 – SlowHTTPTest – Slow Headers - Apache oletusasetuksilla

Ensimmäisessä testissä hyökättiin SlowHTTPTestin Slow Headers-hyökkäyksellä (sama kuin Slowloris) Palvelimelle 1 asennettuun Apacheen, jossa ei ole suojauksia tätä hyökkäystä vastaan. Hyökkäyksessä käytettiin sellaista yhteysmäärää, joka ei riitä täydelliseen palvelunestotilanteeseen ja tutkitaan miten palvelin käyttäytyy tässä tilanteessa ja kuinka se vaikuttaa latausaikoihin.

Latausajan mittaamiseen käytettiin Apache Benchmarkia. Mittaukset suoritettiin toiselta koneelta, jotta SlowHTTPTest ei vaikuttaisi tuloksiin.

Apache Benchmarkin komennot ovat muotoa:

```
ab –r –n [latausmäärä] –c [yhteydet] [kohde]
```

Apache Benchmark tulostaa tilastot latausajoista ruudulle ja tiedot mahdollisesta epäonnistuneista sivulatauksista.

Kuviossa 12 on latausajat perustilanteessa. Normaalitilanteessa staattisen html-sivun lataus kestää 16 ms – 219 ms, keskiarvon ollessa 47 ms.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    3   3.9      1      11
Processing:     5   44  33.8     39     214
Waiting:        5   44  33.9     39     214
Total:         16   47  33.7     41     219
```

Kuvio 12. Latausajat perustilanteessa

SlowHTTPTestin komennot ovat muotoa:

```
slowhttptest –g –o [nimi] –c [yhteysmäärä] –l [kesto] –u
[kohde]
```

Tässä –g –o [nimi] tuottaa tilastot [nimi].html tiedostoon. Yhteysmäärä määrittelee käytettävän yhteysmäärän ja kesto määrittelee maksimiajan, jos palvelin katkaisee yhteydet, testi päättyy aiemmin.

Testiskriptin 1 (ks. liite 1) avulla tämä voidaan ajaa:

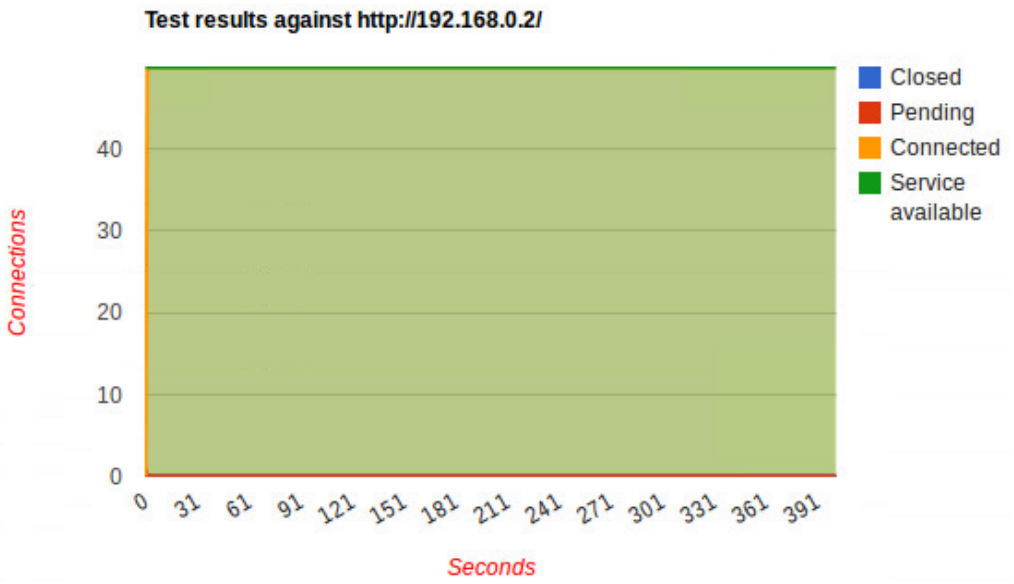
```
/testit/testi1.sh [nimi] [kohde] [yhteysmäärä] [kesto]
```

Ensimmäinen testi ajetaan komennolla:

```
/testit/testi1.sh testi1 http://192.168.0.2/ 50 400
```

Tämä komento luo 50 yhteyttä ja kestää maksimissaan 400 sekuntia, jonka jälkeen yhteydet katkaistaan. Tilastot kirjoitetaan testi1.html-tiedostoon.

Kuviossa 13 tulokset, joista huomataan, ettei palvelin katkaise yhteyksiä testin aikana (oranssi käyrä). Tämä paljastaa palvelimen olevan haavoittuva hitaille hyökkäyksille.



Kuvio 13. SlowHTTPTest 50 yhteydellä

Kuten kuviosta 14 nähdään, palvelin pystyy vastaamaan lähes samassa ajassa vaikka hyökkääjällä on varattuna 50 yhteyttä.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    6   8.4      1      24
Processing:    11   39  26.2     39     213
Waiting:       11   39  26.2     38     213
Total:         31   46  25.3     42     224
```

Kuvio 14. Latausajat testin aikana

4.2.3 Testi 2 – Maksimi yhteysmäärän määrittäminen SlowHTTPTestillä

SlowHTTPTestissä pystytään määrittelemään kuinka nopeasti yhteyksiä muodostetaan parametrillä –r. Tarpeeksi pienellä arvolla saadaan kuvio, jossa näkyy kuinka monen yhteyden jälkeen palvelin lakkaa vastaamasta.

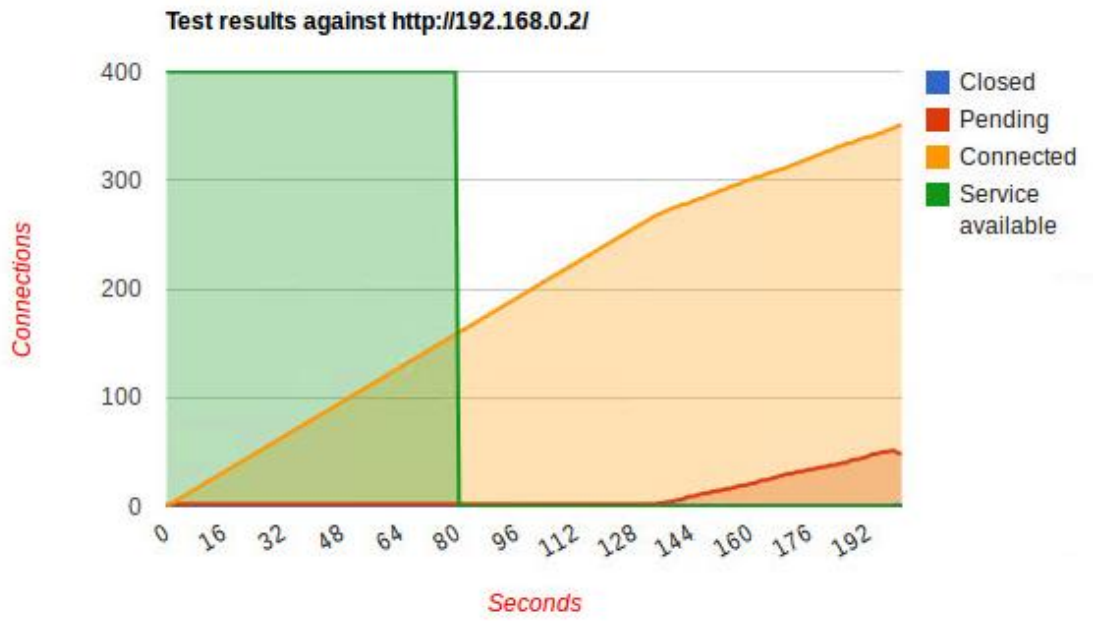
Tätä käytetään määrittämään Pavelimen 1 kapasiteetti eli kuinka monta yhteyttä siihen voi olla samanaikaisesti. Tässä testissä oletetaan, että palvelin ei rajoita yhdestä IP-osoitteesta tulevia yhteyksiä. Tässä tilanteessa palvelin lakkaisi vastaamasta vain hyökkääjälle ja hyökkääjän tarvitsee käyttää hajautettua palvelunestohyökkäystä.

Testiscripti ajetaan komennolla (ks. liite 1):

```
/testit/testi2.sh testi2 http://192.168.0.2/ 400
```

Testiscripti määrittelee testin pituuden automaattisesti yhteismäärälle sopivaksi. Yhteyksiä määritellään muodostettavaksi kaksi sekunnissa, jolloin testin pituus on sekunti per kaksi yhteyttä.

Kuviosta 15 nähdään, että palvelin lakkaa vastaamasta (vihreä käyrä) testiyhteyksiin n. 150 yhteyden (oranssi käyrä) kohdalla.



Kuvio 15. Maksimiyhteysmäärän selvittäminen

Kuviossa 16 top-ohjelman tuloste samalta ajalta. Tässä nähdään Apachen prosessi kohtainen CPU- ja muistikäyttö ja koko palvelimen CPU- ja muistikäyttö. Tästä voidaan todeta, ettei hyökkäys aiheuta suurta CPU- tai muistikuormaa, vaan ainoastaan täyttää webpalvelimen (tässä tapauksessa Apache) yhteyspaikat, johon hitaissa hyökkäyksissä palvelunesto perustuu.

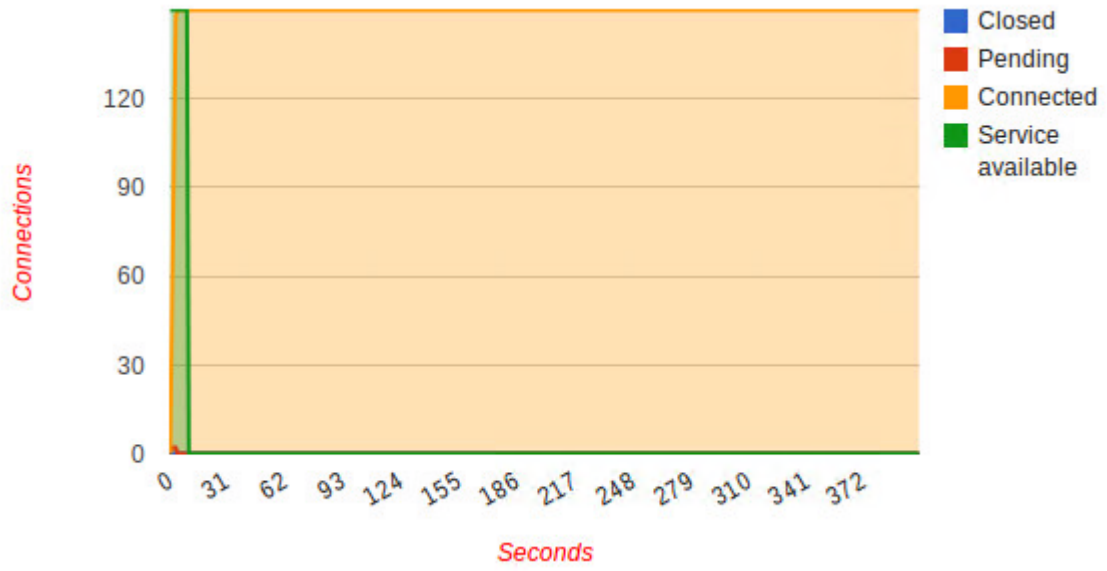
```
top - 01:45:46 up 2 days,  5:50,  1 user,  load average: 0.01, 0.02, 0.05
Tasks: 218 total,   1 running, 217 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1019080k total,   517164k used,   501916k free,    38328k buffers
Swap:  1048572k total,        0k used,  1048572k free,   341616k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 4872 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 4874 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 4884 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5035 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5042 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5043 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5044 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5045 www-data  20   0  101m 5416 1068 S  0.0  0.5   0:00.00 apache2
 5046 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5047 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5049 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5070 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5076 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5077 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5095 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5100 www-data  20   0  101m 5472 1120 S  0.0  0.5   0:00.00 apache2
 5102 www-data  20   0  101m 5244  916 S  0.0  0.5   0:00.00 apache2
 5113 www-data  20   0  101m 5472 1120 S  0.0  0.5   0:00.00 apache2
```

Kuvio 16. Apachen CPU- ja muistikäyttö

Tulos voidaan varmistaa testaamalla se samaan tapaan kuin testissä 1. Tämä tapahtui komennolla:

```
/testit/testi1.sh testi2b http://192.168.0.2/ 150 400
```

Kuviossa 17 näkyvistä tuloksista voidaan todeta, että näillä asetuksilla 150 yhteyttä riittää palvelunestohyökkäykseen (vihreä käyrä). Oranssista käyrästä nähdään, ettei palvelin katkaise yhteyksiä, tämä mahdollistaa määräämättömän pitkän palvelunestohyökkäyksen.



Kuvio 17. SlowHTTPTest 150 yhteydellä Palvelimeen 1

4.2.4 Testi 3 – mod_reqtimeout

Mod_ReqTimeout-moduulilla voidaan rajoittaa otsikon ja mahdollisen rungon sisällön lukemiseen käytettävän ajan.

Ubuntussa mod_reqtimeout-moduuli voidaan ottaa käyttöön komennolla:

```
ae2enmod reqtimeout
```

Ubuntun oletusasetukset (ks. kuvio 18) moduulille sallivat otsikoille 20 sekuntia, jota voidaan kasvattaa 40 sekuntiin, kun lähetysnopeus on 500 tavua/s. Itse rungolle ei ole maksimi aikaa, vaan ainoastaan sama minimi nopeus kuin otsikoissa.

```
# Wait max 20 seconds for the first byte of the request line+headers
# From then, require a minimum data rate of 500 bytes/s, but don't
# wait longer than 40 seconds in total.
# Note: Lower timeouts may make sense on non-ssl virtual hosts but can
# cause problem with ssl enabled virtual hosts: This timeout includes
# the time a browser may need to fetch the CRL for the certificate. If
# the CRL server is not reachable, it may take more than 10 seconds
# until the browser gives up.
RequestReadTimeout header=20-40,minrate=500

# Wait max 10 seconds for the first byte of the request body (if any)
# From then, require a minimum data rate of 500 bytes/s
RequestReadTimeout body=10,minrate=500
```

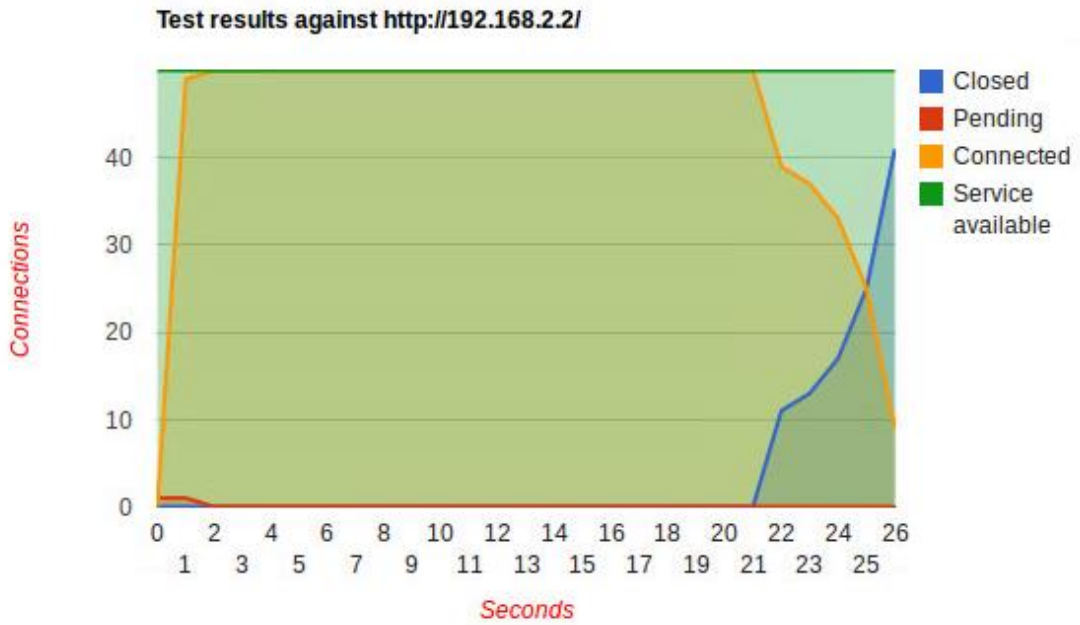
Kuvio 18. Ubuntun oletusasetukset mod_reqtimeout moduulille

Asetukset voidaan testata komennolla:

```
/testit/testi1.sh testi3 http://192.168.2.2/ 50 400
```

Tämä suorittaa SlowHTTPTest:in 50 yhteydellä ja 400 sekunnin maksimi kestolla ja tilastot kirjoitetaan testi3.html-tiedostoon.

Kuviosta 19 nähdään, että palvelin alkaa katkaisemaan yhteyksiä n. 21 sekunnin kohdalla, joten asetusten voidaan todeta toimivan.

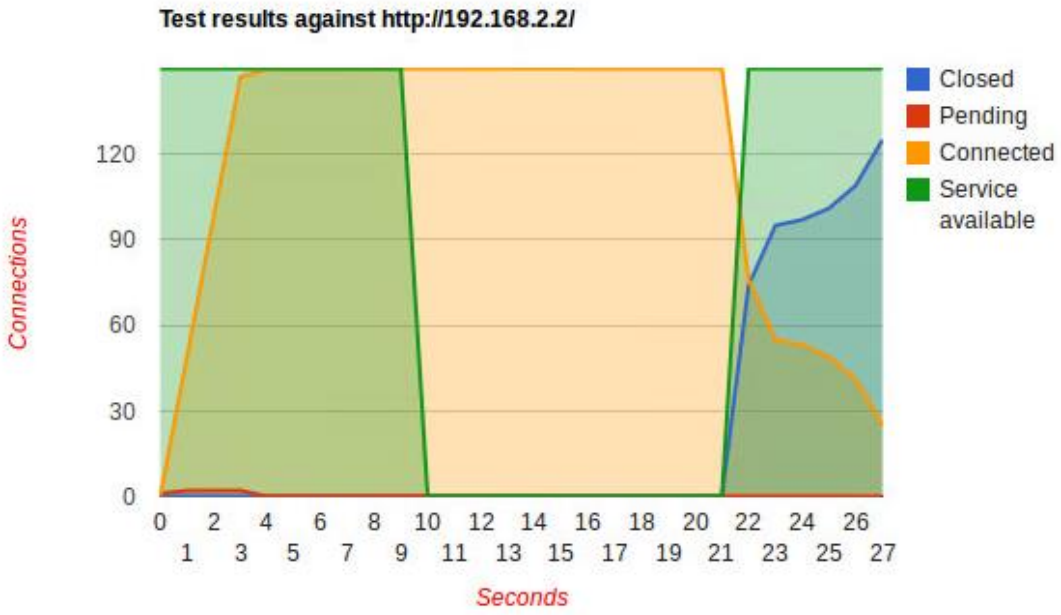


Kuvio 19. SlowHTTPTest 50 yhteydellä Palvelimen 2, kun käytössä mod_reqtimeout

Ilman suojausta 150 yhteyttä riitti estämään palvelimelle pääsyyn. Kuten kuviosta 20 nähdään, näillä asetuksilla ei ole tähän vaikutusta ja se riittää edelleen estämään toiminnan. Toisin kuin aiemmin palvelin katkaisee yhteydet ja sen jälkeen se on taas

toiminnassa eli hyökkääjä ei pysty pitämään yhteyksiä määräämättömän kauan. Hyökkääjä voi kuitenkin toistaa hyökkäystä, jolloin sivuston toiminta häiriintyy.

```
/testit/testi1.sh testi3 http://192.168.2.2/ 150 400
```



Kuvio 20. SlowHTTPTest 150 yhteydellä Palvelimelle 2, kun käytössä mod_reqtimeout

Kuten kuviosta 21 nähdään, sivunlataukset toimivat, mutta ne ovat erittäin hitaita. Sivulatauksissa kestää noin 14 sekuntia eli ne joutuvat odottamaan siihen asti kun palvelin alkaa tiputtamaan yhteyksiä. Tämä on sen verran pitkä aika, että käyttäjä voi luulla sivuston olevan alhaalla.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.2      1       1
Processing: 14523 14533   7.0  14535   14541
Waiting:    14523 14533   7.0  14535   14541
Total:      14524 14533   7.0  14535   14541
```

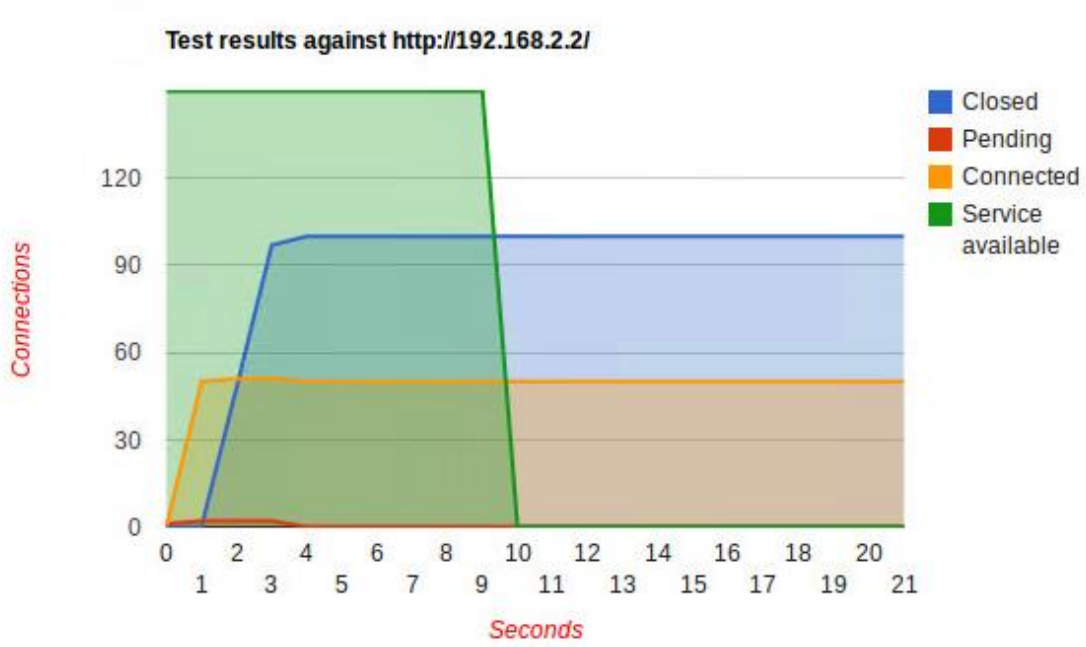
Kuvio 21. Vastausaika hyökkäyksen aikana.

4.2.5 Testi 4 – ModSecurity

ModSecurityä voidaan käyttää rajoittamaan yhtäaikaisten yhteyksien määrää. Normaalisti selaimet käyttävät vain muutamaa yhteyttä, mutta rajoituksissa on otettava huomioon myös NAT- ja välityspalvelin ratkaisut, jolloin yhden IP-osoitteen takana voi olla useita käyttäjiä. ModSecurity asennettiin Palvelimelle 2 liitteen 10 mukaisesti.

Lukutilassa olevien yhteyksienmäärän rajoittamiseen käytetään komentoa SecReadStateLimit, jolle annetaan parametrina sallittava yhteysmäärä. Esimerkiksi ”SecReadStateLimit 50” rajoittaa samasta IP-osoitteesta tulevat yhteydet 50:neen ja katkaisee loput.

Toimivuus voidaan todeta ottamalla sääntö käyttöön ja testaamalla se SlowHTTPTestillä. Kuten kuviosta 22, nähdään palvelin katkaiseen yhteyksiä heti (sininen viiva). Eron voi huomata vertaamalla esimerkiksi testiin 3. Palvelin lakkaa vastaamasta hyökkääjän pyyntöihin, jolloin SlowHTTPTest luulee palvelun kaatuneen (vihreä viiva).



Kuvio 22. SlowHTTPTest 150 yhteydellä

Palvelun normaali toimivuus muille käyttäjille voidaan kuitenkin todeta testaamalla se toiselta koneelta Apache Benchmarkilla (ks. kuvio 23).

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.2      0       6
Processing:     2   21   6.3     21      94
Waiting:        2   20   5.7     21      94
Total:          5   22   5.7     22      94
```

Kuvio 23. Apache Benchmark Hyökkäys 2-koneelta.

Säännön toimivuus voidaan todeta myös Apachen virhelogista, josta löytyy tiedot ModSecurityn katkaisemista yhteyksistä. Kuten kuviosta 24 nähdään, testin aikana tuli logimerkintöjä katkaistuista yhteyksistä.

```
Too many threads [51] of 50 allowed in READ state from 192.168.50.2 - Possible DoS Consumption Attack [Rejected]
Too many threads [51] of 50 allowed in READ state from 192.168.50.2 - Possible DoS Consumption Attack [Rejected]
Too many threads [51] of 50 allowed in READ state from 192.168.50.2 - Possible DoS Consumption Attack [Rejected]
Too many threads [51] of 50 allowed in READ state from 192.168.50.2 - Possible DoS Consumption Attack [Rejected]
```

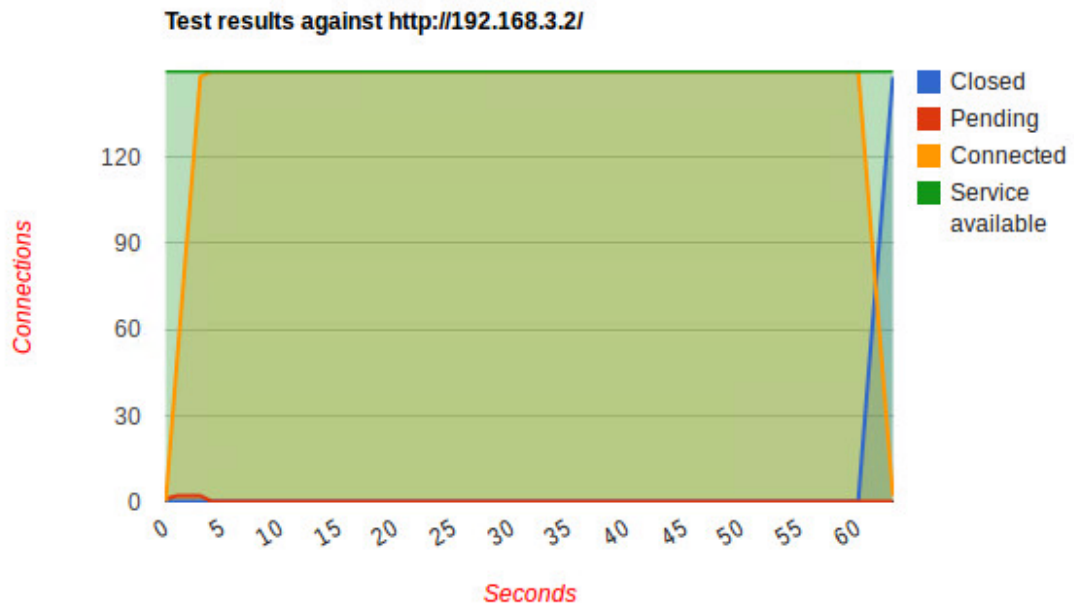
Kuvio 24. ModSecurityn estämät yhteydet Apachen logissa

4.2.6 Testi 5 – nginx käänteisenä välityspalvelimena

Palvelimella 3 on käytössä nginx, joka toimii käänteisenä välityspalvelimena Apachelle, joka on asennettua samalla koneelle. Apache on samoilla asetuksilla kuin Palvelin 1 eli siinä ei ole suojauksia hyökkäyksiä vastaan.

Kuviosta 25 huomataan, ettei 150 yhteyttä riitä estämään toimintaa kuten Apachen kanssa, vaan palvelu pysyy toiminnassa koko testin ajan. Lisäksi voidaan todeta, että nginx oletusasetuksilla katkaisee yhteydet 60 sekunnin jälkeen (sininen viiva). Tästä voidaan todeta myös se että taustalla oleva Apache ei rajoita palvelun toimivuutta.

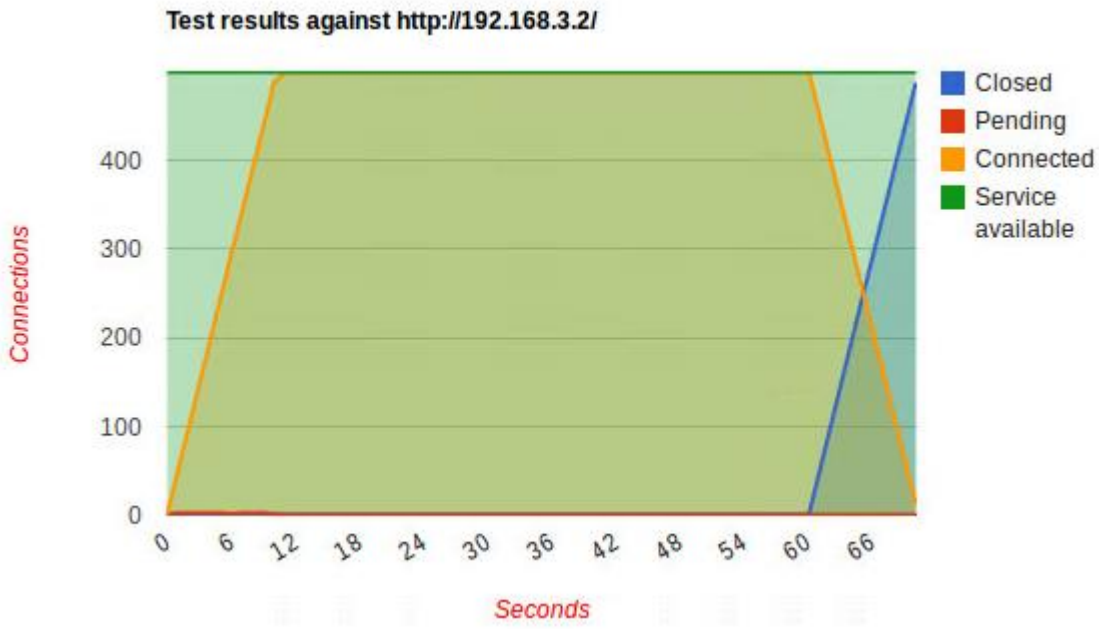
```
/testit/testi1.sh testi5a http://192.168.3.2/ 150 400
```



Kuvio 25. SlowHTTPTest nginx-palvelimeen 150 yhteydellä

Kuten kuviosta 26 nähdään, hyökkäyksen kasvattaminen 500 yhteyteen ei riitä luomaan palvelunestotilannetta. Testaukseen käytettiin komentoa:

```
/testit/testi1.sh testi5b http://192.168.3.2/ 500 400
```



Kuvio 26. SlowHTTPTest nginx-palvelimeen 500 yhteydellä.

Kuten kuviosta 27 nähdään, samaan aikaan palvelin pystyy vastaamaan käyttäjien pyyntöihin ongelmitta.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    5   3.9      5      13
Processing:     7   42   8.9     43      83
Waiting:        7   41   8.9     42      83
Total:         18   47   8.1     47      94
```

Kuvio 27. Vastausaika hyökkäyksen aikana 100 käyttäjällä.

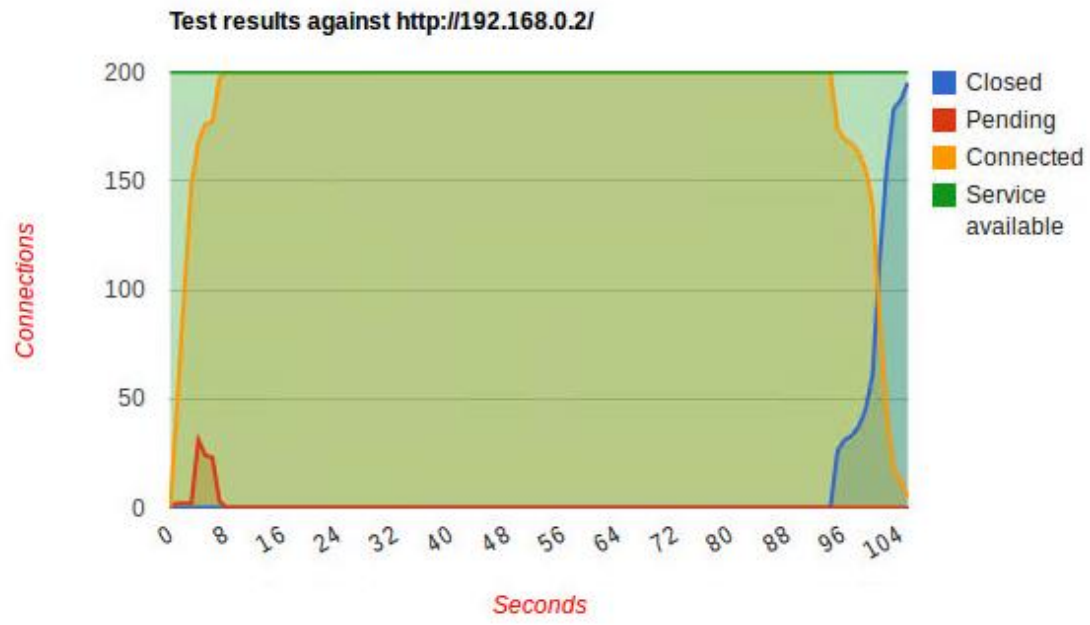
4.2.7 Testi 6 – SlowHTTPTest – Slow Read

SlowHTTPTestin Slow Read-hyökkäyksessä palvelimelta luetaan hitaasti tiedostoa. Koska tässä lähetetään valmis kysely, hyökkäykseen ei vaikuta edellisessä testeissä tehdyt muutokset. Tämä hyökkäys edellyttää, että ladattava sivu on sellainen, että se ei mahdu käyttöjärjestelmän TCP-lähetyspuskuriin, jolloin webpalvelin (tässä tapauksessa Apache) voisi siirtyä seuraavan yhteyden käsittelyyn. Testaukseen käytettiin komentoa:

```
/testit/testi3.sh slowread1 http://192.168.0.2/ 200 400
```

Komento ajaa SlowHTTPTestin Slow Read-tilassa Palvelimelle 1. 200 yhteydellä ja 400 sekunnin maksimi kestolla.

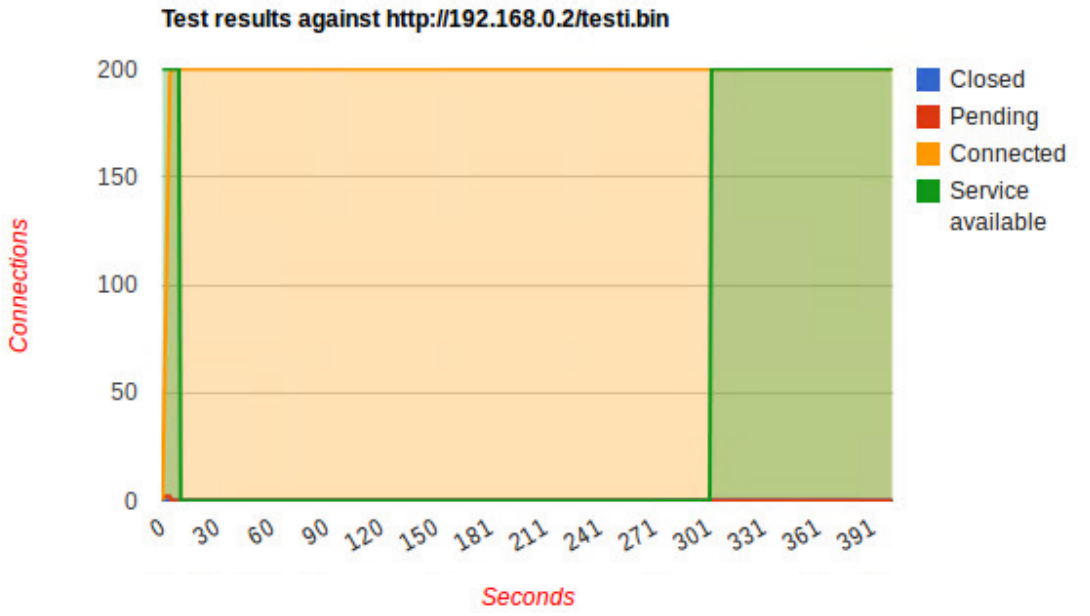
Kuten kuviosta 28 huomataan, tämän seurauksena 200 yhteyttä ei ole riittävä tällä estämään palvelimen toimintaa.



Kuvio 28. Slow Read-hyökkäys 177 tavun sivuun Palvelimella 1

Kuten kuviosta 29 nähdään, isompaa tiedostoa käyttäessä (tässä tapauksessa testi.bin – 500 kilotavua) palvelunesto onnistuu. Yhden isomman tiedoston sijaan hyökkääjä voisi tehdä peräkkäisiä kyselyitä pienemmistä tiedostoista. (ks. kuvio 29).

```
/testit/testi3.sh slowread2 http://192.168.0.2/testi.bin
200 400
```

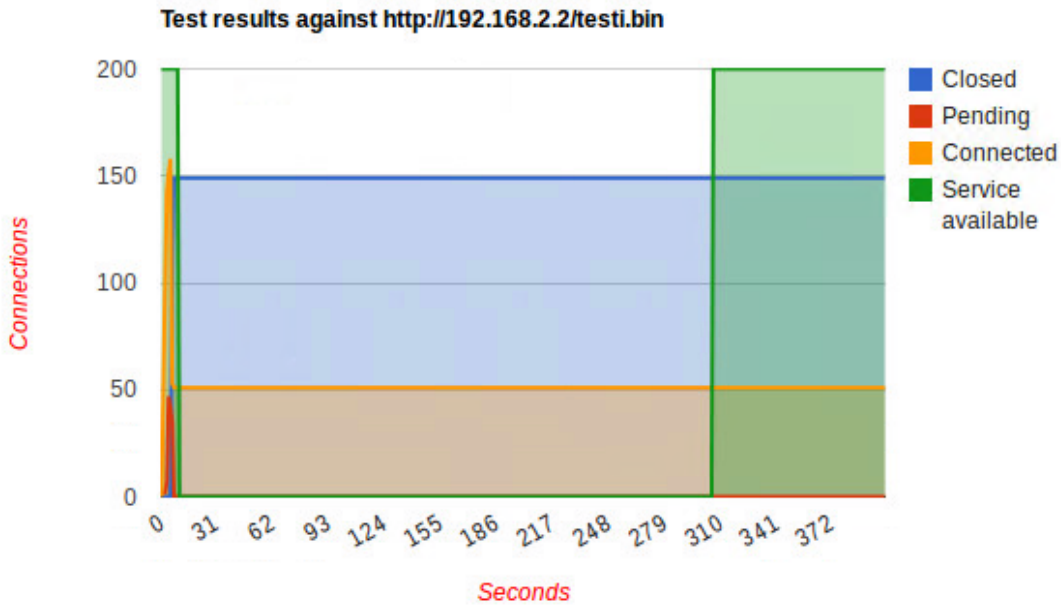


Kuvio 29. Slow Read-hyökkäys 500 kilotavun tiedostoon Palvelimella 1

4.2.8 Testi 7 – SlowHTTPTest – Slow Read – Suojautuminen

ModSecurity tarjoaa lukutilassa olevien yhteyksien rajoittamisen lisäksi mahdollisuuden rajoittaa kirjoitustilassa olevien yhteyksien määrän. Se onnistuu SecWriteStateLimit, jolle annetaan parametrina sallittava yhteysmäärä. Esimerkiksi ”SecWriteStateLimit 50”.

Kuten kuviosta 30 huomataan, palvelin katkaisee yhteydet (sininen käyrä) ja palvelin vaikuttaa olevan alhaalla hyökkääjän kannalta. Se johtuu kuitenkin palvelimen rajoituksista, jotka estävät myös testaukseen käytetyt yhteydet.



Kuvio 30. Slow Read-hyökkäys 200 yhteydellä suojattuun kohteeseen

Palvelimen toimivuus voidaan todeta testaamalla se toiselta palvelimelta, jolloin yhteysmäärän rajoitus ei estä testausta. Testaus tapahtui 50 yhteydellä ja 500 pyynnöllä ja palvelin pystyi vastaamaan pyyntöihin normaalilla nopeudella, kuten kuviosta 31 nähdään.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.2      0       6
Processing:     3   20   3.7     20      30
Waiting:        2   20   3.7     20      30
Total:          7   20   3.0     21      32
```

Kuvio 31. Vastausaika hyökkäyksen aikana

4.3 Websovellusten suojaus

4.3.1 Ympäristö

Testattavaksi websovellukseksi valikoitui Wordpress, joka on suosittu blogi- ja sisällönhallintajärjestelmä.

Websovellukset asennetaan ja testataan samoin kuin webpalvelimen eli Palvelimelle 1 asennetaan Wordpress oletusasetuksilla, ilman mitään erityisiä suojaustoiminpiteitä. Tämä toimii esimerkkinä suojaamattomasta kohteesta ja muodostaa samalla vertailukohdan suojausten toimivuudelle.

Palvelimelle 2 asennetaan Wordpressiin oletusasetuksilla ja sen lisäksi APC koodi- ja objektivälimuisti liitteen 7 mukaisesti. APC-Stats avulla voidaan seurata välimuistintoimivuutta.

Palvelimelle 3 asennetaan Wordpress, APC koodivälimuisti ja nginx konfiguroidaan tallentamaan välimuistiin sivut ja käyttämään niitä mahdollisuuksien mukaan.

Ympäristön koneilta Wordpress on saatavissa selaimen kautta taulukon 2 mukaisissa osoitteissa ja näitä osoitteita käytetään myös testauksessa.

Taulukko 2. Wordpressin osoitteet

Palvelin	Osoite
Palvelin 1	http://192.168.0.2/wordpresss/
Palvelin 2	http://192.168.2.2/wordpresss/
Palvelin 3	http://192.168.3.2/wordpresss/

4.3.2 Testi 1 – Vastausaika ilman hyökkäystä

Normaalitilanteessa Wordpressin vastausaika on noin 930 ms Apache Benchmarkilla viidellä yhtäaikaisella käyttäjällä ja 50 pyynnöllä, kuten kuviosta 32 vuodaan todeta.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       3
Processing:   878  929  19.5    932     965
Waiting:      878  929  19.1    932     965
Total:        879  930  19.6    932     967
```

Kuvio 32. Vastausaika Wordpress-etusivu.

Staattisella html-sivulla (kuvio 33) ja yksinkertaisella PHP-sivulla, joka ei käytä tietokantaa (kuvio 34), vastausaika on huomattavasti pienempi.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:     1    1   0.5      1       3
Waiting:        0    1   0.5      1       3
Total:          1    2   0.5      2       3
```

Kuvio 33. Vastausaika staattisella sivulla.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       2
Processing:     2    8   4.4      7      23
Waiting:        1    5   2.0      6       9
Total:          3    9   4.4      8      23
```

Kuvio 34. Vastausaika phpinfo-sivulla.

4.3.3 Testi 2 - Keep-Dead Wordpress palvelimella 1

Keep-Dead hyökkäyksellä hyökätään Wordpressin hakulomakkeeseen, joka on todennäköisesti yksi raskaimmista osista. Wordpress on suosittu blogi- ja sisällönhallinta järjestelmä. Toimivuutta hyökkäyksen aikana mitataan Apache Benchmarkilla sekä staattiseen sisältöön, että Wordpressiin.

Keep-Dead oletusasetuksilla tulvittaa yhteyksiä mahdollisimman nopeasti. Testeissä alle minuutin hyökkäys jumitti Apachen lähes puoleksi tunniksi. Tuloksena tämä on pahin mahdollinen, koska ympäristössä laitteiden välillä ei synny viivettä. Realistisempi saadaan käyttämällä pientä viivettä yhteyksin ja pyyntöjen välissä. Internetissä hyökkääjän viive riippuu sijainnista. (ks. kuvio 35).

```
top - 02:26:48 up 4 days,  9:07,  1 user,  load average: 147.07, 81.51, 33.46
Tasks: 219 total, 151 running,  68 sleeping,   0 stopped,   0 zombie
Cpu(s): 97.4%us,  2.6%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4049592k total,  3826580k used,   223012k free,      468k buffers
Swap:  1048572k total,    98996k used,   949576k free,    13004k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7465 www-data  20   0  211m  28m 2916 R  1.0  0.7   0:17.32 apache2
11745 www-data  20   0  208m  27m 3140 R  1.0  0.7   0:01.73 apache2
11758 www-data  20   0  210m  28m 3140 R  1.0  0.7   0:01.60 apache2
11760 www-data  20   0  210m  28m 3140 R  1.0  0.7   0:01.59 apache2
```

Kuvio 35. top-ohjelman tuloste Keep-Dead hyökkäyksessä Wordpressiin

Keep-Dead hyökkäys staattiseen sisältöön samalle palvelimelle ei aiheuta palvelunestotilannetta, vaan palvelin pystyy edelleen vastaamaan. Kuten kuviosta 36 nähdään, CPU-käyttö on kuitenkin normaalia korkeampi.

```
top - 16:20:14 up 5 days, 23:00,  1 user,  load average: 0.69, 0.28, 0.13
Tasks:  79 total,   2 running,  77 sleeping,   0 stopped,   0 zombie
Cpu(s): 43.4%us, 17.7%sy,  0.0%ni, 33.2%id,  0.0%wa,  0.0%hi,  5.7%si,  0.0%st
Mem:   4049592k total,   846668k used,  3202924k free,    26736k buffers
Swap:  1048572k total,    52748k used,   995824k free,   429860k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
19858 www-data  20   0  210m  31m 3228 S  6.3  0.8   0:16.14 apache2
19859 www-data  20   0  210m  31m 3220 S  6.3  0.8   0:16.12 apache2
19868 www-data  20   0  210m  31m 3220 S  6.3  0.8   0:16.13 apache2
19872 www-data  20   0  208m  29m 3220 S  6.3  0.8   0:16.16 apache2
19873 www-data  20   0  210m  31m 3228 S  6.3  0.8   0:16.13 apache2
19862 www-data  20   0  210m  31m 3228 S  6.0  0.8   0:16.14 apache2
19867 www-data  20   0  210m  31m 3220 R  6.0  0.8   0:16.18 apache2
19869 www-data  20   0  210m  31m 3228 S  6.0  0.8   0:16.16 apache2
19874 www-data  20   0  210m  31m 3232 S  6.0  0.8   0:16.11 apache2
19864 www-data  20   0  210m  31m 3220 S  5.7  0.8   0:16.14 apache2
```

Kuvio 36. top-ohjelman tuloste Keep-Dead hyökkäyksessä

Kuten kuviosta 37 voidaan todeta hyökkäys ei vaikuta staattisen html-sivun vastaus aikaan vaan vastausaika pysyy muutamassa millisekunnissa (verkkoviive).

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       2
Processing:     1    1   0.5      1       3
Waiting:        0    1   0.5      1       3
Total:          1    2   0.5      2       3
```

Kuvio 37. Staattisen html-sivun vastausaika hyökkäyksen aikana.

Keep-Dead suoritettiin komennolla:

```
php /testit/keepdead/palvelinX.php
```

Jossa X on palvelimen numero yhdestä kolmeen. Jokaiselle kohteelle on oma tiedosto, koska Keep-Deadin asetukset ovat suoraan lähdekoodissa eikä se tue parametreja. (ks. liite 9.)

Kuten kuviosta 38 nähdään, käytettäessä Keep-Deadin suositeltuja viiveitä (0.01 sekuntia pyyntöjen välissä ja 0.5 sekuntia yhteyksien välissä) hyökkäys toimii edelleen

ja saa aiheutettua suuren CPU-kuorman (Esrun 2011, lähdekoodi). Jumittumiseen menee kauemmin, kuin ilman viiveitä.

```
top - 01:05:50 up 5 days,  7:46,  1 user,  load average: 150.00, 140.68, 113.20
Tasks: 219 total, 151 running,  68 sleeping,   0 stopped,   0 zombie
Cpu(s): 97.7%us,  2.3%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4049592k total,  3905536k used,   144056k free,      712k buffers
Swap:  1048572k total,   166192k used,   882380k free,     8852k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  877 mysql     20   0  897m  30m 2488 S  1.0  0.8   3:40.01 mysqld
17427 www-data  20   0  210m  28m 2712 R  1.0  0.7   0:25.79 apache2
18738 www-data  20   0  208m  26m 2752 R  1.0  0.7   0:06.73 apache2
18765 www-data  20   0  209m  26m 2752 R  1.0  0.7   0:06.20 apache2
18829 www-data  20   0  208m  26m 2752 R  1.0  0.7   0:05.45 apache2
```

Kuvio 38. top-ohjelman tuloste hyökkäyksessä, kun viiveet ovat käytössä.

Kuten kuviosta 39 huomataan, tämä hyökkäys ei tarvitse paljoa kaistaa, näillä asetuksilla n. 131 Kb/s lähtevää ja vain 14 Kb/s saapuvaa. Saapuvaan liikenteeseen vaikuttaa sivun otsikoiden koko.

```
attack.local              => 192.168.0.2                  131kb   122kb   126kb
                          <=                              14.3kb  17.2kb  13.6kb

TX:             cum:   2.31MB   peak:    142kb  rates:   131kb   122kb   126kb
RX:                    269kB             46.0kb           14.3kb  17.2kb  13.6kb
TOTAL:                 2.57MB            188kb            146kb   139kb   139kb
```

Kuvio 39. Kaistankäyttö hyökkäyksen aikana

Apachen logissa Keep-Dead hyökkäys näkyy suurena määränä HEAD-kyselyjä. Ensimmäinen kysely on Keep-Deadin palvelimen Keep-Alive tuen testaamiseksi. (ks. kuvio 40.)

```
192.168.50.2 - - [27/Apr/2014:23:59:03 +0300] "HEAD / HTTP/1.1" 200 313 "-" "Mozilla/
192.168.50.2 - - [27/Apr/2014:23:59:03 +0300] "HEAD /wordpress/?s=kqqbqwvh HTTP/1.1"
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=khea HTTP/1.1" 200
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=ptjrmacz HTTP/1.1"
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=udvfm HTTP/1.1" 200
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=layxjwkf HTTP/1.1"
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=watffaoyw HTTP/1.1"
192.168.50.2 - - [27/Apr/2014:23:59:04 +0300] "HEAD /wordpress/?s=auwzuzle HTTP/1.1"
```

Kuvio 40. Keep-Dead hyökkäys Apachen logissa

4.3.4 Testi 3 – Keep-Dead Wordpress palvelimella 2

Palvelimelle 2 on asennettu APC (Alternative PHP Cache) -moduuli liitteen 11 mukaisesti. APC tallentaa tulkitun koodin välimuistiin ensimmäisellä latauskerralla, jolloin sitä voidaan hyödyntää seuraavilla latauskerroilla ja säästää tulkintaan kulunut aika. APCn asennuksen toimivuudesta voidaan varmistua phpinfo-sivulta, josta löytyy APC:n asetukset ja versiotiedot kuten kuviossa 41.

apc

APC Support	enabled
Version	3.1.7
APC Debugging	Disabled
MMAP Support	Enabled
MMAP File Mask	*no value*
Locking type	pthread mutex Locks
Serialization Support	php
Revision	$Revision: 307215 $
Build Date	May 2 2011 19:00:42

Kuvio 41. APC phpinfo-sivulla.

Kuviosta 42 nähdään, että APCn kanssa vastausaika on huomattavasti pienempi kuin ilman testissä 1.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:   224  267  12.7    267     298
Waiting:      223  267  12.2    267     291
Total:        224  268  12.8    268     299
```

Kuvio 42. Vastausaika koodivälimuistin kanssa.

CPU-käyttöön tällä ei ole suurta vaikutusta (vrt. kuvio 43 ja 38), vaan se on edelleen korea. Tämä voi kuitenkin johtua siitä, että palvelin ehtii käsitellä enemmän kyselyitä. Palvelin ei enää jumitu, vaan toimii pidemmänkin hyökkäyksen ajan.

```
top - 20:12:11 up 13 days,  2:52,  1 user,  load average: 0.81, 0.64, 0.49
Tasks:  81 total,   2 running,  79 sleeping,   0 stopped,   0 zombie
Cpu(s): 68.1%us,  4.0%sy,  0.0%ni, 27.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4049588k total,   925704k used,  3123884k free,    78440k buffers
Swap:  1044476k total,        0k used,  1044476k free,   558760k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7131 www-data  20   0  241m  29m  17m S 35.6  0.7   0:58.63 apache2
 7134 www-data  20   0  241m  29m  17m R 31.3  0.7   0:54.57 apache2
  870 mysql     20   0  871m  63m 7948 S  3.3  1.6   4:30.62 mysqld
```

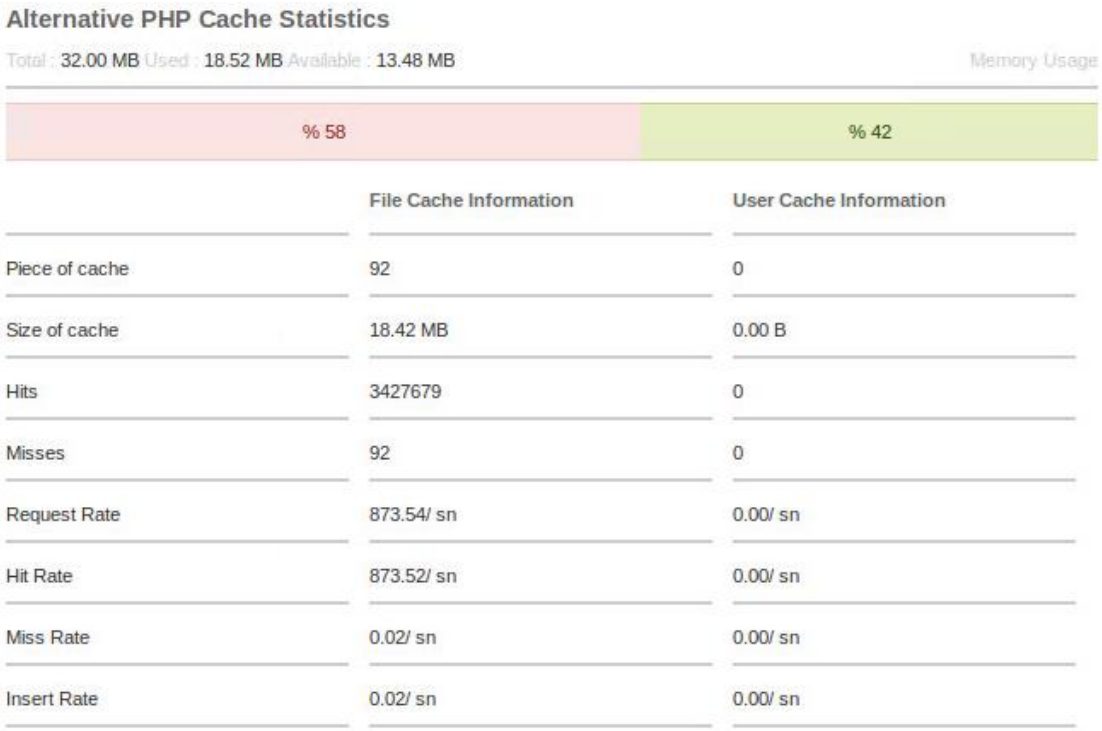
Kuvio 43. CPU-käyttö Keep-Dead hyökkäyksen aikana

Kuten kuvosta 44 nähdään, hyökkäyksen aikana koodivälimuistin kanssa sivunlataus kestää n. 300 – 500 ms. Tämä on huomattavasti parempi kuin aikaisemmin tapahtunut jumittuminen.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.5      0       2
Processing:   294  377  51.0    382     474
Waiting:      294  377  50.9    382     472
Total:        295  378  50.8    382     474
```

Kuvio 44. Latausajat koodivälimuistin kanssa hyökkäyksen aikana.

Kuvion 45 APC-tilastoista nähdään, että lähes kaikki tiedostot tulevat koodivälimuistista. Epäonnistuneita hakuja on vain 0.02 sekunnissa eli kyseessä on todennäköisesti vain ensimmäinen sivulataus, minkä aikana tarvittiin uutta tiedostoa.



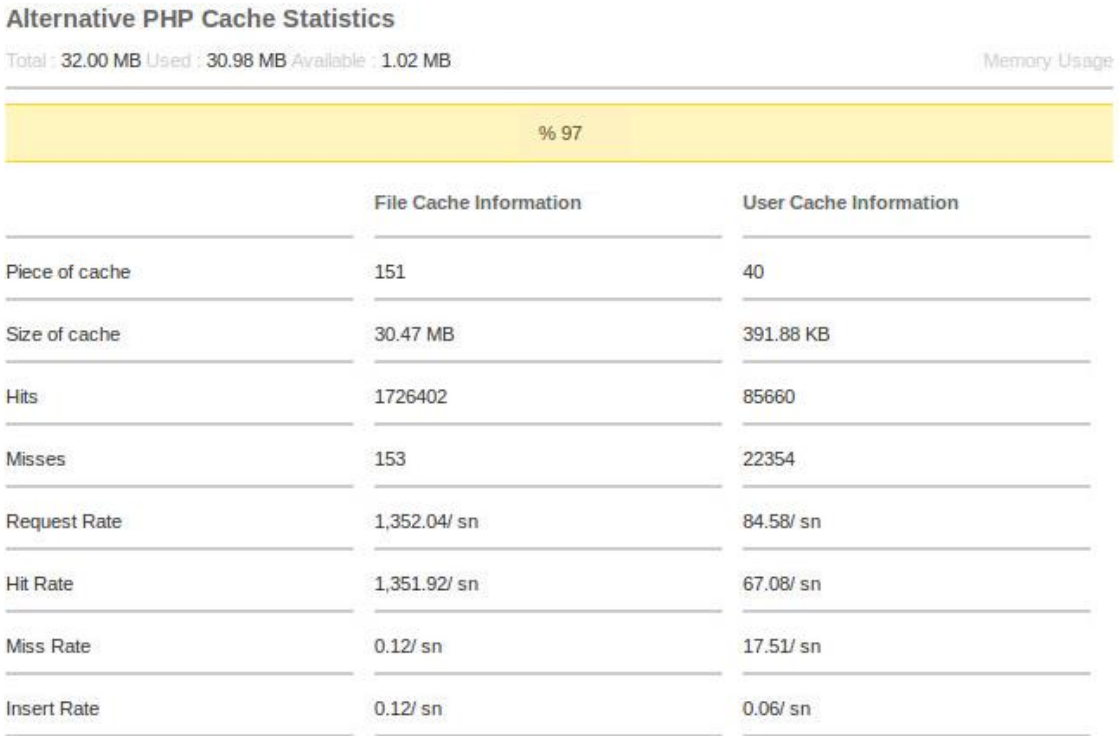
Kuvio 45. APC-tilastot hyökkäyksen aikana.

APC:tä voidaan käyttää koodivälimuistin lisäksi käyttää objektivälimuistina. Wordpressissä ei ole sisäänrakennettua tukea, mutta siihen on saatavilla useita lisäosia esimerkiksi APC Object Cache Backend, jonka voi asentaa hallintapaneelista. Kuten kuviosta 46 nähdään, tällä on vain pieni vaikutus.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.8      0       6
Processing:   276  338  33.2    337     450
Waiting:      276  337  33.2    336     450
Total:        276  338  33.2    340     451
```

Kuvio 46. Latausajat objektivälimuistin kanssa.

APC-tilastoista kuviossa 47 nähdään, että objektivälimuisti (User Cache Information) on toiminnassa, mutta kaikkea kysyttyä ei ole löydetty (Miss Rate), mutta ei myöskään lisätty (Insert Rate) välimuistiin.



Alternative PHP Cache Statistics

Total : 32.00 MB Used : 30.98 MB Available : 1.02 MB Memory Usage

% 97

	File Cache Information	User Cache Information
Piece of cache	151	40
Size of cache	30.47 MB	391.88 KB
Hits	1726402	85660
Misses	153	22354
Request Rate	1,352.04/ sn	84.58/ sn
Hit Rate	1,351.92/ sn	67.08/ sn
Miss Rate	0.12/ sn	17.51/ sn
Insert Rate	0.12/ sn	0.06/ sn

Kuvio 47. APC-tilastot

Kuten kuviosta 48 nähdään, CPU-käyttöön objektivälimuistin käyttöönotolla ei ollut vaikutusta.

```
top - 21:58:52 up 13 days,  4:38,  1 user,  load average: 0.78, 0.69, 0.57
Tasks:  81 total,   2 running,  79 sleeping,   0 stopped,   0 zombie
Cpu(s): 68.1%us,  2.0%sy,  0.0%ni, 29.6%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   4049588k total,   979592k used,  3069996k free,    85168k buffers
Swap:  1044476k total,        0k used,  1044476k free,   587072k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 7533 www-data  20   0  241m  28m  16m S 33.6  0.7   1:21.51 apache2
 7520 www-data  20   0  241m  30m  18m R 20.3  0.8   1:56.91 apache2
 7531 www-data  20   0  247m  43m  26m S 13.6  1.1   1:40.17 apache2
  870 mysql     20   0  871m  65m 7960 S  2.3  1.7   5:19.05 mysqld
```

Kuvio 48. CPU-käyttö Keep-Dead hyökkäyksen aikana (objektivälimuisti käytössä).

Olemattomat erot käytettäessä objektivälimuistilla ja ilman voivat johtua siitä, ettei Wordpressin tietokannassa ole sivuja tai APC:n välimuistin täyttymisestä. APC:n varaamaa muistia voidaan kasvattaa, muuttamalla asetusta apc.shm_size php.ini-tiedostossa. Testausta varten tietokantaan generoitiin 400 viestiä. Kuten kuviosta 49 nähdään, tällä oli huomattava vaikutus latausaikaan (n. kaksinkertainen).

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:   733  797  58.9    781     965
Waiting:      732  795  58.9    780     963
Total:        733  798  59.0    781     966
```

Kuvio 49. Latausajat suurella tietokannalla ilman objektivälimuistia.

Kuten kuviosta 50 nähdään, suuremmalla viestimäärällä objektivälimuistilla syntyy jo suurempi ero, etenkin maksimiajassa.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:   670  726  33.7    719     797
Waiting:      668  721  33.1    715     796
Total:        670  726  33.8    720     797
```

Kuvio 50. Latausajat suurella tietokannalla objektivälimuistilla.

APC-tilastoista kuviossa 51 voidaan todeta, että objektivälimuistia ei käytetä kovinkaan paljoa. Tästä voidaan todeta, ettei Wordpress käytä sitä kovinkaan tehokkaasti tässä tapauksessa. Syynä voi olla esimerkiksi se, että Keep-Deadilla hyökätään hakusivuun satunnaisilla hakusanoilla, mikä tekee välimuistiin tallentamisesta tehotonta.

Alternative PHP Cache Statistics

Total : 64.00 MB Used : 24.37 MB Available : 39.63 MB Memory Usag

% 38	% 62

	File Cache Information	User Cache Information
Piece of cache	115	182
Size of cache	23.90 MB	351.16 KB
Hits	1170931	48332
Misses	115	14432
Request Rate	1,599.79/ sn	85.74/ sn
Hit Rate	1,599.63/ sn	66.03/ sn
Miss Rate	0.16/ sn	19.72/ sn
Insert Rate	0.16/ sn	0.28/ sn

Kuvio 51. APC-tilastot objektivälimuisti suuremmalla muistimäärällä.

Keep-Dead perustuu palvelimelle lähetettävään suureen määrän pyyntöjä, ModSecurityä voidaan käyttää rajoittamaan lyhyessä ajassa tulevia kyselyitä. Nämä kohdistuu kaikkiin kyselyihin, joten tämä suojaa samalla kaikilta tulvitukseen perustuvilta hyökkäyksiltä.

Kuviossa 52 on sääntö, jolla rajataan yhdestä IP-osoitteesta tulevat kyselyt keskimäärin sekunti per kysely nopeuteen ja sen ylittävillä näytetään virheilmoitus. Säännöllä lasketaan IP-osoitteesta tulevat kyselyt (rivi 4) ja joka sekunti vähennetään yhdellä (rivi 2). Rivillä 3 tarkistetaan ylittääkö IP-osoitteesta tulleet kyselyt 60 ja jos ylittää lähetetään 509 virhe. Keskiarvon käyttäminen mahdollistaa purskeet, koska yksi sivu koostuu useasta osasta (CSS-tyylit, javascript, kuvat) tämä on tärkeää toimivuuden kannalta. Nolog-asetus estää ylimääräiset logimerkinnät, mutta ei kuitenkaan poista Apachen logiin tulevia merkintöjä epäonnistuneista latauksista. Haun kohdistamista näihin sivulatauksiin helpottaa, jos valitaan sellainen virhekoodi, jota ei käytetä muualla. Tässä tapauksessa valittiin virhekoodi 509.

```
SecAction initcol:ip=%{REMOTE_ADDR},pass,nolog
SecAction "phase:5,deprecatevar:ip.reqcount=1/1,pass,nolog"
SecRule IP:REQCOUNT "@gt 60" "phase:2,deny,status:509,skip:1,nolog"
SecAction "phase:2,pass,setvar:ip.reqcount=+1,nolog"
```

Kuvio 52. ModSecurity sääntö sivulatauksien rajoittamiseen

Kuten kuviosta 53 nähdään, tämän säännön lisäämisen jälkeen CPU-kuormaa juurikaan ole. Läpi päästetyt kyselyt näkyvät ajoittaisina piikkeinä.

```
top - 03:31:20 up 15 days, 10:11,  1 user,  load average: 0.00, 0.19, 0.40
Tasks:  92 total,   1 running,  91 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.7%us,  1.0%sy,  0.0%ni, 97.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4049588k total,  1388592k used,  2660996k free,   108108k buffers
Swap:  1044476k total,        0k used,  1044476k free,   871312k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6950 www-data  20   0  272m  25m  14m S  0.7  0.7   0:06.54 apache2
```

Kuvio 53. CPU-kuorma hyökkäyksen aikana.

Kuten kuviosta 54 nähdään, vastausaika hyökkäyksen aikana ilman sääntöjä kesimäärin 727 ms.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:   413  727 205.0    754    1259
Waiting:      330  634 194.0    680    1110
Total:        413  727 204.9    755    1259
```

Kuvio 54. Vastausaika hyökkäyksen aikana ilman sääntöjä (10 yhteyttä, 50 kyselyä).

Kuten kuviosta 55 nähdään, vastausaika hyökkäyksen aikana sääntöjen kesimäärin 578 ms.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:   266  578 172.5    522     933
Waiting:      266  577 172.5    522     933
Total:        266  578 172.5    522     934
```

Kuvio 55. Vastausaika hyökkäyksen aikana sääntöjen kanssa (10 yhteyttä, 50 kyselyä).

4.3.5 Testi 4 – Keep-Dead Wordpress palvelimella 3

Koska Keep-Dead perustuu valmisiin kyselyihin, ne välitetään taustalla pyörivällä Apachelle. Tästä johtuen nginx ei anna samanlaista suojaa Keep-Dead hyökkäystä vastaan kuin hitaita hyökkäyksiä. Kuten kuviosta 56 nähdään, CPU-käyttö nousee korkealle hyökkäyksen seurauksena.

```
top - 03:23:25 up 10 days, 13:23,  1 user,  load average: 0.27, 9.60, 14.82
Tasks:  86 total,   2 running,  84 sleeping,   0 stopped,   0 zombie
Cpu(s): 66.1%us,  2.0%sy,  0.0%ni, 31.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4049588k total,  1043260k used,  3006328k free,    63936k buffers
Swap:  1044476k total,        0k used,  1044476k free,   537508k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14437 www-data  20   0  210m  31m 3296 S  8.6  0.8   0:01.83 apache2
14529 www-data  20   0  215m  37m 3472 S  8.6  0.9   0:02.25 apache2
14450 www-data  20   0  210m  31m 3288 S  8.3  0.8   0:01.75 apache2
14462 www-data  20   0  210m  31m 3280 S  8.3  0.8   0:01.75 apache2
14487 www-data  20   0  215m  37m 3592 S  8.3  0.9   0:02.10 apache2
14422 www-data  20   0  210m  31m 3280 R  6.3  0.8   0:02.29 apache2
14473 www-data  20   0  219m  39m 4636 S  5.7  1.0   0:01.91 apache2
14455 www-data  20   0  210m  32m 3396 S  4.3  0.8   0:01.80 apache2
14489 www-data  20   0  212m  33m 4584 S  4.3  0.8   0:01.65 apache2
14531 www-data  20   0  210m  31m 3304 S  4.3  0.8   0:01.51 apache2
 8305 mysql     20   0  865m  47m 8264 S  0.7  1.2   1:51.48 mysqld
```

Kuvio 56. CPU-käyttö nginx-palvelimella.

Kuten kuviosta 57 nähdään, sivunlataukseen menee 10 s- 16s. Moni käyttäjä ehtisi jo yrittää uudelleen eli käytännössä palvelunestohyökkäys toteutuu.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.9      0      10
Processing:  2185 10278 1915.9  10234   16028
Waiting:     2185 10278 1915.9  10234   16028
Total:       2192 10279 1915.6  10234   16033
```

Kuvio 57. Apache Benchmark (50 yhteyttä, 500 pyyntöä).

Nginx avulla voidaan tehdä ratkaisu, jossa valmiit sivut tallennetaan välimuistiin ja sivut haetaan taustapalvelimelta (tässä tapauksessa Apache) vain jos sopivaa versiota ei ole välimuistissa tai se on vanhentunut.

Parhaiten tämä toimii sellaisissa tapauksissa, jossa käyttäjät eivät ole kirjautuneena sisään, eikä täyttä dynaamisuutta tarvita. Jos käyttäjät ovat kirjautuneena, niin jokaisen käyttäjän sivu eroaa, jolloin välimuistista ei ole hyötyä.

Kuten kuviosta 58 nähdään Keep-Dead hyökkäys tuottaa enää vain 11 % kuorman palvelimelle.

```
top - 01:44:41 up 14 days, 11:44,  1 user,  load average: 0.00, 0.01, 0.05
Tasks:  87 total,   1 running,  86 sleeping,   0 stopped,   0 zombie
Cpu(s): 11.0%us,  0.3%sy,  0.0%ni, 88.4%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   4049588k total,  1102328k used,  2947260k free,    76840k buffers
Swap:  1044476k total,        0k used,  1044476k free,   585232k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16492 www-data  20   0  210m  32m 3440 S  5.7  0.8   0:02.70 apache2
15145 www-data  20   0  210m  32m 3576 S  5.3  0.8   0:03.63 apache2
 8305 mysql     20   0  877m  51m 8308 S  0.3  1.3   3:05.75 mysqld
```

Kuvio 58. CPU-käyttö Keep-Dead hyökkäyksen aikana

Kuviosta 59 nähdään, että keskiarvo latausajalle on n. 200 ms ja maksimi n. 2 sekuntia. Apache Benchmarkilla avattiin etusivua, kun Keep-Dead hyökkäys tapahtui hakusivulle eli testatessa sitä ei ollut välimuistissa. Testiasetuksina oli 10 yhtäaikaista kyselyä ja yhteensä 100 sivulatausta. Tästä voidaan päätellä, että ensimmäiset 10 yhteyttä (10 %) joutui odottamaan hakua taustapalvelimelta. Tämän jälkeen tulleet kyselyt saatiin suoraan välimuistista.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.4      0       2
Processing:     0  191 572.1      1    1921
Waiting:        0  191 572.0      1    1921
Total:          0  192 572.4      1    1922

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      2
  80%      4
  90%   1870
  95%   1918
  98%   1919
  99%   1922
 100%   1922 (longest request)
```

Kuvio 59. Apache Benchmark tulokset sivulataukselle

Testin toistaminen tuotti tuloksen, jossa kaikki sivulataukset tulivat välimuistista ja kuten kuviosta 60 nähdään, kaikki sivulataukset kestivät vain muutaman millisekuntia.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:     0    2   0.8      2       4
Waiting:        0    2   0.7      2       4
Total:          1    2   0.7      2       4
```

Kuvio 60. Apache Benchmark tulokset sivulataukselle toistettuna

4.4 TLS/SSL-hyökkäyksiltä suojautuminen

4.4.1 Testi 1 - SSL Squeeze

SSL Squeezella hyökätään SSL-suojattuun palveluun. Tässä testissä kohteena on Palvelimella 1 sijaitseva Apache-webpalvelin. Hyökkäyksessä yritetään kuluttaa palvelimen CPU lähettämällä tekaistuja SSL-yhteyden avauksia. Palvelimen täytyy kuitenkin tarkistaa yhteyksien kelvollisuus.

SSLSqueeze suoritettiin komennolla:

```
/testit/sslsqueeze [kohde ip] [portti] [avaimenkoko]
[yhteydet]
```

SSL Squeeze vaatii, että tiedossa on palvelimen käyttämän sertifikaatin avaimenkoko. Tämä voidaan selvittää openssl-avulla:

```
openssl s_client -connect [kohde]:[portti]
```

Komento muodostaa yhteyden SSL-palvelimeen ja mahdollistaa tekstipohjausten protokollien, kuten HTTP-testauksen käsin ja esimerkiksi SSL Reneogation tuen testaamisen. Komento tulostaa mm. palvelimen sertifikaatin, siinä käytetyn avaimenkoon, käytetyn SSL tai TLS-version ja valitun salaustavan. Kuviossa 61 on esimerkkinä Palvelimen 1

```
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
```

Kuvio 61. SSL-avaimen ja yhteyden tiedot

Renegotiation tuen voi testata kirjoittamalla ”R” ja painamalla enteriä. Kuten kuviosta 62 nähdään, palvelin katkaisee yhteyden. Jos asiakkaalta renegotiation olisi sallittu, hyökkääjä voisi lähettää näitä pyyntöjä uuden yhteyden avaamisen sijaan.

```
R
RENEGOTIATING
140284584117920:error:1409E0E5:SSL routines:SSL3_WRITE_BYTES:ssl handshake failure:s3_pkt.c:596:
```

Kuvio 62. SSL Renegogation

Sadalla yhteydellä SSLSqueezen avulla saadaan yli 80 % CPU-kuorma, kuten kuvosta 63 nähdään.

```
/testit/sslsqueeze 192.168.0.2 443 1024 100
```

```
top - 00:07:29 up 6 days,  6:48,  1 user,  load average: 9.57, 6.80, 3.27
Tasks: 115 total,   7 running, 107 sleeping,   0 stopped,   1 zombie
Cpu(s): 61.5%us, 15.4%sy,  0.0%ni,  8.4%id,  0.0%wa,  0.0%hi, 14.7%si,  0.0%st
Mem:   4049592k total,  1546256k used,  2503336k free,    29132k buffers
Swap:  1048572k total,    50704k used,   997868k free,  1235472k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21554 www-data  20   0  189m 6432 1352 S  2.7  0.2   0:00.82 apache2
21556 www-data  20   0  189m 6432 1352 R  2.7  0.2   0:00.82 apache2
21376 www-data  20   0  189m 6432 1352 S  2.3  0.2   0:02.83 apache2
21448 www-data  20   0  189m 6432 1352 S  2.3  0.2   0:01.87 apache2
```

Kuvio 63. Top-ohjelman tuloste SSL Squeeze-hyökkäyksen aikana

Mittaukseen käytetiin Apache Bechmarkia, jolla luotiin kymmen samanaikaista pyyntyöä ja yhteensä 100 sivulatausta.

Kuten kuvosta 64 nähdään, ilman hyökkäystä Wordpressin sivulatausaika keskimäärin 1.7 sekentia.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:  1452 1728  70.6   1718    1991
Waiting:     1452 1728  70.7   1718    1991
Total:       1452 1728  70.6   1718    1991
```

Kuvio 64. Wordpress ilman hyökkäystä

Kuten kuvosta 65 nähdään, hyökkäyksen aikana Wordpressin sivunlataus aika hidastui selvästi keskimäärin 3.4 sekuntiin ja maksimiaika kasvoi 9.5 sekuntiin.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   0.8      0       4
Processing:  1272 3412 2030.7   2101    9441
Waiting:     1272 3412 2030.6   2101    9440
Total:       1273 3413 2030.5   2102    9442
```

Kuvio 65. Wordpress hyökkäyksen aikana

Hyökkäyksellä ei vaikutusta staattisiin sivuihin, kuten kuvion 66 tilastosta voidaan todeta.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       3
Processing:     1    9   6.9      6      29
Waiting:        1    8   6.9      5      29
Total:          2    9   6.9      6      29
```

Kuvio 66. Staattinen html-sivu hyökkäyksen aikana

4.4.2 Testi 2 – Suojautuminen

Hyökkäykseltä voidaan suojautua rajoittamalla SSL-yhteyksien määrää. Tämä toimii palvelusta riippumatta, muuttamalla käytettävän portin oikeaksi. Käyttämällä samaa hashlimit taulua (--hashlimit-name) rajoitus jaetaan sääntöjen kesken, mikä estää useamman samalla palvelimella olevan SSL-salatun palvelun hyödyntämisen hyökkäyksessä.

```
iptables -A INPUT -p tcp --dport 443 --syn -m state --state
NEW -m hashlimit --hashlimit-above 120/minute --hashlimit-
burst 20 --hashlimit-mode srcip --hashlimit-name ssl -j DROP
```

Sääntöjen toimivuus voidaan todeta “iptables -nvx -L” komennolla, joka näyttää tilastot. Kuten kuviosta 67 nähdään, osa paketeista tiputetaan.

```
Chain INPUT (policy ACCEPT 7366 packets, 701978 bytes)
    pkts      bytes target     prot opt in     out     source               destination
    3715   222900 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0
02 state NEW limit: above 2/sec burst 20 mode srcip htable-expire 60
```

Kuvio 67. Iptables-tilastot säännölle

Kuten kuviosta 68 nähdään, palvelimen CPU-käyttö nousee vain 2 % aiemman yli 80 % sijaan.

```
top - 05:39:45 up 16 days, 12:19,  1 user,  load average: 0.10, 1.79, 1.88
Tasks:  81 total,   1 running,  80 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.3%us,  0.7%sy,  0.0%ni, 97.7%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   4049588k total,  1422448k used,  2627140k free,   132632k buffers
Swap:  1044476k total,        0k used,  1044476k free,  1035416k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
10999 www-data  20   0  266m 6756 1352 S  0.3  0.2   0:02.10 apache2
11060 www-data  20   0  266m 6752 1356 S  0.3  0.2   0:01.18 apache2
11078 www-data  20   0  266m 6752 1356 S  0.3  0.2   0:01.16 apache2
11374 www-data  20   0  266m 6756 1352 S  0.3  0.2   0:00.36 apache2
```

Kuvio 68. CPU-käyttö suojatussa kohteessa hyökkäyksen aikana

Kuten kuviosta 69 nähdään, ilman hyökkäystä vastausaika on keskimäärin n. 560 millisekuntia ja maksimissaan 720 ms.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:   406  558  48.4    552     719
Waiting:      406  557  48.2    551     719
Total:        406  558  48.4    552     720
```

Kuvio 69. Wordpress latausajat ilman hyökkäystä.

Kuten kuviosta 70 nähdään, hyökkäyksen aikana keskimääräinen latausaika nousee hieman n. 570 milleskuntiin. Maksimiaika nousee enemmän n. 910 milleskuntiin.

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       1
Processing:   171  573  66.6    567     911
Waiting:      171  572  66.6    567     911
Total:        172  573  66.5    568     912
```

Kuvio 70. Wordpress latausajat hyökkäyksen aikana.

5 Tulokset

5.1 Hitaat hyökkäykset

Hitaat hyökkäykset ovat tehokastapa hyökätä webpalvelimiin. Hyökkäys perustuu yhteyksien varaamiseen ja toimii sellaisiin palvelimiin, jotka pystyvät käsittelemään rajallisen määrän yhteyksiä. Ympäristössä tämä testattiin SlowHTTPTestillä Slow Headers ja Slow Read-tilassa.

Palvelin 1, suojaamaton kohde

Palvelin 1 mahdollisti kaikkein yhteyksien varaamisen hyökkääjän toimista, jolloin käyttäjät eivät pääse palvelimelle ennen hyökkäyksen loppumista.

Palvelin 2, suojattu kohde

Palvelin 2 suojattiin käyttämällä Apachen reqtimeout-moduulia, joka ei kuitenkaan ole 2.2-versiossa oletuksena käytössä. Lisäksi rajoitettiin ModSecurity-moduulilla yhdestä IP-osoitteesta tulevat yhteydet 50:neen.

Rajoitukset olivat toimivat, mutta niillä ei saada täydellistä suojausta, sillä hajautetuissa palvelunestohyökkäyksissä on käytössä useita IP-osoitteita riippuen verkon koosta, jolloin rajoitukset eivät auta.

Hyökkäyksen vaikutukseen voi myös vaikuttaa kasvattamalla säikeiden maksimi määrää, mutta tämä puolestaan vaatii enemmän muistia palvelimelta.

Palvelin 3, suojattu kohde

Palvelin 3 suojattiin asentamalla nginx-webpalvelin, joka on erikoistunut suurten yhteysmäärien käsittelyy. Nginx toimi käänteisenä välityspalvelimena samalla palvelimella sijaitsevalle Apache:lle, jossa oli käytössä samat asetukset, kuin Palvelimella 1.

Tämä toimi huomattavasti paremmin, kuin Apache, joka tässä tapauksessa rajoittua 150 yhteyteen. Hyökkäys perustuu suureen määrään avoimia yhteyksiä ja niiden ylläpitämiseen lähettämällä dataa harvoin, mutta kuitenkin niin, ettei palvelin aika katkaise yhteyttä. Nginx ei kuitenkaan välitä pyyntöjä taustalla toimivalle Apachelle, ennen kuin kysely on valmis, joten siksi Apache ei ole rajoittava tekijä tässä hyökkäyksessä.

5.2 Keep-Dead

Keep-Deadilla hyökättiin palvelimelle asennettuun Wordpressiin. Tällä hyökkäyksellä ei ollut vaikutusta staattisiin sivuihin, sillä niiden lataaminen ei saa palvelimen prosessorikäyttöä nousemaan tarpeeksi.

Palvelin 1, suojaamaton kohde

Pahimmillaan palvelin 1 jumittui, kuten hitaissa hyökkäyksissä, kun palvelin ei kyennyt vastaamaan tarpeeksi nopeasti.

Palvelin 2, suojattu kohde

Palvelin 2 suojattiin rajoittamalla yhdestä IP-osoitteesta tulevien kyselyiden määrää. Lisäksi asennettiin PHP:hen APC-lisäosa, joka tallentaa tulkitun koodin välimuistiin, jolloin tulkintaa ei tarvitse suorittaa joka sivulataukselle.

Lisäksi ModSecurityllä rajoitettiin yhdestä IP-osoitteesta tulevat sivupyynnöt yhteen per sekunti minuutin aikajaksolla.

Palvelin 3, suojattu kohde

Palvelin 3 suojattiin ottamalla käyttöön nginxin välimuisti, jolloin jokainen sivu ladataan vain kerran ja muilla kerroilla se saadaan välimuistista.

Hyökkäyksen aikana ei välimuistissa oleva sivulataus kesti n. 1.9 sekuntia, mutta välimuistissa olevat toimivat kuten staattiset sivut ja näissä vastausaika on vain pari millisekuntia.

Välimuistia käyttämällä menetetään kuitenkin dynaamisuus, jolloin tämä ratkaisu ei sovellu kaikille. Lisäksi tämä ei toimi sellaisissa sivuissa, joissa suurin osa käyttäjistä on kirjautuneena sisään, koska tällöin jokaisen saama sivu eroaa, jolloin niistä ei ole hyötyä muiden sivupyynnöissä. Wordpressissä kirjautuminen on lähinnä sivun ylläpitäjälle, joten tämä on toimiva ratkaisu.

Hyökkäyksen aiheuttamaan prosessorikuormaan tällä ei kuitenkaan ole vaikutusta, koska hyökkäyksessä ladattiin aina satunnaisella hakusanalla, jolloin välimuistia ei ollut mahdollista hyödyntää. Tähän voidaan yhdistää palvelimelle 2 tehdyt muutokset,

jolloin itse hyökkäykseenkin voidaan puuttua ja näin nopeuttaa sivujen, jotka eivät ole välimuistissa latautumista.

5.3 SSL Squeeze

SSL Squeezella hyökättiin 100 yhteydellä ja verrattiin keskimääräistä Wordpressin-sivunlataus aikaa. Tällä hyökkäyksellä ei ollut vaikutusta staattisiin sivuihin.

Palvelin 1, suojaamaton kohde

Suojaajattomassa kohteessa tällä hyökkäyksellä oli samat vaikutukset kuin Keep-Dead hyökkäyksessä eli CPU-käyttö nousi korkeaksi ja sen myötä vastausaika hidastui. Tässä ei kuitenkaan havaittu samanlaista jumittumista, kuin Keep-Dead hyökkäyksessä.

Palvelin 2, suojattu kohde

Palvelin 2 suojattiin rajoittamalla SSL-yhteyksien muodostusta käyttäen IPTables-palomuuria. Tämä esti hyökkäyksen toiminnan ja prosessorikäyttö ei enää noussut ja hyökkäyksen aiheuttama vaikutus pieneni. Hyökkäyksellä on silti pieni vaikutus maksimilataus aikaan, joka nousi n. 200 millisekunnilla, kun keskimääräinen latausaika pysyi lähes muuttumattomana hieman yli 10 ms nousulla.

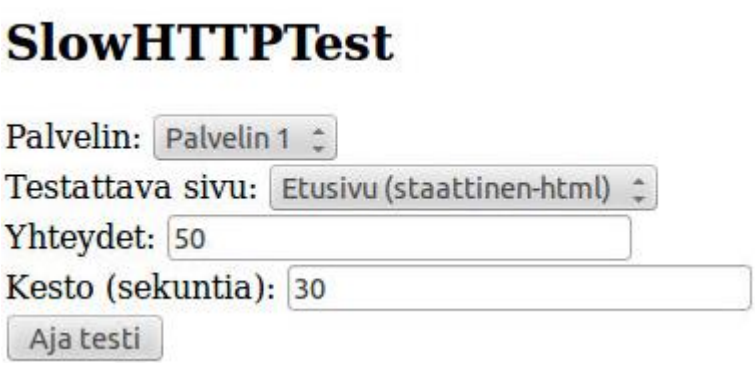
5.4 Testien ajo ympäristössä

Valmiissa ympäristössä voidaan ajaa testejä käyttämällä joko webkäyttöliittymää tai komentoriviä. Komentoriviltä voidaan ajaa webkäyttöliittymästä löytyvien Slow-HTTPTestin ja Apache Benchmarkin lisäksi Slowloris, SSLSqueeze ja Keep-Dead. Palvelimilla on käyttössä lopulliset asetukset (ks. liitteet 12-14).

Webkäyttöliittymä

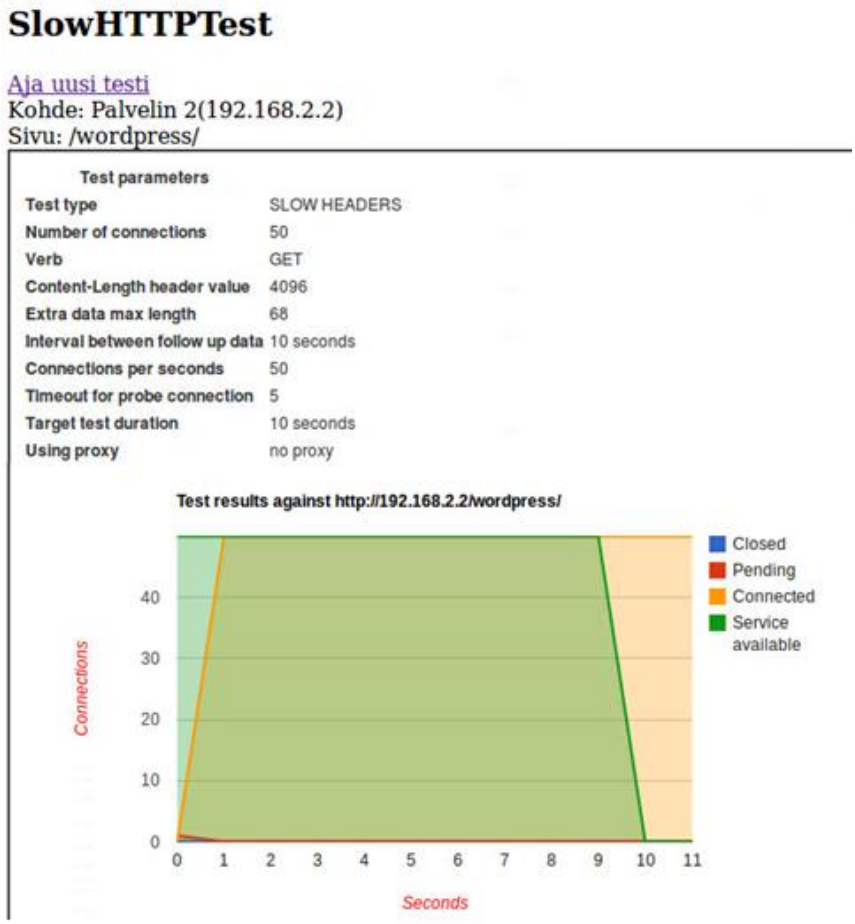
Ympäristöön toteutettiin PHP-kielillä yksinkertainen käyttöliittymä (ks. liite 15). Webkäyttöliittymä on jaettu kahteen osaan. Ensimmäinen, varsinaiseen hyökkäykseen käytettävä osa on hyökkäys 1-palvelimella. Toista osaa käytetään mittauksiin ja se on hyökkäys 2-palvelimella.

Webkäyttöliittymä hyökkäyksiä varten on Hyökkäys 1-koneella. Webkäyttöliittymässä on kuvion 71 mukaiset asetukset, joilla valitaan kohde ja kesto.



Kuvio 71. Webkäyttöliittymän testiasetukset.

SlowHTTPTestin kautta saadaan tulokset suoraan html-muodossa kuvion 72 mukaisesti, mikä helpottaa sen liittämistä osaksi sivua. Webkäyttöliittymässä ei ole reaaliaikaista edistymisen seurantaan ja sivunlataus kestää testin keston ajan.



Kuvio 72. SlowHTTPTestin Tulossivu webkäyttöliittymässä.

Webkäyttöliittymä mittauksia varten on Hyökkäys 2-koneella, tämä siksi, että hyökkäys vaikuttaisi muuten testistä saataviin tuloksiin. Mittauksissa on tuettuna Apache Benchmark ja sillä tuettuna kuvion 73 mukaiset asetukset.

Kuvio 73. Apache Benchmarkin asetukset webkäyttöliittymässä

Komentoriviltä

Komentoriviltä voidaan ajaa SlowHTTPTestin ja Apache Benchmarkin lisäksi Slowloris, SSLSqueeze ja Keep-Dead hyökkäykset. Nämä testit kestävät pysäytykseen asti, mikä tekee automatisoinnista vaikeaa.

Slowloris-hyökkäys voidaan ajaa Hyökkäys 1-koneelta komennolla:

```
perl /testit/slowloris.pl –dns [palvelimen ip]
```

Mittauskohteena on lyhyt staattinen html-sivu, joka on palvelimen juurikansiossa. Slowloriksessa ei mahdollisuutta muuttaa tätä. Hyökkäys lopetetaan mittauksen jälkeen CTRL+C-näppäinyhdistelmällä. Slowloriksessa on lisäksi testauskomento, jota voidaan käyttää sopivan aikakatkaisun määrittämiseen ja sen seuksena

```
perl /testit/slowloris.pl –dns [palvelimen ip] -test
```

Vaihtoehtoisesti SlowHTTPTestiä voi käyttää myös komentoriviltä, yhteyksien määrää (200) ja kestoa (120 s) voi muuttaa.

```
/testit/testi1.sh testi http://[palvelimen ip]/ 200 120
```

Tai kutsumalla SlowHTTPTestiä suoraan halutuilla parametreillä.

```
slowhttptest –g –o testi –c 200 –l 120 –u http://[palvelimen
ip]/
```

Vastausaikojen mittaamisen voidaan käyttää Hyökkäys 2 / Mittaus-palvelimelta löytyvää Apache Benchmark (ab)- ohjelmaa. Esimerkiksi kymmen sivulatausta ja yhteensä 100 sivulatausta.

```
ab -r -c 10 -n 100 http://[palvelimen ip]/
```

6 Pohdinta

Opinnäytetyön tavoitteena oli tutkia sovelluskerroksen palvelunestohyökkäyksiä, tapoja torjua ja tunnistaa näitä hyökkäyksiä sekä toteuttaa ympäristö, jossa voidaan testata näitä hyökkäyksiä ja vertailla hyökkäysten vaikutusta suojaamattomiin ja suojattuihin kohteisiin. Lisäksi palvelimet olivat saman tehoisia, jolloin se ei aiheuta eroa. Palvelimen kapasiteettia kasvattamalla voidaan suojautua hyökkäyksiä vastaan, mutta jos palvelimia ei olla suojattu palvelunestohyökkäys on silti mahdollinen ja tehokkaillakin palvelimilla pätevät samat suojausmenetelmät.

Valmiiseen ympäristöön kuuluu kolme kohdetta, joista yksi on suojaamaton ja kaksi hieman eri tavalla suojattua. Tämä mahdollistaa vaikutusten vertailemisen. Hyökkäyksiä varten on kaksi erillistä konetta, joista toisella suoritetaan varsinainen hyökkäys ja toisella mittaukset.

Tutustuin ja testasin hitaisiin-, kyselytulvitus- ja SSL-Squeeze-hyökkäyksiin. Kyselytulvitukseen käytettiin Keep-Dead-skriptiä, joka käyttää HEAD-pyyntöjä ja Keep-Alive yhteyksiä. Tämä vähentää TCP-yhteyksien avauksia ja hyökkääjälle takaisinpäin tulevaa liikenne ja mahdollistaa siten palvelimen tulvittamisen nopeammin.

Kävin läpi tapoja tunnistaa näitä hyökkäyksiä, esimerkiksi logista. Näitä olisi voinut hyödyntää toteutuksessa ja tehdä skripti, joka seuraa logia ja etsii merkkejä näistä hyökkäyksistä ja ilmoittaa siitä ylläpitäjälle esimerkiksi sähköpostilla. Tämä mahdollistaisi reagoimisen hyökkäyksiin reaaliajassa, mikäli sille on tarvetta.

Prosessorikäytön nostoon tähtäävillä hyökkäyksillä kuten Keep-Dead ja SSL Squeeze ei ollut vaikutusta staattisten sivujen latausaikaan, ainoastaan dynaamisten PHP-sivujen. Nykyään kuitenkin monet verkkosivuista on toteutettu dynaamisesti erilaisilla sisällönhallintajärjestelmillä, jolloin kohteen löytäminen ei ole vaikeaa.

Tällä hetkellä testausta varten on yksinkertainen PHP:llä koodattu webkäyttöliittymä. Webkäyttöliittymän kautta voidaan käyttää SlowHTTPTest ja Apache Benchmark-työkaluja. PHP:llä työkalujen käynnistys on helppoa, mutta esimerkiksi tulosteen lukeminen ja välittäminen selaimelle reaaliajassa on huomattavasti hankalampaa. Tässä versiossa selaimelle ei tule minkäänlaista tietoa testin edistymisestä. Tiedot edistymisestä olisi erityisesti pitkissä testeissä tarpeen.

Muiden työkalujen osalta webkäyttöliittymää ei ollut mahdollista järkevästä toteuttaa, koska nämä ovat tehty konsolilta käytettäväksi ja näissä ei ole tarpeellisia parametreja, joilla saada hyökkäys loppumaan automaattisesti.

Tätä voisi parantaa tekemällä testit moduulina valmiiseen testausjärjestelmään, kuten Metasploit, jolloin ei tarvitse huolehtia käyttöliittymän koodauksesta. PHP-skriptit eivät ole paras tapa, koska PHP:ssä ei ole helppoa tapaa kommunikoida käynnissä olevan skriptin kanssa.

Lähteet

Apache HTTP Server Version 2.4 Documentation. 2014. Apache Software Foundation. Viitattu 4.4.2014. http://httpd.apache.org/docs/2.4/.

Arbor Networks. 2013. Arbor Networks Releases Q3 Global DDoS Attack Trends Data. Viitattu 20.3.2014. http://www.arbornetworks.com/news-and-events/press-releases/2013-press-releases/5026-arbor-networks-releases-q3-global-ddos-attack-trends-data.

Atias, R. 2014. One of World's Largest Websites Hacked: Turns Visitors into "DDoS Zombies". Viitattu 6.4.2014. http://www.incapsula.com/blog/world-largest-site-xss-ddos-zombies.html.

Clark J. 2013. Malicious JavaScript flips ad network into rentable botnet. Viitattu 4.4.2014. http://www.theregister.co.uk/2013/07/31/whitehat_security_ad_networks_botnet/.

Dae-il, J., Minsoo, K., Hyun-chul, J. & Bong-Nam. 2009. Analysis of HTTP2P Botnet: Case Study Waledac. Viitattu 7.2.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, IEEE Explore.

Damas, J. Graff, M. & Vixie, P. 2013. Extension Mechanisms for DNS. Viitattu 17.3.2014. http://tools.ietf.org/html/rfc6891.

Dierks, T. & Rescorla, E. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. Viitattu 4.4.2014. https://tools.ietf.org/html/rfc5246.

DNS Amplification Attacks. 2013. US-CERT. Viitattu 10.2.2014. https://www.us-cert.gov/ncas/alerts/TA13-088A.

Durcekova, V., Schwartz, L., Shahmehri, N. 2012. Sophisticated Denial of Service Attacks Aimed at Application Layer. Viitattu 14.2.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, Books 24x7.

Erb, B. 2012. Concurrent Programming for Scalable Web Architectures. Viitattu 4.4.2014. http://berb.github.io/diploma-thesis/original/043_threadsevents.html.

Esrun. 2011. Keep-Alive DoS Script. Viitattu 13.4.2014. http://www.es-run.co.uk/blog/keep-alive-dos-script/.

Fantz, A., & Shuber, A. 2010. WikiLeaks 'Anonymous' hackers: 'We will fight'. Viitattu 2.3.2014. http://edition.cnn.com/2010/US/12/09/hackers.wikileaks/.

Fette, I., Melnikov, A. 2011. The WebSocket Protocol. Viitattu 14.4.2014. https://tools.ietf.org/html/rfc6455.

Fielding, R., Irvine, U., Gettys, J., Mogul, J., Frystyk, H., Masinter, H., Leach, L. & Berners-Lee, T. 1999. Hypertext Transfer Protocol -- HTTP/1.1. Viitattu 15.2.2014. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

FreeBSD Project. 2000. FreeBSD Kernel Developer's Manual - ACCF_HTTP. Viitattu 15.2.2014. http://www.freebsd.org/cgi/man.cgi?query=accf_http&sektion=9.

Giobbi, R. 2009. Mitigating Slowloris. Viitattu 15.4.2014. https://www.cert.org/blogs/certcc/post.cfm?EntryID=42.

Grossman, J. & Johansen, M. 2013. Million Browser Botnet. https://media.blackhat.com/us-13/us-13-Grossman-Million-Browser-Botnet.pdf. Viitattu 7.2.2014.

Hansen, R. 2009. Slowloris HTTP DoS. http://ha.ckers.org/slowloris/. Viitattu 16.2.2014.

Jyvsectec. 2013. Jyvsectec. http://jyvsectec.fi/jyvsectec/. Viitattu 4.4.2014

Leyden, J. 2013. Casino DDoS duo caged for five years after blackmail buyout threat. Viitattu 1.3.2014. http://www.theregister.co.uk/2013/12/19/casino_cyber_extortionists_jailed/.

MacVittie, L. 2011. Dispelling the New SSL Myth. Viitattu 5.4.2014. https://devcentral.f5.com/articles/dispelling-the-new-ssl-myth.

Mantripragada, S. 2013. DNS Related DDOS Detection and Prevention Techniques. Viitattu 14.3.2014. https://community.infoblox.com/blogs/2013/03/26/dns-related-ddos-detection-and-prevention-techniques.

Mehan, J. 2009. CyberWar, CyberTerror, CyberCrime. Viitattu 1.3.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, Ebrary.

Mehmud, A. 2011. Internet Denial of Service Attacks and Defense Mechanisms. Viitattu 6.2.2014. http://people.cs.pitt.edu/~mehmud/docs/abliz11-TR-11-178.pdf.

Memcached. 2014. PHP. Viitattu 15.4.2014. http://www.php.net/manual/en/intro.memcached.php.

ModSecurity. 2014. Trustware. Viitattu 19.3.2014. http://www.modsecurity.org/projects/modsecurity/.

Moore, S. 2014. WordPress Pingback Attacks and our WAF. Viitattu 15.4.2014. . http://blog.cloudflare.com/wordpress-pingback-attacks-and-our-waf.

Nottingham, M. 2012. WG Review: Hypertext Transfer Protocol Bis (httpbis). . Viitattu .4.2014. http://lists.w3.org/Archives/Public/ietf-http-wg/2012Jul-Sep/1282.html

NTP Amplification Attacks Using CVE-2013-5211. 2014. US-CERT. Viitattu 10.2.2014. https://www.us-cert.gov/ncas/alerts/TA14-013A.

Onn Chee, W. & Brennan, T. 2010. H.....t.....t....p....p....o....s....t. Viitattu 13.3.2014 https://www.owasp.org/images/4/43/Layer_7_DDOS.pdf.

OPcache. 2014. PHP. Viitattu 15.4.2014.. http://fi2.php.net/manual/en/intro.opcache.php.

Prince, M. 2014. Technical Details Behind a 400Gbps NTP Amplification DDoS Attack. Viitattu 13.4.2014. http://blog.cloudflare.com/technical-details-behind-a-400gbps-ntp-amplification-ddos-attack.

Prowell, S., Kraus, R. & Borkin, M. 2010. Seven Deadliest Network Attacks. Viitattu 6.2.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, Books 24x7.

Shekyan, S. 2013. SlowHTTPTest. Viitattu 13.3.2014. http://code.google.com/p/slowhttptest/.

Sukalp, B. 2012. Server based DoS vulnerabilities in SSL/TLS Protocols. Viitattu 11.4.2014. http://alexandria.tue.nl/extra1/afstversl/wsk-i/bhople2012.pdf.

Thomson, I. 2013. Egyptian navy captures divers trying to cut undersea internet cables. Viitattu 3.3.2014. http://www.theregister.co.uk/2013/03/27/egypt_cables_cut_arrest/.

Tipton, H. & Krause, M. 2007. Information Security Management Handbook, Sixth Edition, Volume 1. Viitattu 6.2.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, Books 24x7.

Usage of SPDY for websites. 2014. w3techs. Viitatu 4.4.2014. http://w3techs.com/technologies/details/ce-spdy/all/all.

Zhang, L., Yu, S. & Wu D. 2011. A Survey on Latest Botnet Attack and Defense. Viitattu 7.2.2014. http://www.jamk.fi/kirjasto, Nelli-portaali, IEEE Explore.

Ylönen, T., Lonvick, C . 2006. The Secure Shell (SSH) Connection Protocol. Viitattu 6.4.2014. http://tools.ietf.org/html/rfc4254.

Liitteet

Liite 1 - Apache + PHP + MySQL

Apachen, PHP5 ja MySQL asennus onnistuu Ubuntussa komennolla:

```
sudo apt-get install apache2 php5 libapache2-mod-php5 php5-
mysql mysql-server
```

SSL ei ole oletuksena käytössä, mutta se voidaan ottaa käyttöön komennolla:

```
sudo a2enmod ssl
```

Lisäksi käyttöön voidaan ottaa oletusasetukset SSL-sivustolle komennolla:

```
sudo a2ensite default-ssl
```

Asetukset löytyvät tiedostosta /etc/apache2/sites-available/default-ssl. Asetukseen tarvitsee muuttaa SSL-sertifikaatin ja -avaimen sijainti.

```
SSLCertificateFile    /etc/ssl/certs/server.crt
SSLCertificateKeyFile /etc/ssl/private/server.key
```

Asetukset otetaan käyttöön käynnistämällä Apache uudelleen komennolla

```
sudo service apache2 restart
```

Liite 2 - nginx asennus

Nginx asennettiin komennolla

```
sudo apt-get install nginx
```

Liite 3 - Sertifikaatin luominen

SSL kanssa tarvitaan sertifikaatti, jolla palvelin tunnistetaan. Normaalisti nämä ovat luotetun tahon allekirjoittamia, mutta testatessa voidaan käyttää myös itseallekirjoitettuja.

Sertifikaattia varten tarvitaan avain, jonka voi luoda esimerkiksi OpenSSL avulla seuraavalla komennolla:

```
openssl genrsa -out server.key 1024
```

Tämä luo 1024 bittisen RSA-avaimen.

Tämän jälkeen luodaan allekirjoituspyyntö avaimelle komennolla:

```
openssl req -new -key server.key -out server.csr
```

Komento kysyy tarvittavat tiedot, kuten omistajan ja sijainnin. Itseallekirjoitetussa sertifikaatissa ainoastaan Common Name kohdalla on väliä ja siinä käytetään verkkosivun osoitetta.

Allekirjoituspyynnöstä luodaan varsinainen sertifikaatti komennolla.

```
openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
```

Liite 4 - SlowHTTPTest asennus

SlowHTTPTest ei tule Ubuntun oletus repositoryissa vaan se täytyy kääntää itse tai lisätä jokin repository josta se löytyy.

Kääntämiseen tarvitaan kääntötyökalut ja SSL-kirjasto.

```
apt-get install make g++ libssl-dev
```

SlowHTTPTestistä ladattiin versio 1.6, mikä oli kirjoitus hetkellä uusin

```
wget http://slowhttptest.googlecode.com/files/slowhttptest-
1.6.tar.gz
```

Paketti puretaan, jonka jälkeen se voidaan kääntää ja asentaa.

```
tar -xf slowhttptest-1.6.tar.gz
cd slowhttptest-1.6
./configure
make
sudo make install
```

Liite 5 - Keep-Dead

Keep-Dead ladataan komennolla:

```
wget http://www.esrun.co.uk/blog/wp-
content/uploads/2011/03/Keep-Dead.zip
```

Tämän jälkeen ladattu paketti puretaan

```
unzip Keep-Dead.zip
```

Lisäksi tarvitaan PHP-tulkki, joka voidaan Ubuntussa asentaa komennolla:

```
sudo apt-get install php5-cli
```

Koska asetukset ovat suoraan lähdekoodissa, siitä tehdään jokaiselle kohdepalvelimelle oma kopio, johon muokataan palvelimen osoite, muilta osin asetukset ovat samat kaikille.

```
/* target_url
The URL to be attacked. You should try and choose a resource intensive page suc
or live stat page. Use %rand% for a random value to be automatically generated
*/
$target_url = "http://192.168.0.2/wordpress/?s=%rand%";

/* max_requests
The maximum number of requests to be made. If you're running this via command l
high and simply quit the script at any point . If you plan to run this via a we
*/
$max_requests = 100000000;

/* max_requests_per_connection
The maximum number of requests to be made per connection. Maximum value is 100
*/
$max_requests_per_connection = 100;

/* delay_between_connections
The number of seconds to delay between opening a new connection. Recommended va
*/
$delay_between_connections = 0.5;

/* delay_between_requests
The number of seconds to delay between outgoing requests. Recommended value: 0.
*/
$delay_between_requests = 0.01;
```

Liite 6 – Slowloris

Slowloris on yksittäinen perl-tiedosto, joka voidaan ladata komennolla:

```
wget http://ha.ckers.org/slowloris/slowloris.pl
```

Lisäksi tarvitaan kaksi perl-kirjastoa, jotka voidaan asentaa komennolla

```
perl -MCPAN -e 'install IO::Socket::INET'
perl -MCPAN -e 'install IO::Socket::SSL'
```

Liite 7 – SSLSqueeze

SSL Squeeze ladattiin Githubista komennolla:

```
wget https://github.com/mmgaggle/sslsqueeze/archive/master.zip
unzip master.zip
```

SSL Squeeze ei suoraan käänny, johtuen linkittäjässä tapahtuneiden muutoksien takia, tätä varten jouduttiin muuttamaan Makefilestä ”LDFLAGS”-kohta ”LDLIBS”, ennen kuin SSL Squeeze saatiin käännettyä.

Liite 8 – iftop

Iftop asennettiin komennolla:

```
sudo apt-get install iftop
```

Liite 9 - Testiscriptit

Testi 1 – SlowHTTPTest

```
slowhttptest -g -o $1 -c $3 -l $4 -u $2
```

```
/testit/testi1.sh [nimi] [kohde] [yhteysmäärä] [kesto]
```

Testi 2 - SlowHTTPTest maksimi yhteysmäärän testaus

```
#!/bin/bash
L=$(($3/2))
slowhttptest -g -o $1 -c $3 -l $L -r 2 -u $2
```

```
/testit/testi2.sh [nimi] [kohde] [yhteysmäärä]
```

Liite 10- ModSecurity

ModSecurity asennettiin komennolla:

```
sudo apt-get install libapache2-modsecurity
```

ModSecurityn oletusasetukset otettiin käyttöön komennolla:

```
sudo cp /etc/modsecurity/modsecurity.conf-recommended
/etc/modsecurity/modsecurity.conf
```

Liite 11 – PHP-apc asennus

PHP:n APC-välimuistimoduuli voidaan asentaa Ubuntussa suoraan repositorysta.

```
apt-get install php-apc
```

APC sisältää toiminnot välimuistin tilan tarkkailuun, mutta lisäksi tarvitaan skripti, jolla tarkistella niitä yksi tällainen on apc-stats, jonkan voi ladata Githubista.

```
wget https://github.com/o/apc-stats/archive/master.zip
```

Tämän jälkeen tiedosto puretaan ja kansio (apc-stats-master) siiretään haluttuun paikaan (esim. /var/www, jolloin se on käytettävissä muuttamatta Apachen asetuksia).

```
unzip master.zip
```

Liite 12 - Asetukset – Palvelin 1

Reqtimeout moduuli poistettu käytöstä komennolla:

```
ae2dismod reqtimeout
```

Liite 13 - Asetukset – Palvelin 2

ModSecurity (/etc/modsecurity/saanto.conf)

```
SecReadStateLimit 50
SecWriteStateLimit 50

SecRuleEngine On
SecDataDir /tmp/

SecAction initcol:ip=%{REMOTE_ADDR},pass,nolog
SecAction "phase:5,deprecatevar:ip.reqcount=1/1,pass,nolog"
SecRule IP:REQCOUNT "@gt 60" "phase:2,deny,status:509,skip:1,nolog"
SecAction "phase:2,pass,setvar:ip.reqcount=+1,nolog"
```

Liite 14 - Asetukset – Palvelin 3

nginx (/etc/nginx/sites-enable/default)

```
proxy_cache_path /data/nginx/cache keys_zone=cache_zone:10m;
proxy_temp_path /tmp;

server {
        listen 80;
        listen 443 ssl;

        ssl_certificate /etc/ssl/certs/server.crt;
        ssl_certificate_key /etc/ssl/private/server.key;

        server_name 192.168.3.2;

        location / {
                proxy_pass http://server3:8080;
                proxy_set_header Host   $host;

                proxy_cache cache_zone;
                proxy_cache_valid 200 302 10m;
                        #proxy_cache_key
        }
}
```

Liite 15 – Webkäyttöliittymän lähdekoodi

Hyökkäys

```
<?php

$kohteet = array(
		array(
				'nimi' => 'Palvelin 1',
				'ip' => '192.168.0.2'
		),
		array(
				'nimi' => 'Palvelin 2',
				'ip' => '192.168.2.2',
		),
		array(
				'nimi' => 'Palvelin 3',
				'ip' => '192.168.3.2',
		)
);

if (empty($_POST['testi']))
		testilista();
elseif ($_POST['testi'] == 'sl1')
{
		$kohde = $kohteet[$_POST['kohde']];
		echo '<h2>SlowHTTPTest</h2>
				<a href="/">Aja uusi testi</a><br>
				Kohde: ', $kohde['nimi'], '(', $kohde['ip'], ')<br />
				Sivu: ', $_POST['testisivu'], '<br>';

		exec(
				sprintf(
						'slowhttptest -g -o /var/www/sl -c %d -l %d -u
%s',
						(int) $_POST['yhteydet'],
						(int) $_POST['kesto'],
						escapeshellarg('http://' . $kohde['ip']
.$_POST['testisivu'])
				),
				$out, $virhekoodi
		);
		if ($virhekoodi)
		echo '<strong>SlowHTTPTest palautti virhekoodin ', $virhekoodi;

		echo '
<iframe src="/sl.html" style="width: 100%; height: 800px">
</iframe>';
}

function testilista()
{
		global $kohteet;?>
<form action="/" method="post">
		<h2>SlowHTTPTest</h2>
		<input type="hidden" name="testi" value="sl1">
		Palvelin: <select name="kohde">
				<?php
				foreach ($kohteet as $id => $kohde)
						echo '<option value="', $id, '">',
$kohde['nimi'],'';
				?>
		</select><br >
		Testattava sivu: <select name="testisivu">
				<option value="/">Etusivu (staattinen-html)
				<option value="/wordpress/">Wordpress
		</select><br>
		Yhteydet: <input name="yhteydet" value="50"><br>
		Kesto (sekuntia): <input name="kesto" value="30"><br>
		<input type="submit" value="Aja testi">

</form><?php
}
?>
```

Mittaus

```
<?php

$kohteet = array(
		array(
				'nimi' => 'Palvelin 1',
				'ip' => '192.168.0.2'
		),
		array(
				'nimi' => 'Palvelin 2',
				'ip' => '192.168.2.2',
		),
		array(
				'nimi' => 'Palvelin 3',
				'ip' => '192.168.3.2',
		)
);

if (empty($_POST['testi']))
		testilista();
elseif ($_POST['testi'] == 'ab')
{
		$kohde = $kohteet[$_POST['kohde']];
		echo '<h2>Apache Benchmark</h2>
				<a href="/">Aja uusi testi</a><br>
				Kohde: ', $kohde['nimi'], '(', $kohde['ip'], ')<br />
				Sivu: ', $_POST['testisivu'], '<br>';

		exec(
				sprintf(
						'ab -r -c %d -n %d %s',
						(int) $_POST['yhteydet'],
						(int) $_POST['kesto'],
						escapeshellarg('http://' . $kohde['ip']
.$_POST['testisivu'])
				),
				$out, $virhekoodi
		);
		if ($virhekoodi)
		echo '<strong>Apache Benchmark palautti virhekoodin ', $virhekoodi;

		echo '<pre>', implode("\n", $out), '</pre>';
}

?>

<?php
function testilista()
{
		global $kohteet;?>
<form action="/" method="post">
		<h2>Apache Benchmark</h2>
		<input type="hidden" name="testi" value="ab">
		Palvelin: <select name="kohde">
				<?php
				foreach ($kohteet as $id => $kohde)
						echo '<option value="', $id, '">',
$kohde['nimi'],'';
				?>
		</select><br >
		Testattava sivu: <select name="testisivu">
				<option value="/">Etusivu (staattinen-html)
				<option value="/wordpress/">Wordpress
		</select><br>
		Yhteydet: <input name="yhteydet" value="50"><br>
		Sivulataukset yht.: <input name="kesto" value="100"><br>
		<input type="submit" value="Aja testi">

</form><?php
}
?>
```