Joonas Karjalainen

Sovellustason palvelunestohyökkäysten havainnointiin ja torjumiseen käytettävät menetelmät ja teknologiat

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2015

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Joonas Karjalainen

Yhteystiedot: joonas.of.karjalainen@student.jyu.fi

Ohjaaja: Anneli Heimbürger

Työn nimi: Sovellustason palvelunestohyökkäysten havainnointiin ja torjumiseen käytettävät menetelmät ja teknologiat

Title in English: Denial-of-service attack detection and prevention methods and technologies

Työ: Kandidaatintutkielma

Sivumäärä: 26+0

Tiivistelmä: Tutkielmassa tarkastellaan sovelluskerroksiin kohdistuvien palvelunestohyökkäysten tunnistamiseen ja torjumiseen käytettäviä menetelmiä ja teknologioita.

Avainsanat: internet, palvelunestohyökkäys, DoS, DDoS, tunnistaminen, torjuminen, sovellustaso

Abstract: This research reviews different methods for detecting, mitigating and countering application layer denial-of-service attacks.

Keywords: internet, denial-of-service, DoS, DDoS, mitigation, detection, countermeasures, application layer

Kuviot

Sisältö

1 Johdanto

Palvelunestohyökkäykset ovat viime vuosina olleet aika-ajoin otsikoissa, viimeisimpänä Suomessa Osuuspankin sekä Nordean palveluihin kohdistuneet hyökkäykset vuodenvaihteessa 2014–2015. Osuuspankki vastasi palvelunestohyökkäykseen rajoittamalla ulkomailta tulevaa verkkoliikennettä, jonka johdosta myös oikea ulkomailta tuleva liikenne kärsi ("OP-Pohjolan verkkopankin ongelmat ehkä tältä erää ohi" 2015). Osuuspankin tapauksessa kuitenkin osa palvelunestohyökkäyksestä tuli Suomen sisältä; tätä vastaan ulkomailta tulevan liikenteen rajoittaminen on jokseenkin tehotonta.

Computer emergency response team (CERT), joka on yksi tietoturva-alan johtavista tutkimuskeskuksista, määrittelee palvelunestohyökkäyksen sellaiseksi hyökkäykseksi, jonka tarkoitus on estää palveluun oikeutettujen käyttäjien pääsy palveluun ("Denial of Service Attacks" 1997). Palvelunestohyökkäys voi kohdistua joko palveluun jonka käyttö halutaan estää, suoraan käyttäjään jonka pääsy palveluun halutaan estää tai käyttäjien ja palvelun välissä sijaitsevaan kommunikaatiolinkkiin. Palvelunestohyökkäykset ovat ongelmallisia sekä kohteena olevien palveluiden tarjoajille että niiden käyttäjille. Palvelunestohyökkäys voi, toteutustavasta riippuen, aiheuttaa jopa laitevahinkoa kohteelleen (Mirkovic ja Reiher 2004, sivu 47). Vähintäänkin palvelunestohyökkäykset aiheuttavat sen, ettei palveluun tai järjestelmään pääse ulkopuolelta käsiksi. Tämä voi aiheuttaa käyttäjien tyytymättömyyttä palvelua kohtaan, mahdollisesti jopa rahallista haittaa palvelun tarjoajalle. Tämän vuoksi palvelunestohyökkäykset ovat ongelmallisia sekä palveluiden tarjoajille että käyttäjille, ja ne olisi syytä torjua mahdollisuuksien mukaan.

Erityisesti sovellustason palvelunestohyökkäykset ovat hyvin ajankohtaisia. Perinteisesti palvelunestohyökkäykset ovat olleen lähinnä verkkotasoon kohdistuvia. Vuonna 2010 "Anonymous"-nimellä itseään kutsuva joukko kuitenkin hyökkäsi Mastercardin, PayPalin, Visan sekä PostFinancen verkkosivuja vastaan ("Operation Payback cripples MasterCard site on revenge for WikiLeaks ban" 2010) sovellustason palvelunestohyökkäyksillä. Akamain, joka on yksi maailman johtavista pilvipalvelujen tarjoajista, tytäryhtiön Prolexicin mukaan vuoden 2014 toisen neljänneksen aikana 11% kaikista palvelunestohyökkäyksistä oli sovellustason palvelunestohyökkäyksiä ("Q2 2014 Global DDoS Attack Stats" 2014). Yleisim-

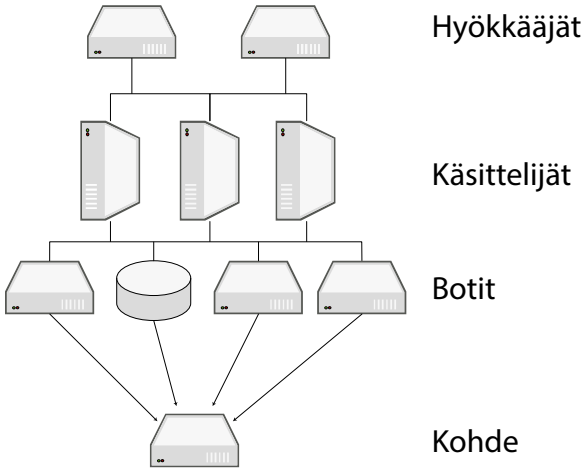
piä sovellustason palvelunestohyökkäyksiä olivat HTTP GET -hyökkäykset (7.5% kaikista hyökkäyksistä) ja HTTP POST -hyökkäykset (2.3% kaikista hyökkäyksistä). Vuoden 2014 kolmannen vuosineljänneksen raportissa ilmoitettiin sovellustason palvelunestohyökkäysten lukumäärän nousseen 2% edellisestä vuosineljänneksestä ("Akamai PLXsert's Q3 2014 State of the Internet - Security Report Released" 2014). Saman verran nousua oli havaittavissa myös verkkotason palvelunestohyökkäysten lukumäärässä. Viimeisellä vuosineljänneksellä raportoitiin sovellustason palvelunestohyökkäysten lukumäärän nousseen 51% edellisestä vuosineljänneksestä, verkkotason palvelunestohyökkäysten lukumäärän nousseen 121% edellisestä sekä kaikkien palvelunestohyökkäysten lukumäärän nousseen 90% ("Q4 2014 State of the Internet - Security Report" 2014). Vaikka suurin osa palvelunestohyökkäyksistä toteutetaankin yhä verkkotason palvelunestohyökkäyksillä, sovellustason palvelunestohyökkäykset on syytä ottaa vakavasti.

Tutkimus on toteutettu kuvailevana kirjallisuuskatsauksena. Tutkimuksessa perehdytään erilaisiin sovellustason palvelunestohyökkäyksiin sekä niiden tunnistamiseen ja torjumiseen käytettäviin menetelmiin ja teknologioihin. Aivan aluksi luvussa 2 tutustutaan erilaisiin sovellustason palvelunestohyökkäyksiin. Luvussa 3 perehdytään palvelunestohyökkäysten tunnistamiseen, luvussa 4 puolestaan perehdytään olemassa oleviin palvelunestohyökkäysten torjumiseen tähtääviin menetelmiin ja teknologioihin. Tutkielmassa ei arvioida sitä, ovatko tunnistamiseen ja torjumiseen käytettävät menetelmät ja teknologiat realistisesti implementoitavissa järjestelmiin.

2 Palvelunestohyökkäykset

Palvelunestohyökkäys ja hajautettu palvelunestohyökkäys ovat tunnistamisen ja torjumisen kannalta jokseenkin sama asia. Ainoa eroavaisuus näissä kahdessa on niiden skaala: ”tavalliset” palvelunestohyökkäykset toteutetaan suhteellisen pienellä määrällä hyökkääviä koneita, mutta hajautetuissa palvelunestohyökkäyksissä hyökkäykseen voidaan valjastaa tuhansia tai kymmeniä tuhansia boteiksi kutsuttuja koneita.

Hajautetuissa palvelunestohyökkäyksissä itse liikenne generoidaan hyökkääjien valjastamien bottien avulla. Tietokonevirusten ja muiden haittaohjelmien avulla hyökkääjät voivat rakentaa käyttöönsä bottiverkon, jota hyökkääjät kontrolloivat epäsuorasti käsittelijäkoneiden avulla. Kuviossa 1 on karkeasti esitetty bottiverkon rakenne. Bottiverkot on kategorisoitu kolmeen pääryhmään niiden kontrollointijärjestelmän mukaan: IRC-pohjaisiin, Web-pohjaisiin sekä vertaisverkkopohjaisiin bottiverkkoihin (Zargar, Joshi ja Tipper 2013). Tässä työssä ei perehdytä eri ryhmien eroavaisuuksiin.

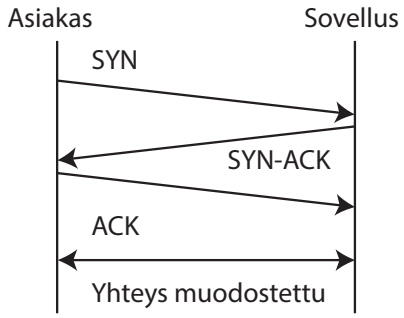


Kuvio 1. Bottiverkon elementit. Muokattu lähteestä (Zargar, Joshi ja Tipper 2013).

Sovellustason palvelunestohyökkäyksissä hyökkääjän bottiverkkoon kuuluvat koneet muodostavat kohteena olevaan palvelimeen TCP-yhteyden (kuvio 2), jonka kautta palvelimelle lähetetään HTTP-protokollan mukaisia valideja pyyntöjä. Pyynnöt voivat olla GET-pyyntöjä, POST-pyyntöjä tai mitä tahansa sellaisia pyyntöjä, jotka kohteena oleva palvelin hyväksyy. Sovellustason palvelunestohyökkäyksiä yhdistävä tekijä on se, että lähetettävät paketit ovat

periaatteessa oikeita paketteja sisältönsä puolesta. Myös paketteihin sisällytetyt IP-osoitteet ovat pakettien lähettäjien oikeita IP-osoitteita, sillä muutoin TCP-yhteyttä palvelimeen ei olisi mahdollista pitää auki.

Toisin kuin verkkotason palvelunestohyökkäyksissä, sovellustason palvelunestohyökkäyksissä olennaista ei ole muodostuvan hyökkäysliikenteen tukahduttavan suuri määrä. Sovellustasolle hyökättäessä keskitytäänkin itse palvelimen resursseihin: hyökkäyksellä pyritään kuluttamaan palvelimen sovellustason resurssit (muisti, prosessoriaika) loppuun.



Kuvio 2. Kolmivaiheinen TCP-kättely.

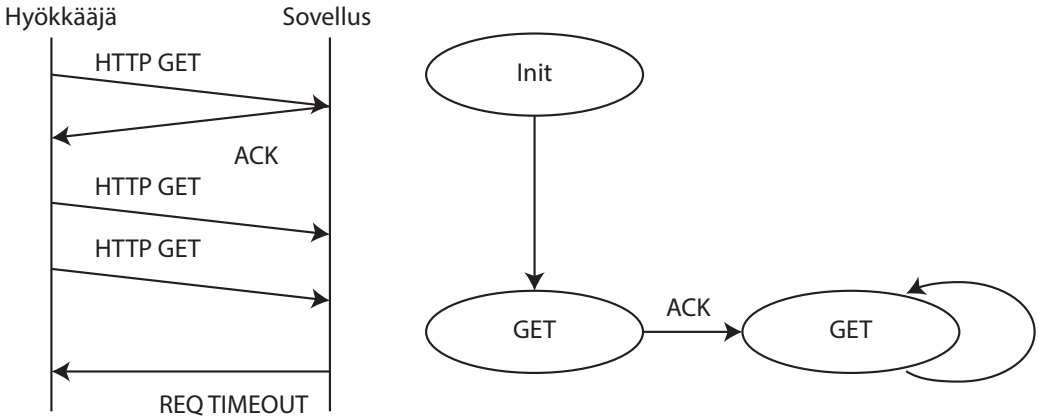
Vaihtoehtoisesti sovellustason palvelunestohyökkäys voi kohdistua jonkin palvelimen käyttämän sovelluksen haavoittuvuuteen. Haavoittuvuuksia hyväksikäyttävät hyökkäykset kuitenkin muuttuvat haavoittuvuuksien ilmentymisen ja paikkauksen mukaisesti, joten haavoittuvuuksia hyväksikäyttäviin hyökkäyksiin ei tässä työssä paneuduta.

Tässä luvussa tutustutaan erilaisiin palvelunestohyökkäystapoihin. HTTP GET -hyökkäysten ja hitaiden hyökkäysten lisäksi sovelluskerroksella palvelunestohyökkäykset kohdistuvat käytettävissä järjestelmissä oleviin muihin haavoittuvuuksiin.

2.1 HTTP GET -hyökkäys

HTTP GET -hyökkäyksessä hyökkääjä generoi ja lähettää suuren määrän valideja HTTP GET -pyyntöjä kohteena olevalle palvelimelle. Hyökkäävä osapuoli ei välitä palvelimen vastauksista, vaan vain lähettää jatkuvasti uusia GET-pyyntöjä kuvion 3 mukaisesti. HTTP GET -hyökkäyksissä ei ole tarvetta suurelle määrälle hyökkääviä botteja sillä jokainen hyökkäykseen osallistuva botti voi generoida suuren määrän valideja pyyntöjä, usein yli 10 pyyntöä

sekunnissa (“Taxonomy of DDoS Attacks” 2014). HTTP GET -hyökkäyksessä voidaan käyttää hyväksi HTTP 1.1 -version ominaisuutta, joka sallii useamman yhtäaikaisen pyynnön lähettämisen yhden istunnon aikana (“Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing” 2014, luku 6.1). Vaihtoehtoisesti hyökkääjä voi lähettää kohteena olevaan palvelimeen paketteja, jotka sisältävät useampia HTTP GET -pyyntöjä.



Kuvio 3. HTTP GET -hyökkäyksen viestijärjestys sekä tilakone hyökkääjän toiminnasta. Muokattu lähteestä (Dantas, Nigam ja Fonseca 2014).

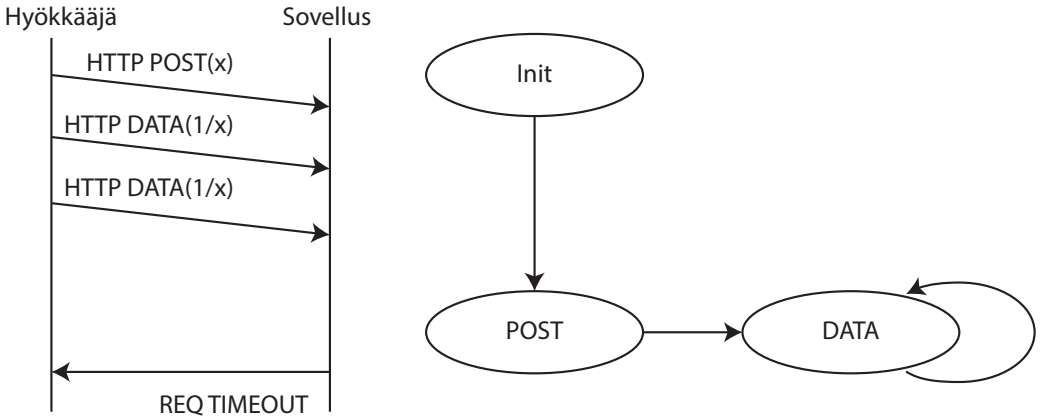
Koska saman sisällön pyytäminen uudestaan ja uudestaan GET-pyynnöllä on kohtalaisen helposti havaittavissa (luku 3.1), HTTP GET -hyökkäyksen voi toteuttaa lähettämällä pyyntöjä rekursiivisesti tai satunnaisrekursiivisesti. Rekursiivisessa hyökkäyksessä hyökkääjä kerää palvelimen sivuista, kuvista ja muista tiedostoista polun, jota hyökkäyksessä käytettävät botit noudattavat. Satunnaisrekursiivinen GET-hyökkäys puolestaan on suunniteltu foorumeita ja muita numeroivia järjestelmiä käyttäviä palveluita varten. Koska useimmin foorumeiden keskustelujen sivut ja uutissivustojen artikkelit ovat peräkkäisin luvuin indeksoitu, hyökkäävään pyyntöön on riittävää sisällyttää satunnainen luku palvelimen hyväksymältä lukuväliltä. Tällöin jokainen GET-pyyntö on toisesta eroava ja myös mahdollisesti vaikeammin havaittavissa.

2.2 Hitaat hyökkäykset

Hitaisiin hyökkäyksiin kuuluvat Slow POST ja Slow READ -hyökkäykset. Hitaissa hyökkäyksissä, kuten myös HTTP GET -hyökkäyksessä, tarkoituksena ei ole häiritä itse verkkoa

suurella datamäärällä, vaan sitoa palvelimen resurssit siten ettei palvelimen ole mahdollista palvella oikeita käyttäjiä.

Slow HTTP POST -hyökkäyksessä hyökkääjä käyttää hyväksi HTTP-protokollan POST-pyyntöä. POST-pyynnössä palvelimelle välitetään viesti välitettävästä sisällöstä, esimerkiksi tiedoston tai lomakkeen lähettämisestä palvelimelle. HTTP POST -hyökkäys toteutetaan siten, että palvelimelle lähetetään oikean tapainen HTTP POST -ylätunniste, jossa ”Content-Length”-kenttään on määritelty jokin suuri koko, jonka palvelin hyväksyy. Tämän jälkeen hyökkääjä lähettää varsinaista viestiä hyvin hitaalla tahdilla kuvion 4 mukaisesti, esimerkiksi tavun kahdessa minuutissa. Koska itse pyyntö ei ole erotettavissa oikeasta pyynnöstä ja palvelin vastaanottaa dataa kuten pyynnössä pyydettiin, palvelin pyrkii vastaanottamaan koko ”Content-Length”-kentässä määritellyn kokoisen viestin vastaan. Näin hyökkääjä saa tukittua palvelimen resursseja erittäin tehokkaasti (“Denial of Service Attacks: A Comprehensive Guide to Trends, Techniques and Technologies” 2012).



Kuvio 4. HTTP POST -hyökkäyksen viestijärjestys sekä tilakone hyökkääjän toiminnasta. Muokattu lähteestä (Dantas, Nigam ja Fonseca 2014).

Slow Read -hyökkäys on puolestaan käytännössä päinvastainen Slow HTTP POST -hyökkäykseen verrattuna: hyökkääjä luo suuren määrän yhteyksiä palvelimeen, joiden kautta hyökkääjä pyytää palvelimelta oikeutta ladata dokumenttia tai jotain muuta suurta tiedostoa. Kun tiedoston lataaminen on alkanut, hyökkäävät koneet alkavat hidastamaan kuittauspakettien lähettämistä hidastaen latausta (“Denial of Service Attacks: A Comprehensive Guide to Trends, Techniques and Technologies” 2012). Tämä sitoo palvelimen resursseja tehokkaasti,

sillä hyökkäävät koneet vaikuttavat vain olevan hitaan verkkoyhteyden päässä. Vaihtoehtoisesti hyökkääjät luovat yhteyden palvelimeen pientä viesti-ikkunakokoa mainostaen. Palvelin noudattaa tätä kokoa, eikä lähetä sitä suurempia paketteja. Tällöin hyökkääjä saa varsin tehokkaasti sidottua palvelimen resursseja vastaanottamalla hyvin pienejä viestejä. Koska hyökkäävät koneet ovat suorittaneet kolmivaiheisen TCP-kättelyn, palvelin pitää hyökkäyksessä käytettäviä istuntoja auki.

3 Palvelunestohyökkäysten havaitseminen

Koska sovellustason palvelunestohyökkäykset toteutetaan siten että lähetettävät paketit suorittavat kolmivaiheisen TCP-kättelyn, sovellustason palvelunestohyökkäyksissä käytettyjä vihamielisiä paketteja ei ole mahdollista tunnistaa samoin keinoin kuin verkkotason palvelunestohyökkäyksissä.

Sovellustasoon kohdistuvat palvelunestohyökkäyksiä on kohtuullisen hankala havainnoida niiden luonteen takia. Näiden hyökkäysten kohteena on yleensä jokin yksittäinen sovellus, jonka toiminnan häiriöityminen ei kuitenkaan vaikuta muiden sovellusten toimintaan, vaikeuttaen hyökkäyksen tunnistamista ulkopuolelta. Sovellustasoon kohdistuvien tulvahyökkäysten tunnistaminen on myös jokseenkin hankalaa, sillä oikeakin liikenne voi tulla ”yhtäkkisenä” tulvana – esimerkiksi julkimon Twitter-linkkauksen takia (“How Stephen Fry takes down entire websites with a single tweet” 2010). Tässä luvussa tutustutaan sovellustason palvelunestohyökkäysten havaitsemiseen pyrkiviin menetelmiin ja teknologioihin.

3.1 HTTP GET -hyökkäyksen tunnistaminen

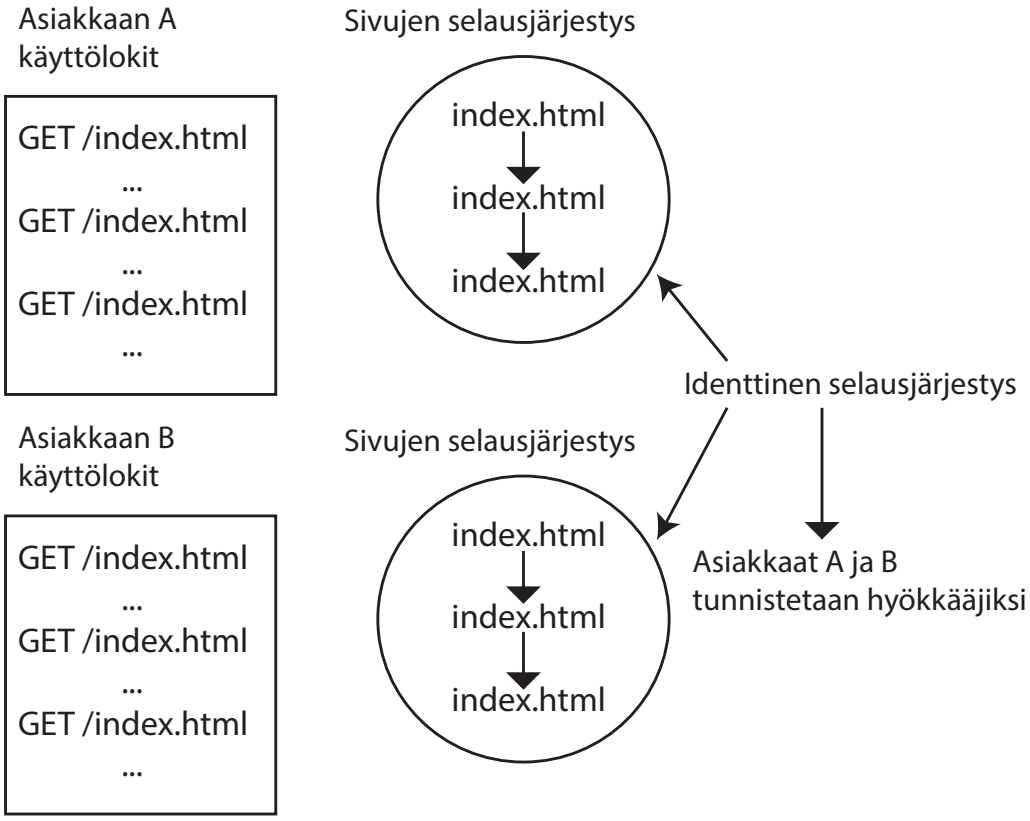
HTTP GET -tulvan tunnistaminen voi joissakin tapauksissa olla hyvinkin hankalaa, sillä HTTP GET -hyökkäyksen erottaminen palvelimen sisällön yhtäkkisen kasvun noususta on vaikeaa. Mikäli esimerkiksi palvelin joutuu näennäisesti hyökkäyksen kohteeksi siten, että jotakin palvelimella olevaa sisältö ladataan runsaasti, kyse voi kuitenkin olla hyökkäyksen sijaan siitä, että kyseistä sisältöä on yksinkertaisesti linkitetty jollekin suositulle foorumille.

Artikkelissa Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior (Yatagai, Isohara ja Sasase 2007) on ehdotettu kahta toisistaan eroavaa algoritmia HTTP GET -tulvan tunnistamiseen verkon yhdyskäytävässä.

3.1.1 Sivujen selausjärjestys

Algoritmeista ensimmäinen (algoritmi 1) keskittyy sivustojen selausjärjestykseen. Mikäli palvelin on palvelunestohyökkäyksen kohteena eikä hyökkääjä käytä satunnaisrekursiivis-

ta HTTP GET -hyökkäysmetodia, hyökkäävien tietokoneiden sisältöpyynnöt ovat keskenään identtisiä. Tällöin voidaan tallentaa analyysiin tarvittava määrä jokaisen IP-osoitteen selausjärjestystä. Tallennettuja järjestyksiä analysoidaan ja verrataan muiden IP-osoitteiden selausjärjestykseen etsien IP-osoitteet joiden selausjärjestys on identtistä (kuvio 5). Mikäli sellaisia löydetään voidaan päätellä kyseisten IP-osoitteiden osallistuvaan hyökkäykseen ja jättää IP-osoitteista tulevat GET-pyynnöt huomiotta.



Kuvio 5. Algoritmin 1 hyökkäyksentunnistusmalli. Muokattu lähteestä (Yatagai, Isohara ja Sasase 2007).

Esitelty menetelmä ei kuitenkaan ole täydellinen. Vaikka algoritmi ilmoittikin erheellisesti hyökkäyksiksi vain 1% oikeasta liikenteestä, algoritmi luokitteli virheellisesti tavalliseksi liikenteeksi jopa 22% hyökkäysliikenteestä. Oikeasta liikenteestä 1% hyökkäysliikenteeksi luokitteleminen on ymmärrettävää, sillä käyttäjät voivat pyytää sivuja samassa järjestyksessä kuin hyökkäyksessä käytettävät botit. Koska HTTP GET -hyökkäyksiin kuuluvat myös sivun uudelleenlataushyökkäykset sekä DoS-työkalujen avulla tehtävät hyökkäykset joissa pyydettävien sivujen järjestys on mahdollista määrittää itse, tällä algoritmilla ei näitä voida

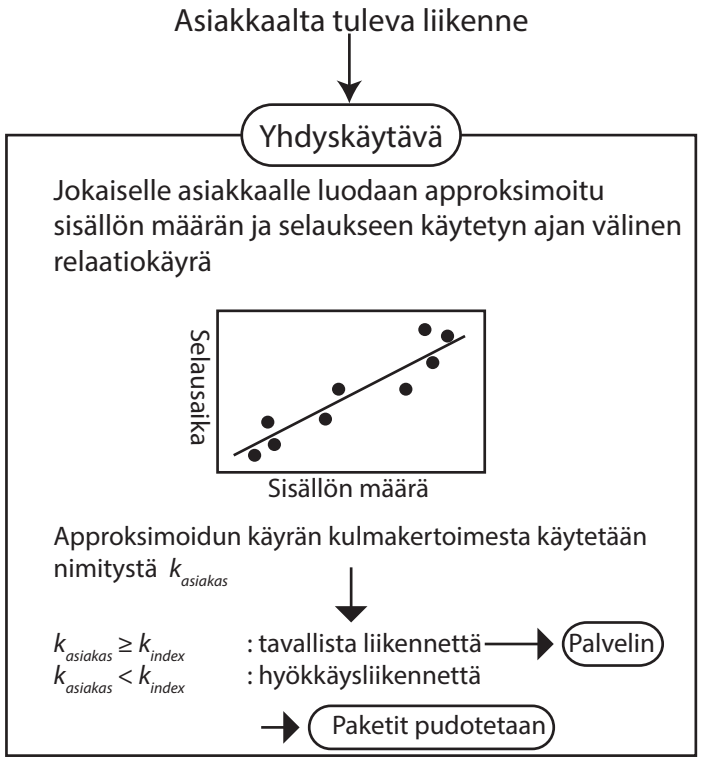
havaita.

3.1.2 Informaation määrän ja selausajan välinen korrelaatio

Esitetyistä metodeista (Yatagai, Isohara ja Sasase 2007) jälkimmäinen (algoritmi 2) puolestaan keskittyy sivun selaukseen käytettävän ajan ja sivun informaation väliseen korrelaatioon. Tavallisten käyttäjien selatessa sivuja ajatellaan, että sivun selaamiseen käytettävä aika kasvaa sivulla esitetyn informaation määrän kasvaessa. Esitetyssä metodissa tallennetaan jokaisen IP-osoitteen vierailemat sivut sekä sivuihin yhdistetyt selausajat. Näiden välinen suhde lasketaan ja tätä suhdetta verrataan laskettuun arvioon keskimääräisen palvelun käyttäjän selausajasta. Arvio palvelun käyttäjän selauskäyttäytymisestä esitetään käyränä k_{index} joka lasketaan tavallisen käyttäjän sivun selaamiseen käyttämää aikaa vertaamalla sivun sisältämien elementtien lukumäärään. Jokaiselle GET-pyyntöjä lähettävälle IP-osoitteelle lasketaan vastaava käyrä $k_{asiakas}$, jota sitten verrataan k_{index}-käyrään. Mikäli $k_{asiakas} \geq k_{index}$ niin liikenne on normaalia, mutta jos $k_{asiakas} < k_{index}$, liikenne tulkitaan HTTP GET -hyökkäykseksi kuten kuviossa 6.

Esitetyistä algoritmeista jälkimmäinen onnistuu kuitenkin tavallaan paremmin – HTTP GET -hyökkäyksistä tunnistamatta jäi 0%. Kaikki käyttäjät eivät ole täysin identtisiä vaan osa käyttäjistä saattaa olla vikkelämpiä käyttäymiseltään kuin keskimääräinen käyttäjä, jonka käyttäytymisen perusteella k_{index} on laskettu, jolloin algoritmi luokittelee vikkelien käyttäjien liikenteen hyökkäykseksi. Algoritmi luokitteleekin oikeasta liikenteestä jopa 10% virheellisesti hyökkäysliikenteeksi.

Molemmissa edellämainituista algoritmeista on omat heikkouksensa ja vahvuutensa. Mikäli oikeiden asiakkaiden palveleminen on tärkeintä ja palvelin kestää läpi pääsevän palvelunestohyökkäyksen aiheuttaman rasituksen, ensimmäinen esitetty algoritmi on käytännöllinen. Jos tilanne on toinen, hyökkäysten torjumisen ollessa tärkeintä, jälkimmäinen esitetyistä algoritmeista tekee tehtävänsä.



Kuvio 6. HTTP GET -tulvan tunnistaminen algoritmin 2 avulla. Muokattu lähteestä (Yatagai, Isohara ja Sasase 2007).

3.2 Hitaiden hyökkäysten havaitseminen

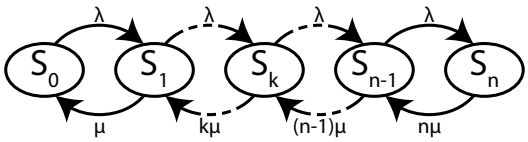
Hitaiden hyökkäysten tunnistaminen perustuu hyvin vahvasti tilastollisiin menetelmiin. Konferenssijulkaisussa Method of Slow-Attack Detection (Duravkin, Loktionova ja Carlsson 2014) Duravkin ym. ehdottavat palvelimen olevan mahdollista esittää M/M/n-tyyppisenä jonotusjärjestelmänä, sillä HTTP-pyyntöjen parametrit (pituus, datan vastaanottonopeus, vastaanotetun datan koko, kuittausviive) ovat vakioita. Palvelimen pyynnönkäsittelyä tarkasteltaessa jonotusjärjestelmänä voidaan ottaa huomioon palvelimen mahdollisuus siirtyä ylikuormitettuun tilaan. Koska palvelimen liikenteen puskurointi vaikuttaa ainoastaan palvelunestohyökkäyksen kestoon eikä palvelimen saatavuuteen hyökkäyksen aikana, palvelimen voi määritellä olevan jonoton jonotusjärjestelmä (kuvio 7).

Palvelimelle luodaan tilamalli, jonka tiloina ovat

$$S_0 = 0 \text{ pyyntöä käsiteltävänä,}$$
$$S_1 = 1 \text{ pyyntö käsiteltävänä,}$$
$$S_k = k \text{ pyyntöä käsiteltävänä,}$$
$$S_{n-1} = n-1 \text{ pyyntöä käsiteltävänä,}$$
$$S_n = n \text{ pyyntöä käsiteltävänä, palvelin on ylikuormittunu.}$$

Tilamallin parametrit ovat

$$n - \text{ suurin sallittu määrä samanaikaisia HTTP-pyyntöjä,}$$
$$k - \text{ nykyinen määrä samanaikaisia HTTP-pyyntöjä,}$$
$$\lambda - \text{ HTTP-pyyntöjen saapumisnopeus,}$$
$$\mu - \text{ HTTP-pyyntöjen käsittelyn intensiteetti.}$$



Kuvio 7. Palvelimen tilamuutokset. Muokattu lähteestä (Duravkin, Loktionova ja Carlsson 2014).

Luku n on suurin palvelimen käsiteltävissä oleva määrä samanaikaisia pyyntöjä. Tilasiirtymien välillä siirryttäessä intensiteetti määräytyy vastaanotettavien pyyntöjen intensiteetin mukaan. Kun palvelin ei ole hyökkäyksen kohteena, avattavien ja suljettavien yhteyksien keskimääräinen suhde on tasapainossa – tällöin todennäköisyys sille, että palvelin siirtyisi seuraavaksi tilaan S_n lähestyy nollaa. Hyökkäyksen iskiessä palveluun pyyntöjen käsittelyaika ja avattavien yhteyksien intensiteetti kasvavat, jolloin tilakone siirtyy väistämättä lähemmäs tilaa S_n. Tilojen välisille siirtymille on mahdollista laskea todennäköisyyksiä ja näiden todennäköisyyksien avulla on mahdollista laskea kulloisellekin tilalle ylikuormittuneeseen tilaan siirtymiseen menevä aika.

Duravkin ym. esittämä menetelmä onkin erinomainen hitaiden palvelunestohyökkäysten tunnistamiseen. Menetelmällä voidaan ajoissa tunnistaa palvelimen ollessa palvelunestohyökkäyksen kohteena, jolloin palvelunestohyökkäyksen torjuminen on mahdollista aloittaa jo aikaisessa vaiheessa.

4 Palvelunestohyökkäysten torjuminen

Koska sovellustason palvelunestohyökkäykset eroavat kovasti toisistaan, myös niiden torjuminen on suurimmaksi osaksi hyvinkin eroavaista. Tästä poikkeuksena on kuitenkin SeVen-algoritmi joka saa nimensä Open Systems Interconnection -mallin (OSI-malli) seitsemännestä tasosta, sovellustasosta. SeVen-algoritmi on muista esitellyistä menetelmistä sikäli poikkeuksellinen että sen avulla on mahdollista torjua sekä HTTP GET -hyökkäyksiä, HTTP POST -hyökkäyksiä että HTTP PRAGMA -hyökkäyksiä joihin tässä työssä ei perehdytä.

HTTP GET -hyökkäysten torjumiseen esitellään kaksi eri metodia SeVenin lisäksi: Client-puzzle -protokollat sekä Completely Automated Public Turing test to tell Humans and Computers Apart -testit (CAPTCHA-testit). Tässä luvussa tutustutaan sovellustason palvelunestohyökkäysten torjumiseen käytettäviin menetelmiin ja teknologioihin.

4.1 SeVen

Konferenssijulkaisussaan A Selective Defense for Application Layer DDoS Attacks Dantas ym. (Dantas, Nigam ja Fonseca 2014) esittelevät lupaavan torjumisalgoritmin sovellustason palvelunestohyökkäyksiä vastaan. Esitelty algoritmi, SeVen, perustuu Adaptive Selective Verification (ASV) -torjumisemenetelmään (Khanna ym. 2012) jota käytetään verkkotason palvelunestohyökkäysten torjumiseen. ASV-torjumismenetelmä ei itsessään sovellu sovellustason palvelunestohyökkäysten torjumiseen, sillä tämä torjumismenetelmä olettaa hyökkääjän ja palvelimen välisen kommunikaation olevan tilatonta SYN-ACK -kommunikaatiota sovellustason – kuten HTTP:n – kommunikaation ollessa tilallista.

Kuten ASV, myöskään myöskään SeVen ei välittömästi prosessoi saapuvia pyyntöjä. Tämän sijasta SeVen kerää viestejä sisäiseen puskuriin ajan t_s pituisen kierroksen ajan. SeVen koostuu kahdesta puskurista: puskurista $\mathscr{P}$ joka sisältää osittain käsitellyt pyynnöt sekä puskurista $\mathscr{R}$ joka sisältää vastaanotetut mutta käsittelemättömät pyynnöt. Yksinkertaisuuden takia ajatellaan puskurien $\mathscr{P}$ sekä $\mathscr{R}$ jokaisen elementin olevan $\langle id, N, T \rangle$-muotoa oleva tupletti jossa id on käsiteltävälle pyynnölle uniikki tunniste, T on pyynnön käsiteltävän datan määrä ja N puskurista riippuva luonnollinen luku. Puskurissa $\mathscr{P}$ N on jo käsitellyn datan mää-

rä, puskurissa $\mathscr{R}$ N puolestaan on vastaanotetun ja käsiteltävän datan määrä. Näiden lisäksi oletetaan $\mathscr{P}$ ja $\mathscr{R}$ olevan ylhäältä rajoitettu k:ssa: suurin määrä pyyntöjä jotka sovellus voi käsitellä kullakin ajan hetkellä. Puskurien lisäksi SeVen sisältää laskurin `PMod`.

Koska voidaan olettaa sovelluksen voivan käsitellä k pyyntöä kullakin hetkellä, SeVen ei hylkää yhtäkään pyyntöä kierroksen t_s aikana niin kauan kuin saapuneiden pyyntöjän määrä on pienempi kuin k. Mikäli kuitenkin puskuri $\mathscr{R}$ sisältää jo k pyyntöä ja vastaanottaa uuden pyynnön, on päätettävä tiputetaanko uusi pyyntö vai lisätäänkö pyyntö puskuriin $\mathscr{R}$ jonkin toisen pyynnön tilalle.

Uudelle pyynnölle r suoritettava toiminto päätetään siten, että puskurin $\mathscr{P}$ ollessa täynnä eli puskurin elementtien lukumäärän ollessa k asetetaan $\texttt{PMod} := \texttt{PMod} + 1$. Jos saapunut pyyntö ei ole jatkumoa jo aiemmin vastaanotetulle pyynnölle on päätettävä tiputetaanko pyyntö vai lisätäänkö se puskuriin. SeVen generoi satunnaisluvun *rand*. Mikäli $rand \leq \texttt{Prob}$ (jossa `Prob` kuten esitetty alhaalla), pyyntö r lisätään puskuriin – muussa tapauksessa pyyntö tiputetaan.

$$\texttt{Prob} = \frac{k}{k + \texttt{PMod}}$$

Jos pyyntö tiputetaan, ilmoitetaan *id*:n edustajalle ettei pyyntöä suoriteta ja puskurit $\mathscr{P}$ ja $\mathscr{R}$ säilyvät muuttumattomina. Jos pyyntö kuitenkin hyväksytään, päätetään tiputettava pyyntö tasajakauman mukaisesti ja ilmoitetaan pyynnön tiputtamisesta *id*:n edustajalle.

Kun kierros on päättynyt, suoritetaan $\mathscr{R}$:n sisältämät pyynnöt, asetetaan $\texttt{PMod} := 0$, siirretään suoritetut pyynnöt $\mathscr{R}$:stä $\mathscr{P}$:hen ja poistetaan aiemmin suoritetut pyynnöt puskurista $\mathscr{P}$. Jos suoritetty pyyntö on puskurin viimeinen elementille *id*, pyyhitään pyyntö molemmista puskureista $\mathscr{P}$ ja $\mathscr{R}$ sekä lähetetään kuittausviesti jokaiselle *id*, jonka lähettämä pyyntö on kierroksen aikana suoritettu. Metodia käyttäessä olisi mahdollista käyttää *id*:n sisältämää dataa (esimerkiksi IP-osoitetta, sitä kautta saatavia sijaintitietoja tai vastaanottoaikaa) todennäköisyysjakaumien rakentamiseen pyyntöjen kohtalon päättämiseksi – esimerkiksi siten, että pitkään puskurissa olleilla pyynnöillä on suurempi todennäköisyys tulla tiputetuksi.

Dantas ym. (Dantas, Nigam ja Fonseca 2014) testasivat SeVen-algoritmia Monte Carlo -simulaatiolla Maudea (Clavel ym. 2007) käyttämällä. Testeissä palvelimen aikakatkaisu sekä kierroksen aika olivat 0,4 sekuntia, asiakkaiden maksimimäärä 200, verkon viive 0,1 sekuntia ja simulaation kestoiksi asetettiin 40 sekuntia. Hyökkääjiä luotiin 120 sekunnissa, ku-

ten myös oikeita asiakkaita. Hyökkääjien käyttäytyminen määriteltiin kuvion 3 mukaiseksi HTTP GET -hyökkäykselle ja kuvion 4 mukaiseksi HTTP POST -hyökkäykselle.

Testeissä huomattiin, että SeVen-algoritmia käyttämällä saadaan oikeita asiakkaita palveltua hyökkääjien lukumäärästä riippumatta suhteellisen tehokkaasti – yhtäaikaisten hyökkääjien lukumäärän ollessa 280 algoritmia käyttävät sovellukset pystyivät palvelemaan yli 70% oikeista asiakkaista. Tämän lisäksi huomattiin sovelluksen puskurin, $\mathscr{P}$, ylärajan vaikuttavan asiakkaiden palvelemiseen tiettyyn pisteeseen asti positiivisesti. Kun hyökkääjiä oli 280 ja puskurin koko 6 pyyntöä, alle 60% asiakkaista saatiin palveltua onnistuneesti. Kun puskurin ylärajan koko kasvatettiin 18 pyyntöön, SeVen-algoritmin avulla saatiin palveltua yli 80% oikeista käyttäjistä – puskurin kokoa kasvattaessa 36 pyyntöön asiakkaista saatiin onnistuneesti palveltua jopa 90%. Asiakkaiden onnistuneen palvelemisen määrän kasvattaminen 10 prosenttiyksiköllä ei kuitenkaan ole järin järkevää, sillä resurssit (puskurin koko) olisi käytännössä kaksinkertaistettava.

4.2 HTTP GET -hyökkäyksen torjuminen

HTTP GET -hyökkäysten torjumiseen on useita eri vaihtoehtoja. Luvussa 4.1 esitelty SeVen-algoritmi toimii erinomaisesti GET -hyökkäysten torjumiseen – johtuen siitä ettei tätä algoritmia käyttävä sovellus välittömästi prosessoi vastaanottamiaan pyyntöjä vaan laittaa ne puskuriin odottamaan kierroksen loppumista, sovellus voi suorittaa korkeintaan $\mathscr{P}$:n ylärajan määrän pyyntöjä kunkin kierroksen lopuksi.

Mainitun SeVen-algoritmin lisäksi on olemassa myös muita HTTP GET -hyökkäysten torjumiseen soveltuvia menetelmiä. Seuraavaksi tarkastellaan kahta torjumismenetelmää: client-puzzle -protokollia sekä CAPTCHA-testejä.

4.2.1 Client-puzzle -protokollat

Fallahin artikkelissa (Fallah 2008) sekä Michalas ym. konferenssijulkaisussa (Michalas ym. 2010) esitellyt client-puzzle -protokollat vaativat jokaista asiakasta ratkaisemaan matemaattisen pulman ennen kuin palvelin hyväksyy yhteyden avaamisen. Palvelin lähettää ensin asiakkaalle pulman, jonka asiakas ratkaistuaan palauttaa palvelimelle. Palvelin joko hyväk-

syy tai hylkää ratkaisun. Mikäli palvelin hyväksyy palautetun ratkaisun, asiakkaan yhdistämispyyntö hyväksytään.

Client-puzzle -protokollien puolustuskäytössä on omat ongelmansa. Pulmien, jotka asiakas tulisi ratkaista, tulisi olla suhteellisen yksinkertaisia ratkaisemisen kuitenkin vaatiessa asiakkaalta resursseja – kaikki laskemiseen käytettävät resurssit ovat itse palvelunestohyökkäyksestä pois. Mikäli palvelunestohyökkäysten torjuminen halutaan tehokkaasti torjua client-puzzle -protokollilla, ratkaistavien pulmien tulisi olla suhteellisen haastavia asiakkaille. Tämä kuitenkin hankaloittaa palvelun käyttämistä sellaisille oikeille asiakkaille joilla ei ole juurikaan ylimääräistä laskentatehoa käytettävissä, kuten esimerkiksi mobiiliasiakkaille. Haastavien pulmien avulla puolustautuminen saattaa myös karkottaa asiakkaita – haastavan pulman ratkaisemiseen voi kulua jopa sekunteja ja oikeiden asiakkaden aika on arvokkaampaa kuin aika, joka pulman ratkaisemiseen hyökkääjältä vaaditaan.

Myös protokollat itse voivat joutua palvelunestohyökkäysten kohteeksi: joko pulmien luominen tai ratkaisujen oikeellisuuden tarkastaminen voi vaatia palvelimelta salausmerkkilaskentaa joka vaatii palvelimelta suoritusaikaa. Hyökkääjät voivat pyytää palvelinta lähettämään pulman ratkaistavaksi ja lähettää palvelimelle itse generoimansa vastauksen jonka oikeellisuudella ei ole hyökkääjän kannalta väliä.

4.2.2 CAPTCHA

Completely Automated Public Turing test to tell Computers and Humans Apart -testit (CAPTCHA-testit) (Ahn, Blum ja Langford 2004) ovat käytännössä Turingin testejä: arvioija yrittää päätellä, onko testiin vastaava ihminen vai ei. Toisin kuin Turingin testissä, arvioijana toimii tietokone, joka myös luo testit automaattisesti. CAPTCHA-testit luodaan jokseenkin satunnaisiksi – CAPTCHA-testi saattaa sisältää vääristynyttä tekstiä, taustakohinaa, tekstin tai kirjainten rinnastusta tai animaatiota. Tämä johtaa siihen, että vaikka koneellinen kuvantunnistus onkin varsin kehittynyttä, jopa siihen pisteeseen asti että sen avulla pystytään tunnistamaan kirjaimia, testit ovat kuitenkin hyökkääjien erittäin vaikeasti ratkaistavissa.

Jos CAPTCHA-testillä halutaan estää HTTP GET -palvelunestohyökkäys, testien on oltava turvallisesti implementoitu järjestelmään. Liian helppo CAPTCHA on mahdollista murtaa

optista merkintunnistusjärjestelmää käyttäen, tai palvelimen uudelleenkäyttäessä CAPTCHA-testejä hyökkääjä voi käyttää sellaisen CAPTCHA-testin istuntotunnistetta jonka ratkaisu on tiedossa. Turvallisesti implementoitu CAPTCHA-testi ei myöskään salli useampia yrityksiä testin ratkaisemiseksi.

Google onkin kehittänyt CAPTCHA-testien pohjalta oman reCAPTCHA-järjestelmänsä ("Google reCAPTCHA" 2009). Järjestelmä on sikäli turvallinen ettei se salli useampia yrityksiä testin ratkaisemiseksi eikä myöskään kierrätä testejä. Näiden lisäksi reCAPTCHA onnistuu olemaan merkintunnistusjärjestelmien ratkaisemattomissa, sillä reCAPTCHA käyttää ratkaistavina teksteinään sellaisia kirjojen tai karttojen tekstejä joita merkintunnistusjärjestelmät eivät ole saaneet ratkaistua.

Valitettavasti testien tekeminen vaikeaksi johtaa myös siihen että niistä tulee vaikeasti ratkaistavia jopa ihmisille. Lisäksi CAPTCHA-testien ratkaiseminen vaatii oikeilta asiakkailta aikaa ja vaivaa, mutta niillä saadaan taisteltua tehokkaasti sovellustason palvelunestohyökkäyksiä vastaan.

4.3 Hitaiden hyökkäysten torjuminen

Slow POST -hyökkäysten torjumiseen on esitelty menetelmä edellä, luvussa 4.1. Koska Slow Read -hyökkäyksessä luetaan sisältöä siten, että kerralla vastaanotettavan datan koko on hyvin pieni, hyökkäykset voidaan estää yksinkertaisesti olemalla hyväksymättä sellaisia yhteyksiä, joiden viesti-ikkunan koko on hyvin pieni. Hyökkäyksiä voi torjua myös poistamalla HTTP:n keep-aliven käytöstä ja rajoittamalla jokaisen yhteyden kokonaiskestoa. Luonnollisesti etenkään viimeisin Slow READ -hyökkäyksen torjumiskeino ei ole järkevä, mikäli palvelin sisältää suuria tiedostoja tai yhteyden avoinna pitäminen on palvelun kannalta olennaista.

Aiemmin luvussa 3.2 esiteltiin menetelmä jolla on mahdollista tunnistaa palvelimen olevan hitaan palvelunestohyökkäyksen kohteena. Yllä mainitut torjumismenetelmät voisivat toimia erinomaisesti Slow Read -hyökkäysten torjumiseen yhdistettynä niiden tunnistamisen kanssa.

5 Johtopäätökset

Kaikkia palvelunestohyökkäyksiä ei ole mahdollista täysin torjua: esimerkiksi sovelluksista löydetään uusia haavoittuvuuksia, joita hyökkääjät voivat hyödyntää, varsin nopeaan tahtiin. Usein haavoittuvuudet saadaan korjattua pikaisestikin, mutta näitä haavoittuvuuksia on silti saatettu jo hyökkäyksissä hyödyntää. Kaikkia palvelunestohyökkäyksiä ei ole täydellä varmuudella mahdollista havaita, mutta hyökkäysten tunnistamisen suhteen kehitystä kuitenkin tapahtuu ja ainakin osa palvelunestohyökkäyksistä onnistutaan tunnistamaan ja estämään kohtuullisen suurella varmuudella.

Erityisen mielenkiintoisiksi menetelmiksi osoittautuivat Duravkin ym. esittelemä hitaiden hyökkäysten tunnistamiseen käytettävä menetelmä sekä Dantas ym. esittelemä, useampaa sovellustason palvelunestohyökkäystyyppiä vastaan toimiva SeVen-torjumisalgoritmi. Vaikka hitaiden hyökkäysten tunnistamiseen käytettävä menetelmä ei osaakaan tunnistaa hyökkääjiä oikeista asiakkaista, sillä voidaan tunnistaa palvelimen ylipäätään olevan hyökkäyksen kohteena aikaisessa vaiheessa. SeVen-algoritmi puolestaan osoittautui mielenkiintoiseksi sen yleiskäyttöisyyden takia. Vaikka algoritmia kokeiltiinkin vain simulaatioiden avulla (Dantas, Nigam ja Fonseca 2014), sillä saadut tulokset olivat erittäin lupaavia. Hyvin positiivisten tulosten lisäksi algoritmin avulla saatiin torjuttua GET-hyökkäysten lisäksi erilaisia hitaita hyökkäyksiä.

Aivan täydellisiä menetelmät palvelunestohyökkäysten tunnistamiseen eivät ole. Kuitenkin menetelmillä saadaan tunnistettua palvelunestohyökkäykset, ja vielä oleellisemmin hyökkäyksiin osallistuvat asiakkaat, kohtuullisen varmasti. Myöskään palvelunestohyökkäysten torjumismenetelmät eivät ole täysin varmoja, mutta niiden tunnistamistodennäköisyydet ovat kohtuullisen suuret.

Lähteet

Ahn, Luis von, Manuel Blum ja John Langford. 2004. “Telling humans and computers apart automatically”. *Communications of the ACM* 47 (2): 56–60. doi:10.1145/966389.966390.

“Akamai PLXsert’s Q3 2014 State of the Internet - Security Report Released”. 2014. Viitattu 27. huhtikuuta 2015. https://blogs.akamai.com/2014/10/akamai-plxserts-q3-2014-state-of-the-internet---security-report-released.html.

Clavel, Manuel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer ja Carolyn Talcott. 2007. *All About Maude - A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic.* Berlin: Springer Berlin Heidelberg.

Dantas, Y.G., V. Nigam ja I.E. Fonseca. 2014. “A Selective Defense for Application Layer DDoS Attacks”. Teoksessa *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint,* 75–82. IEEE, syyskuu. ISBN: 978-1-4799-6363-8. doi:10.1109/JISIC.2014.21.

“Denial of Service Attacks”. 1997. Viitattu 24. helmikuuta 2015. http://www.cert.org/information-for/denial_of_service.cfm.

“Denial of Service Attacks: A Comprehensive Guide to Trends, Techniques and Technologies”. 2012. Viitattu 23. huhtikuuta 2015. https://www.imperva.com/docs/HII_Denial_of_Service_Attacks-Trends_Techniques_and_Technologies.pdf.

Duravkin, Ievgen, Anastasiya Loktionova ja Anders Carlsson. 2014. “Method of Slow-Attack Detection”. Teoksessa *Infocommunications Science and Technology, 2014 First International Scientific-Practical Conference Problems of,* 171–172. IEEE, lokakuu. ISBN: 978-1-4799-7342-2. doi:10.1109/INFOCOMMST.2014.6992341.

Fallah, Mehran. 2008. “A Puzzle-Based Defense Strategy Against Flooding Attacks Using Game Theory”. *Dependable and Secure Computing, IEEE Transactions on* 7 (1): 5–19. doi:10.1109/TDSC.2008.13.

“Google reCAPTCHA”. 2009. Viitattu 27. huhtikuuta 2015. https://www.google.com/recaptcha/.

“How Stephen Fry takes down entire websites with a single tweet”. 2010. Viitattu 23. huhtikuuta 2015. http://www.techradar.com/news/internet/how-stephen-fry-takes-down-entire-websites-with-a-single-tweet-674170.

“Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing”. 2014. Viitattu 27. huhtikuuta 2015. https://tools.ietf.org/html/rfc7230.

Khanna, S., S.S. Venkatesh, O. Fatemieh, F. Khan ja C.A. Gunter. 2012. “Adaptive Selective Verification: An Efficient Adaptive Countermeasure to Thwart DoS Attacks”. *Networking, IEEE/ACM Transactions on* 20 (3): 715–728. doi:10.1109/TNET.2011.2171057.

Michalas, Antonis, Nikos Komninos, Neeli R. Prasad ja Vladimir A. Oleshchuk. 2010. “New Client Puzzle Approach for DoS Resistance in Ad hoc Networks”. Teoksessa *Information Theory and Information Security (ICITIS), 2010 IEEE International Conference on,* 568–573. IEEE, joulukuu. ISBN: 978-1-4244-6942-0. doi:10.1109/ICITIS.2010.5689528.

Mirkovic, Jelena, ja Peter Reiher. 2004. “A taxonomy of DDoS attack and DDoS defense mechanisms”. *ACM SIGCOMM Computer Communication Review* 34 (2): 39–53. doi:10.1145/997150.997156.

“Operation Payback cripples MasterCard site on revenge for WikiLeaks ban”. 2010. Viitattu 3. maaliskuuta 2015. http://www.theguardian.com/media/2010/dec/08/operation-payback-mastercard-website-wikileaks.

“OP-Pohjolan verkkopankin ongelmat ehkä tältä erää ohi”. 2015. Viitattu 15. helmikuuta 2015. http://yle.fi/uutiset/op-pohjolan_verkkopankin_ongelmat_ehka_talta_eraa_ohi/7721795.

"Q2 2014 Global DDoS Attack Stats". 2014. Viitattu 27. huhtikuuta 2015. http://www.prolexic.com/knowledge-center-ddos-attack-report-2014-q2-quarterly-trends-infographic.html.

"Q4 2014 State of the Internet - Security Report". 2014. Viitattu 27. huhtikuuta 2015. http://www.stateoftheinternet.com/resources-web-security-2014-q4-internet-security-report.html.

"Taxonomy of DDoS Attacks". 2014. Viitattu 20. huhtikuuta 2015. http://www.riorey.com/s/RioRey_Taxonomy_DDoS_Attacks_26_2014-1.pdf.

Yatagai, Takeshi, Takamasa Isohara ja Iwao Sasase. 2007. "Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior". Teoksessa *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on,* 232–235. IEEE, elokuu. ISBN: 978-1-4244-1189-4. doi:10.1109/PACRIM.2007.4313218.

Zargar, S.T., J. Joshi ja D. Tipper. 2013. "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks". *Communications Surveys & Tutorials, IEEE* 15 (4): 2046–2069. doi:10.1109/SURV.2013.031413.00127.