Sikkerhetsovervåking av nettverk med Network Flight Recorder.

Deteksjon av den trojanske hesten Netbus.

Hovedoppgave
ved
sivilingeniørutdanning i
informasjons- og kommunikasjonsteknologi

av
Per Kristian Johnsen

Grimstad, juni 1999

Forord

I denne rapporten presenteres resultater fra hovedoppgaven utført vårsemesteret 1999 av Per Kristian Johnsen som er student ved sivilingeniørstudiet i informasjons- og kommunikasjonsteknologi ved Høgskolen i Agder.

Stadig flere bedrifter og organisasjoner er i dag avhengig av datasystemer som et hjelpemiddel i den daglige arbeidsprosessen. Etter hvert som disse systemene har fått en viktigere rolle, samtidig som datasystemene i større grad er blitt en del av Internett, øker viktigheten av å kunne beskytte seg mot en økende trussel; En trussel for at inntrengere skal bryte inn i datasystemer og utøve uønskede aktiviteter.

Denne hovedoppgaven tar utgangspunkt i Network Flight Recorder, et amerikansk datasystem for nettverksanalyse og overvåking. Oppgaven er gitt av firmaet System Sikkerhet AS, et rådgivende ingeniørfirma innen IT-sikkerhet som holder til i Arendal.

Jeg vil takke System sikkerhet AS for oppgaven, faglig veileder Frank Stien (v/ System Sikkerhet AS) for god veiledning og oppfølging, samt for mange gode ideer og råd underveis. Jeg ønsker også å takke ansvarlig veileder Vladimir Oleshchuk ved Høgskolen i Agder for gode råd, samt Nils Ulltveit Moe ved Høgskolen i Agder for god hjelp ved konfigurering av operativsystemet OpenBSD.

Grimstad, 28. Mai 1999

Per Kristian Johnsen

Sammendrag

Stadig flere bedrifter og organisasjoner er i dag avhengige av datasystemer. Sett i sammenheng med at disse systemene i større grad er blitt en del av Internett, har det blitt stadig viktigere å kunne beskytte seg mot innbrudd i disse datasystemene. Som en konsekvens av dette, har det blitt utviklet spesielle systemer for nettverksovervåking og analyse. I denne oppgaven har det blitt sett på muligheten for å benytte et slikt system ved navn Network Flight Recorder til å detektere en såkalt trojansk hest ved navnet Netbus. Dette er et sikkerhets-brytende programme som er mye utbredt i dag. Network Flight Recorder benytter separate kodeenheter kalt filtre for å overvåke ulike typer av nettverkstrafikk. I den sammenheng har det i denne oppgaven blitt utviklet 5 slike filtre for å gjøre dette systemet i stand til å oppdage det omtalte programmet Netbus. Sentrale hendelser i denne utviklingsprosessen har vært omfattende eksperimentering på et dedikert testnettverk. Hensikten med dette var å finne mønstre i datatrafikken som kan brukes for å påvise Netbus. Denne prosessen lyktes, og de 5 resulterende filtre fungerte tilfredsstillende.

Innholdsfortegnelse

1 Innledning

I siste semester ved sivilingeniørstudiet ved Høgskolen i Agder skal studentene gjennomføre en avsluttende hovedoppgave. Oppgaven gis av høgskolen eller næringslivet, og skal gjennomføres selvstendig av studenten i dette semesteret. Oppgavens omfang er på 10 vekttall, tilsvarende normert studiemengde for et semester.

1.1 Beskrivelse av oppgaven

Oppgaven går ut på å skrive filtre til nettverksanalyse- og overvåkningsverktøyet Network Flight Recorder (NFR) for å gjøre dette systemet i stand til å detektere den mye utbredte trojanske hesten[2,3,4] Netbus. En trojansk hest er et program som tilsynelatende gjør noe nyttig eller interessant, men som også utfører en uventet, ofte sikkerhets-brytende funksjon. Trojanske hester beskrives mer utfyllende i kap. 5. NFR er et system som benytter separate kodemoduler kalt filtre for å kunne kjenne igjen ulike typer av nettverkstrafikk. I de tilfeller der de medfølgende filtre ikke dekker det analyse- eller overvåkingsmessige behovet man måtte ha, har man muligheten til å lage egne filtre vha. et internt skript-språk i NFR kalt N-Code.

Oppgaven er utført ved Høgskolen i Agder. Ekstern veileder for oppgaven er Frank Stien ved System Sikkerhet AS. Ansvarlig veileder er Vladimir Oleshchuk ved Høgskolen i Agder.

1.2 Motiv

Som et av landets ledende rådgivende firmaer innen IT-sikkerhet har System Sikkerhet AS pekt ut sikkerhetsovervåkning av nettverk som et av sine nye satsningsområder. De har i den forbindelse blitt eneforhandler og installerer av det anerkjente verktøyet Network Flight Recorder i Norge, og ønsker å styrke egen kompetanse på utvikling av filtre som et ledd i å kunne tilby mer skreddersydde løsninger til sine kunder.

Kandidaten fikk i utgangspunktet relativt frie tøyler når det gjaldt valg av angrep å lage filter for. At kandidaten valgte å lage filter for en trojanske hest skyldes flere faktorer:

- Trojanske hester har fått stor utbredelse de to siste årene.
- Det finnes ved oppgavens starttidspunkt få filtre til NFR for deteksjon av trojanske hester.
- Mange av de eldre angrepene har mistet noe aktualitet pga. sikkerhetsoppgraderinger og fixer fra de ulike berørte leverandører.
- Tredjeparts leverandører har laget gratis filtre for de fleste eldre, mer velkjente angrepstypene.
- Trojanske hester kan kompromittere en infisert PC eller datasystem alvorlig, og utgjør derfor en stor sikkerhetsrisiko.
- De nyeste og mest utbredte trojanske hestene retter seg i hovedsak mot Microsoft plattformene, som størsteparten av verdens maskinpark benytter.

Når beslutningen om å detektere trojanske hester var tatt, falt valget på Netbus fordi dette er en av de mest funksjonsrike, utbredte og fryktede trojanske hester som eksisterer for Microsoft plattformene.

Virusskannere kan oppdage og fjerne mange av de meste utbredte trojanske hestene, inkludert Netbus. Derfor kan man stille seg spørsmålet hvorfor det er behov for å lage filtre for deteksjon av trojanske hester som dette. Det er flere grunner til det:

- I bedrifter osv. slurves det ofte med å holde virusdefinisjons-filen(e) oppdatert.
- Virussjekking kan kobles ut av ansatte, f.eks. for å øke ytelsen på en PC.
- NFR kan logge IP adressen til inntrengere, virussjekking bare fjerner den trojanske hesten.
- Systemansvarlige får bedre oversikt over maskiner som er infisert, og kan deretter vurdere skadeomfanget.

1.3 Avgrensning av oppgaven

Oppgavens avgrensninger ligger i at det kun lages filter for deteksjon av den trojanske hesten Netbus. Å lage filtre som oppdager alle trojanske hester er både praktisk og teoretisk umulig.

1.4 Mål

Det overordnede målet for oppgaven har vært å lage filtre for deteksjon av alle kjente versjoner av Netbus med minst mulig grad av feilprosent. Siden kandidaten fra starten av fikk relativt frie tøyler mht. valg av angrepstype, har det imidlertid vært flere delmål for å kunne nå det endelige målet.

Nødvendige delmål for oppgaven har vært:

- Få en god oversikt over de ulike typer av angrep, for å kunne definere oppgaven.
- Få en god oversikt over NFR, og kunne bruke dette systemet i praksis.
- Lære skript-språket N-Code.

1.5 Rapportens struktur

Rapporten begynner med en innledning med oppgavedefinisjon, motiver, avgrensninger og mål for oppgaven. Deretter følger en metodedel, der det beskrives hvordan kandidaten har jobbet for å løse oppgaven. Videre følger en generell innføring til de aktuelle teknologier/emner som ligger til grunn for å kunne gjennomføre og forstå oppgaven.
Disse teknologiene/emnene er:

- Intrusion Detection Systems (IDS).
- Network Flight Recorder (NFR).
- Trojanske hester.

Videre følger en mer spesifikk beskrivelse av de aktuelle emnene innenfor hver teknologi som angår hovedoppgaven spesielt:

- Netbus.

Deretter følger en detaljert beskrivelse av arbeidet for å detektere Netbus, samt beskrivelse av filtrene som ble laget.

Oppgaven avsluttes og oppsummeres med en diskusjon og konklusjon, samt helt til slutt et stikkordregister og litteraturreferanser med oversikt over litteratur som er brukt i gjennomføring av oppgaven, og som er referert til gjennom hele rapporten.

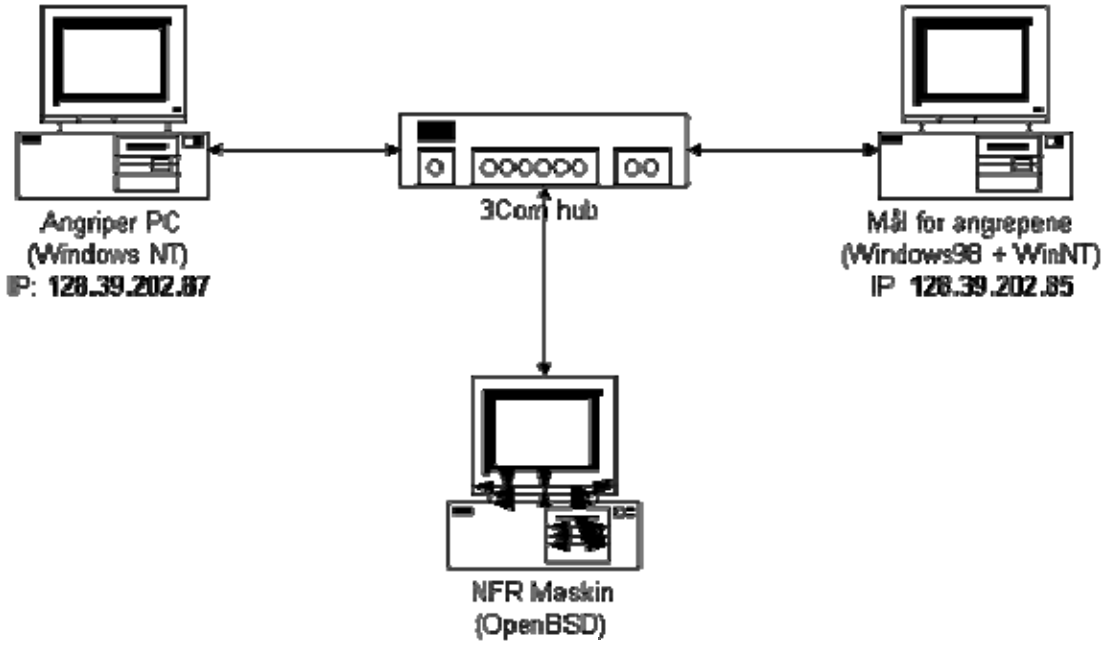
2 Metode

2.1.1 Teoretisk fundament

Fra semesterets start i januar og fremover de neste to månedene ble det foretatt en litteraturstudie for å tilegne seg kunnskap om de ulike angrepene og danne seg en oversikt over disse. Den type informasjon som var nødvendig for å få til dette finnes praktisk talt ikke utgitt i bokform. Nødvendig informasjon er derfor i sin helhet hentet fra utallige kilder på Internett. Mangelen på en samlet, strukturert oversikt over typer av angrep og deres virkemåte, samt det faktum at mye av informasjonen kommer fra undergrunnskilder, gjorde litteraturstudiet til en tidkrevende prosess.

2.1.2 Praktiske aspekter

Siden arbeidet med å få den teoretiske oversikten på mange måter var en modningsprosess, ble det parallelt med dette utarbeidet en plan for den praktiske siden av oppgaven. Det ble her besluttet å sette opp et isolert 10 Mbit nettverk for eksperimentering og uttesting av filtrene. Dette nettverket bestod av tre PC'er: En angriper PC, en mål-maskin for angrepene og en NFR-maskin, hvis hensikt var å oppdage angrepene. Disse tre maskinene ble koblet sammen vha. en 3Com hub og nettverkskabel som vist i figuren nedenfor.



Figur 1. Skisse over testnettverket.

Som alternativ til å lage et eget nettverk for å eksperimentere på, ble det vurdert å koble NFR-maskinen til høgskolens nettverk. Det var imidlertid flere grunner til at dette forslaget ble forkastet på et tidlig tidspunkt:

- NFR overvåker trafikken på et gitt nettverk, og kan lagre kopier av den informasjon som går der. Å gi studenter tilgang til slik informasjon er i utgangspunktet uønsket fra skolens side. Det ville derfor blitt en tidkrevende, kanskje umulig, prosess å få tillatelse til å sette opp NFR-maskinen på dette nettverket.
- For å få best mulig arbeidsforhold under utviklingen og uttestingen av filtrene kreves et nettverk med kontrollerte arbeidsbetingelser. Dette tilbyr et separat nettverk dedikert for oppgaven bedre enn høgskolens nettverk.

Etter noe problemer med konfigurering av NFR-maskinen, fungerte testnettverket tilfredsstillende, og en periode med utprøving og praktisk bruk av NFR fulgte parallelt med de videre litteraturstudier. Netbus ble så installert på de respektive mål- og angrepsmaskiner, og arbeidet med å finne en egnet metode for deteksjon av denne trojanske hesten startet.

2.1.2.1 Arbeidsmetode for deteksjon av Netbus

Siden trojanske hester er separate programmer, vokste det frem en teori om at enkelte sekvenser blant nyttedata (payload) i datapakkene som utveksles over nettverket kan benyttes som signatur for å unikt identifisere Netbus. Dette førte til at det ble installert en pakkesniffer på angrepsmaskinen. Til dette formål ble gratisprogrammet Hoppa Analyzer[7] benyttet. Pakkesnifferen ble deretter brukt til å logge trafikken som utveksles mellom Netbus klienten og serveren. Deretter starter en prosess der disse loggene finkjemmes for mulige signaturer eller mønstre som kan benyttes for å unikt identifisere Netbus. Etter at eventuelle signaturer er identifisert, skrives filtre for deteksjon av de gitte signaturer. Filtrene evalueres så fortløpende på testnettverket. Se kap. 6 og 7 for en mer detaljert beskrivelse av Netbus signaturer og filtre.

3 Intrusion Detection Systems (IDS)

3.1 Innledning

Det siste tiåret har det vært en eksplosiv utvikling når det gjelder utbredelsen av Internett og internettjenester. Internett har gitt oss muligheter for innhenting og utveksling av informasjon på måter som for noen få år tilbake var umulig. Utbredelsen av WWW (*World Wide Web*), og andre internettjenester som elektronisk post og filoverføring har gjort sitt inntog stort sett overalt: i utdanningsinstitusjoner, næringsliv, statlige institusjoner, og private husstander. Datamaskiner, ofte koblet sammen i interne nettverk, er av avgjørende betydning for mange bedrifter, institusjoner og organisasjoner. Med denne utbredelsen av Internett, datamaskiner og nettverk følger det også med en risiko: En risiko for at noen foretar et innbrudd i datasystemet. Med innbrudd menes i denne sammenheng at noen, ofte betegnet en ”hacker” eller ”cracker”, forsøker å bryte inn i eller misbruke datasystemet. Ordet ”misbruke” er her brukt i en vid betydning, og omfatter alt fra alvorlige hendelser som å stjele konfidensiell informasjon til de mindre alvorlige, som for eksempel misbruk av e-post tjenester for utsendelsen av spam-post. Som en følge av denne trusselen om innbrudd og angrep mot datasystemer, har det de senere årene blitt utviklet en rekke verktøy eller systemer for å oppdage slike innbrudd. Disse systemene kalles med en samlebetegnelse for Intrusion Detection Systems (IDS). Betegnelsen ”Intrusion Detection System” kan man igjen dele inn i ulike kategorier.

3.2 Kategorier av Intrusion Detection Systemer

Network Intrusion Detection Systems (NIDS) overvåker datapakker som går gjennom nettverket og forsøker å oppdage om en hacker el. cracker forsøker å bryte inn i et system eller lamme hele eller deler av dette. (Såkalt Denial of Service angrep). Et NIDS kjøres på en uavhengig maskin som passivt overvåker all trafikk på nettverket.

Host-based Intrusion Detection er systemer som kjører på hver enkelte maskin, og som benytter seg av informasjon fra operativsystemet for å overvåke alle operasjoner/hendelser som foregår på maskinen. Disse hendelser sammenliknes så med en predefinert sikkerhetsprofil for å oppdage innbrudd.

System Integrity Verifiers (SIV) overvåker system filer for å oppdage når en inntrenger endrer dem, f.eks. for å etterlate seg en bakdør. Et SIV system kan godt overvåke andre komponenter i tillegg for å oppdage velkjente innbruddssignaturer.

Log File Monitors (LFM) overvåker log filer generert av nettverkstjenester. På en liknende måte som ved NIDS, kikker disse systemene etter mønstre i log filer som kan avsløre en angripende inntrenger. Et typisk eksempel vil være et program som kikker i Web server log filer etter inntrengere som forsøker velkjente sikkerhetshull.

3.3 Hvordan kommer inntrengere seg inn i systemer?

Det er primært 3 måter en inntrenger kan komme inn i et datasystem:

Fysisk innbrudd: Hvis en inntrenger har fysisk tilgang til en PC vil muligheten til å bryte inn være tilstede. Teknikkene varierer fra å benytte metoder som f.eks. bootdisk til å fysisk plukke ut harddisken og lese/skrive til denne på en annen maskin.

System innbrudd: Denne typen av innbrudd forutsetter at inntrengeren allerede har en bruker konto på systemet. Hvis systemet ikke har de siste sikkerhetsoppgraderinger, er det en god sjanse for at inntrengeren kan tilegne seg høyere privilegier ved å benytte kjente exploits (f.eks. GetAdmin for NT).

Fjern innbrudd: Denne typen angrep omfatter forsøk på å få fjerntilgang til systemer gjennom nettverket. Inntrengere starter uten noen spesielle privilegier på systemet som angripes. Denne typen innbrudd vanskeliggjøres betraktelig hvis det eksisterer en god brannmur mellom inntrenger og offer.

3.4 Hvem utgjør trusselen?

Det ble innledningsvis slått fast at en inntrenger ofte betegnes som hacker eller cracker. En hacker er en fellesbetegnelse på en som liker å komme inn i ting. Man kan så skille mellom den vennligsinnede hackeren, som er en person som liker å komme inn i sin egen datamaskin og forstå hvordan den fungerer, og den ondsinnede hackeren som liker å komme inn i andres systemer. I denne rapporten vil betegnelsen inntrenger bli brukt for å beskrive alle som

uautorisert forsøker å bryte inn i et datasystem. Inntrengere kan igjen klassifiseres i to kategorier:

Utenforstående er inntrengere som angriper fra utsiden av nettverket. De vil, i de tilfeller det finnes, forsøke å gå rundt en brannmur for å angripe maskiner tilknyttet det interne nettverket. Utenforstående inntrengere kan angripe fra Internett, via oppringte linjer, fra partner-nettverk som er tilkoblet internnettverket eller ved fysisk innbrudd i bygningen. Av disse er det angrep via Internett som utgjør hovedtyngden og dermed representerer den største trusselen.

Innsidere er inntrengere som lovlig benytter det interne nettverket. Dette innbefatter personer som misbruker privilegier eller som utgir seg for å være personer med høyere privilegier enn dem selv. Enkelte anslag mener at så mange som 80% av alle sikkerhetsbrudd utføres av personer på innsiden av et nettverk.

I tillegg har man flere ulike motivasjoner for å bryte seg inn i et nettverk: De som bryter seg inn fordi de kan og ser på det som en utfordring. Denne typen er vanligvis relativt harmløse, men selvfølgelig allikevel uønsket. Vandaler bryter seg inn i systemer med den hensikt å forårsake skade og ødeleggelser på systemet. En tredje type er de som bryter seg inn med profitt for øyet. Hensikten her kan være å selge informasjon til andre eller bedrive utpressing.

Intrusion detection handler altså om å oppdage angrep og innbruddsforsøk, helst mens de skjer. I forbindelse med NIDS og LFM, er teorien bak dette at ethvert angrep har sin egen, unike ”signatur”, eller kjennetegn, som kan brukes for å kjenne igjen akkurat dette angrepet. Det eksisterer en mengde angrep som retter seg mot enkeltmaskiner, hele nettverk eller begge deler. Man har derfor funnet det hensiktsmessig å klassifisere dem.

3.5 Typer av angrep

Variasjoner i virkemåte og hensikt har ført til at man kan klassifisere angrepene i to typer[1]; exploits og Denial of Service (DoS) angrep. Forut for et angrep foretar en potensiell inntrenger ofte en kartlegging av systemet angrepet skal rettes mot. Et angrep består altså ofte av 2 faser: Kartlegging etterfulgt av selve angrepet. Kartlegging karakteriseres ikke som angrep, men siden denne fasen ofte spiller en viktig rolle forut for angrepet, omtales den her.

3.5.1 Kartlegging

Som navnet antyder går denne fasen ut på å finne ut mest mulig om et gitt system eller nettverk. Dette innebærer som regel at det foretas en skanning etter ulike typer av respons eller tjenester. Den enkleste formen for slik skanning, er å utføre kommandoen ping mot et område av IP adresser for å finne ut hvilke PC'er som er tilgjengelige/slått på. Andre typer av kartlegging er TCP eller UDP skanning. Her skanner man etter åpne (lyttende) TCP eller UDP porter for å finne tjenester som kan misbrukes. Andre former for kartlegging er å forsøke å finne ut hvilket operativsystem som benyttes av et potensielt angrepsmål. Dette gjøres ved å sende ulovlige eller rare ICMP eller TCP pakker, og ut fra responsen tyde hva slags OS som benyttes. En annen mye utbredt kartleggingsmetode er brukerkonto-skanning. Her forsøker man eksempelvis å logge på en PC ved å benytte kontoer som installeres "by default", brukernavn uten passord, samme passord som brukernavn osv. Siden en vell utført kartleggingsfase kan avsløre mye nyttig informasjon om et system, benyttes det ofte i et tidlig stadium av et større og mer alvorlig angrep.

3.5.2 Exploits

Dette er en type angrep som utnytter skjulte funksjoner eller feil i applikasjoner for å få tilgang til systemer. Noen vanlige typer av exploits er;

- CGI skript angrep
- Web server angrep
- Web browser angrep
- SMTP (Sendmail) angrep
- IMAP angrep
- IP spoofing
- Buffer overflow
- DNS angrep

3.5.3 Denial of Service (DoS) angrep

Denial of Service er en type angrep der en inntrenger/angriper forsøker å hindre at man får tilgang til tjenester, PC'er eller hele systemer. Derav navnet Denial of Service. Dette gjøres gjerne ved at man forsøker å kræsje PC'er, eller på andre måter gjøre tjenester utilgjengelige for brukerne. Andre måter å foreta DoS på kan være å overbelaste nettverket eller PC'ens prosessor, eller å fylle opp harddisken. Hensikten med et DoS angrep er som regel ikke å få tilgang til informasjon, men kun av destruktiv art. De aller fleste DoS angrep utnytter feil, mangler eller svakheter i ulike protokoll-implementasjoner for å oppnå sitt mål.
Velkjente DoS angrep lyder navn som:

- Ping of Death/Ssping
- SYN Flooding
- Land/Latierra
- WinNuke
- Smurf
- TearDrop/NewTear
- Bonk/Boink

3.6 Hvordan oppdages innbrudd?

Det er i hovedsak 2 teknikker som benyttes for å detektere innbrudd[1];

Signatur gjenkjenning er den mest vanlige metoden. Her sammenliknes inn- og utgående trafikk med velkjente innbruddssignaturer. F.eks. vil et stort antall feilende TCP forbindelser til en rekke porter indikere at noen foretar en TCP-port skanning.

Anormal deteksjon benytter statistisk analyse for å finne endringer fra "normal" oppførsel. Denne metoden er mindre kraftig enn signatur gjenkjenning, men har den fordelen at den kan oppdage angrep som det ikke eksisterer signaturer for.

3.7 Network Flight Recorder og Intrusion Detection

Som verktøy for å oppdage angreps- og innbruddsforsøk hører NFR inn under kategorien Network Intrusion Detection System (NIDS), og benyttes derfor hovedsakelig for å detektere uønskede aktiviteter internt på nettverk, samt for deteksjon av fjern innbrudd som omtalt i avsnitt 3.3. NFR brukes vanligvis for å overvåke all trafikk på nettverket, og kjøres derfor på en uavhengig maskin. Denne maskinen kobles vanligvis til nettsegmentet rett før eller etter en brannmur, alt ettersom hva man ønsker å overvåke el. analysere. Ønsker man å overvåke all trafikk som rettes mot det interne nettverket, plasseres NFR på utsiden av brannmuren. Ønsker man derimot å overvåke/analysere trafikken som faktisk slipper gjennom brannmuren, og således spres på det interne nettverket, plasseres NFR på innsiden. På denne måten kan NFR faktisk benyttes til å verifisere konfigurasjonen av brannmurer.

4 Network Flight Recorder

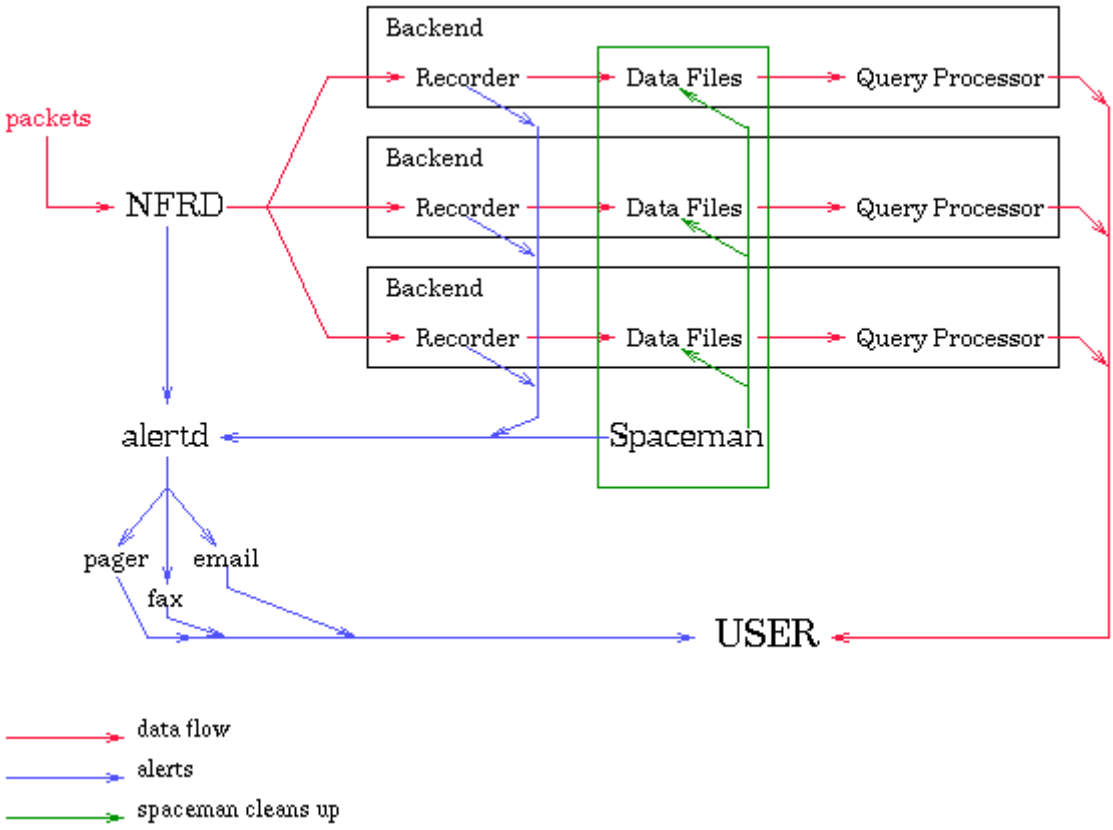
4.1 Innledning

Network Flight Recorder er et fleksibelt verktøy for overvåkning av nettverk og analyse av nettverkstrafikk. Det er utviklet av amerikanske Network Flight Recorder Inc, og ble lansert i desember'97. Hele konseptet med NFR baserer seg på at et NFR system overvåker det ønskede nettverket og passivt samler inn ønsket informasjon fra datapakkene som beveger seg gjennom nettverket. NFR kan logge trafikk, analysere den i sanntid og utløse alarmer når uventede eller uønskede hendelser inntreffer. Hva slags type trafikk man skal overvåke, data som skal lagres og kriterier for å utløse alarmer kan fritt spesifiseres av bruker. NFR leveres med en rekke generelle konfigurasjoner som overvåker og samler inn informasjon om noen av de mest vanlige trafikktyper og mønstre. Ønsker man å overvåke og logge aktiviteter som det ikke følger med konfigurasjoner for, kan bruker selv lage filtre, dvs. egne moduler for trafikkovervåkning vha. N-Code, NFRs innebygde skript-språk. Siden NFR er et generelt verktøy for nettverksanalyse og overvåking er bruksområdene mange. Følgende liste angir bare noen av NFR mange bruksområder.

- Overvåke nettverkstrafikk.
- Dokumentere nettverksendringer.
- Spare på historiske statistikker ang. nettverkets vekst.
- Generere detaljerte oversikter over hvordan nettverkstjenester blir brukt, og av hvem.
- Kikke etter mønstre for misbruk av nettverksressurser og identifisere synderen i sanntid.
- Sette opp innbruddsalarmer som varsler om sikkerhetsbrudd eller uventede forandringer i nettverket.
- Logge og overvåke aktiviteter på nettverket.
- Filtrere e-post for sensitive ord or utrykk.

4.2 NFRs interne arkitektur

NFRs arkitektur er implementert som et sett av separate moduler, som alle er knyttet til en spesiell aktivitet. Det er 4 hovedmoduler[5, 8]: Data samles inn av en eller flere **pakke-samlere**. De overføres så til en **decision engine** som dekoder, filtrerer og ordner de innsamlede pakkene i riktig rekkefølge. Data sendes så videre til "**backends"** for eventuell lagring. **Grensesnittet for spørring/søk** (Query interface) holdes separat fra den innkommende datastrømmen for å minimere de ytelsesmessige påvirkninger av en spørring mens systemet samler inn data. Følgende figur[8] illustrerer arkitekturen til NFR.



Figur 2. Oversikt over arkitekturen til Network Flight Recorder.

4.2.1 Lesing av pakker/pakke samlere (Packet sniffer)

NFR begynner sin prosess når den mottar pakker fra nettverket. Disse pakkene blir lest, og deretter sendt videre til ”decision engine” (også referert til som **nfrd**) for filtrering og gjensammensetning. Hvis informasjonen på bakgrunn av filtreringslogikken blir vurdert som relevant, lagres den av ”backends” for spørringer og videre statistisk behandling.

NFRs pakke-samlere ble opprinnelig basert på libpcap[6] pakke samler interface. Libpcap tilbyr en generell pakke samler funksjonalitet liggende over en rekke operativsystem-spesifikke nettverk samler interface.

4.2.2 Decision engine (nfrd)

Modulen som kalles ”decision engine” utfører flere funksjoner, slik som sammensetning av TCP strømmer. Dens hovedoppgave er imidlertid å eksekvere N-Code, som er NFR interne skript-språk. N-Code utfører ulike funksjoner som trigges/aktiviseres av de innkommende data.

”Decision engine” dekoder den innsamlede informasjonen og sørger for å sette sammen de innsamlede pakkene i riktig rekkefølge. Pakkene blir så utsatt for ulike filtre, som filtrerer den innsamlede informasjonen og sender utvalgte deler til ”backends” for statistisk innsamling eller lagring i såkalte ”recorders”. Filtre kan aktiveres eller deaktiveres etter eget ønske for å kun samle inn de data man trenger. Ved å bruke N-Code kan man skrive egne filtre for mønstergjenkjenning og filtrering av data. Filtre leses inn i decision engine, kompileres og lagres som byte-kode instruksjoner for rask eksekvering. Filterspråket N-Code knytter et filter til en hendelse eller til mottakelsen av en pakke. Pakken forkastes/vrakes etter at den har blitt utsatt for filterne. Gjennom N-Code kan man få tilgang til felter inne i pakkene, vanligvis for å lagre informasjon fra disse felter. Det finnes to hoved-primitiver for å eksportere data ut av et filter; ***alert*** og ***record***. Alert sender en melding til alarm-håndteringssystemet. Record sender en konstruert datastruktur til en ”backend recorder” for videre prosessering.

4.2.3 Backends

Backends kontrollerer hvilke typer data som skal lagres og hvordan dette skal gjøres.
For å gjøre dette, består hver backend egentlig av 3 forskjellige komponenter; Et filter, konfigurasjonsfiler og en recorder. Filteret er ”kjernen” i en backend, og avgjør hva slags trafikk/data som skal trigge NFR til å begynne å samle data.
Avhengig av evalueringsresultater av data fra decision engine sendes data til de ulike backends. Vha. recordere lagrer backends data i filer, og tillater spørringer mot de lagrede data. Det eksisterer foreløpig to typer av recordere i NFR; **histogram** og **list**. Histogram vedlikeholder en kolonnetabell over data, og enten summerer verdifelt eller øker dem. List recorder lagrer en kronologisk rekkefølge over data. Siden list recorder ikke slår sammen informasjon slik som histogram gjør, anses den for å være mindre plass effektiv.

4.2.4 Java-basert grafisk brukergrensesnitt (GUI)

Network Flight Recorder har et java-basert grafisk brukergrensesnitt som man vanligvis aksesserer fra en annen PC via en browser. Ved å bruke dette grafiske brukergrensesnittet kan man forespørre/søke i data som har blitt lagret på disk i ”recorders” av de ulike backends. Gjennom det grafiske brukergrensesnittet kan man søke i og sortere data etter spesifikke kriterier. Resultatet av forespørselen kan presenteres i form av tekst, grafer, kakediagram eller spredningstabeller. Brukergrensesnittet for spørringer er en separat funksjon for å ikke interferere med lagringen av data.

Gjennom det grafiske brukergrensesnittet har man full mulighet til å konfigurere hvilke typer aktivitet som skal få NFR til å utløse en handling, og hvilke kanaler disse handlinger skal sendes gjennom. Handlingene som iverksettes kan i alvorlighetsgrad spenne fra å kun komme med en liten informasjonsanmerkning i en log-fil til å sende ut alvorlige advarsler eller feilmeldinger over dedikerte kanaler.

4.2.5 Andre funksjoner

I tillegg til de 4 hovedmodulene, består NFRs arkitektur av to andre moduler/funksjoner: **Alert manager** og **Space manager**.

4.2.5.1 Alert manager (alertd)

Basert på avgjørelser fra filtreringen kan NFR sende en alarm. Dette er en melding som sendes til brukeren av NFR (typisk systemansvarlig etc.) når usedvanlige eller nevneverdige hendelser inntreffer. Alarmen utløses fra den delen av systemet som oppdager tilfellet, vanligvis et filter eksekvert i decision engine. Alarmen sendes så til Alert manager (også betegnet som **alertd**). Alert manager ordner rekkefølge og prioritet på alarm-data som mottas fra decision engine, og sender alarmen ut til bruker gjennom en passende kanal. Alert manager kan konfigureres til å sende alarmer gjennom flere ulike kanaler, slik som fax, personsøker, mail osv.

4.2.5.2 Space Manager (spaceman)

Etter at nettverkstrafikken har blitt analysert, lagres utvalgte deler av den. Backends lagrer data på harddisker. Diskbruken overvåkes og kontrolleres av NFRs eget system for håndtering av diskplass kalt **spaceman**. Etterhvert som backends lagrer data, kan spaceman arkivere eller slette deler av dataene etter gitte kriterier, som f.eks. alder eller størrelse. Spaceman arbeider i bakgrunnen og sørger for at de store mengdene med data som samles opp blir håndtert på en riktig måte.

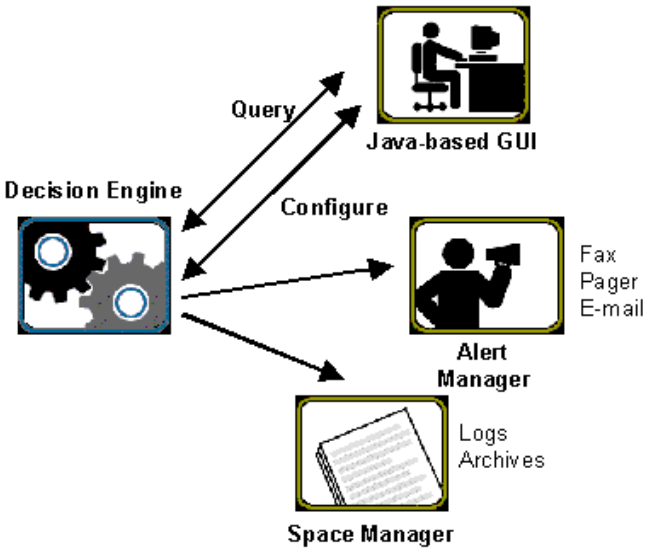


Fig.3. Modulene i NFR og sammenhengen mellom dem.

5 Trojanske hester

5.1 Innledning

Trojanske hester betegner en gruppe applikasjoner som benyttes for å kompromittere enkeltmaskiner og systemer. En trojansk hest kan defineres som et ”ondartet, sikkerhets-brytende program som er forkledd som noe annet, slikt som f.eks. en skjermsparer, et spill osv”[2]. Det finnes imidlertid mange definisjoner av trojanske hester.

Andre definerer en trojansk hest som ”et uautorisert program pakket inn i et legitimt program. Dette uautoriserte programmet utfører funksjoner ukjent for, og som oftest uønsket av, brukeren”[3].

RFC 1244, the Site Security Handbook[4], definerer en trojansk hest på følgende måte:
”En trojansk hest kan være et program som gjør noe nyttig, eller rett og slett noe interessant. Det gjør imidlertid alltid også noe uventet, som f.eks. å stjele passord eller kopiere filer uten din viten.”

Ethvert program som ser ut til å utføre en ønskelig og ofte nødvendig funksjon, men som pga. at det inneholder uautorisert kode i tillegg utfører funksjoner ukjent for brukeren, kan altså høre inn under definisjonen trojansk hest.

Trojanske hester forveksles ofte med virus. Et virus defineres av Norman ved følgende kriterier[9]:

- Evnen til å kopiere seg selv inn i andre filer.
- Virus trenger en vert for å spre seg.
- En hendelse som ikke er tilsiktet fra brukers side må inntreffe.

Trojanske hester skiller seg fra virus ved at de ikke har evnen til å kopiere seg selv inn i andre filer, og således spre seg på denne måten. En trojansk hest kan i og for seg inneholde et virus, men ifølge definisjonen er det ikke virus.

5.2 Hva gjør trojanske hester?

En trojansk hest kan i utgangspunktet være skapt for å utføre hva som helst, men sett i sammenheng med nettverks- og Internettsikkerhet, vil en trojansk hest vanligvis gjøre en av to ting:

- Utføre funksjoner som enten avslører vital og privilegert informasjon om et system eller kompromitterer systemet. Under denne kategorien har man sett eksempler på trojaner som avslører brukernavn og passord for en tredjepart ved å sende dette til vedkommende pr. mail, ICQ osv. Denne typen trojaner kan sies å ha en penetrerende hensikt.

- Gjemme noen funksjoner som enten avslører vital og privilegert informasjon om et system eller kompromitterer systemet. Denne type trojan kan sies å ha en innsamlende funksjon, og eksempel her kan være en trojan som i det skjulte logger alt som blir skrevet på tastaturet.

Noen trojanske hester gjør begge delene. I tillegg finnes det en tredje type trojansk hest som kun har til hensikt å forårsake skade. Eksempel på slik skade kan være å kryptere eller formatere harddisken. Trojanske hester er ikke noe nytt fenomen, men fra å tidligere være relativt ”uskyldige” programmer hvis hensikt var å skape fysisk trøbbel på en maskin, ble det på mange måter satt en ny standard for denne typen applikasjon da en hacker-gruppe ved navnet Cult of the Dead Cow (CDC) 3. august i fjor ga ut den nå berømte og vidt utbredte trojanske hesten Back Orifice (BO), som rammer operativsystemene Windows 95/98. Dette er en trojansk hest som arbeider etter klient/jener prinsippet, der en Back Orifice server installeres på en mål maskin. Enhver person med Back Orifice klienten installert har nå i praksis full tilgang til den infiserte maskinen, og kan utføre en rekke funksjoner. Trojansk hester omtales derfor av og til som bakdører. Pga. dens rikholdige funksjonsgalleri, tilfredsstiller BO alle de tre beskrivelsene ovenfor. I kjølevannet av Back Orifice har det dukket opp en rekke kloner og varianter av denne berømte trojanske hesten. Den mest kjente av den heter Netbus, som i tillegg til å ha kraftig forbedret funksjonalitet også fungerer på Windows NT operativsystemet. Vi har altså konstatert at trojanske hester kan kategoriseres i 2 grupper; De som installeres for å utføre enkelthendelse(r) lokalt på en maskin, slik som f.eks. sletting av filer, kryptering av harddisk osv., og den mer sofistikerte typen trojan som installeres for å tjene som en bakdør til maskiner. Det er denne siste typen som med Back

Orifice og Netbus i spissen har fått stor utbredelse de siste årene, og som utgjør en stor sikkerhetsrisiko.

5.3 Hvorfor er trojanske hester så farlige?

Man kan dele datasikkerhet i 3 ulike deler som alle må være tilstede for at datasikkerheten skal være ivaretatt[10]. Disse er integritet, konfidensialitet og tilgjengelighet.

Integriteten er kompromittert dersom noen uautoriserte klarer å endre data. Man kan da ikke lenger stole på riktigheten av informasjonen.
Konfidensialiteten er kompromittert dersom noen uautoriserte klarer å få tilgang til en PC. Personen kan da få tilgang til informasjon som ikke var ment tilgjengelig for denne. Informasjonen kan til og med bli distribuert.
Tilgjengeligheten er kompromittert dersom noen på ulike måter klarer å gjøre data/informasjon utilgjengelig for de personer den er ment for. Typen av angrep som går under betegnelsen Denial-of-Service har som hensikt å kompromittere tilgjengeligheten.

Trojanske hester utgjør en stor sikkerhetsrisiko fordi mange av dem har mulighet til å kompromittere alle de 3 faktorer nevnt ovenfor. Det er den forholdsvis nye gruppen av trojansk hester som fungere etter klient/tjener prinsippet og tilbyr fjernstyring av infiserte maskiner som typisk er i stand til dette.

5.4 Hvordan overføres/installeres trojanske hester?

Som det går frem av de mange definisjoner, er trojanske hester programmer som er innkapslet i andre, som regel legitime programmer. Trojaner installeres derfor ved at man har foretatt en bevisst eksekvering av en applikasjon. I utgangspunktet kan alle programmer være infisert av en trojansk hest, men typiske programmer som inneholder trojaner er shareware og freeware lastet ned fra Internett. Den kanskje aller mest vanlig spredningsmetoden for trojanske hester i dag er som vedlegg i e-post. Det finnes utallige eksempler på at de mange ”uskyldige” moroprogrammene som florerer i e-post inneholder trojaner. Eksempelvis kan nevnes at det lille spillet Whack-a-mole, som går ut på å slå muldvarper i hodet med en klubbe når de stikker opp fra bakken, finnes i varianter som inneholder versjon 1.6 eller 1.7 av Netbus serveren.

5.5 Den trojanske hesten Netbus

5.5.1 Innledning

Netbus er, som nevnt tidligere, siste skudd på stammen av kraftige, brukervennlig trojanske hester rettet mot Microsoft plattformen. Det er skrevet av svensken Carl-Fredrik Neikter, og ble først lansert i mars'98. Det baserer seg på TCP/IP protokollene. Netbus fungerer i likhet med Back Orifice etter klient/tjener-prinsippet, der en Netbus server installeres på en PC som skal fjernstyres. Enhver maskin med en Netbus klient installert vil nå være i stand til å fjernstyre server-maskinen via Internett eller det interne nettverket. Netbus fungerer både på Windows 95/98 og NT, og ble opprinnelig utviklet som et rent hackerverktøy, hvis eneste hensikt var å gi uautorisert tilgang til datamaskiner som hadde fått Netbus serveren installert. Selv om Netbus serveren fra og med versjon 2.0 kan installeres som en bevisst handling, vil den som oftest bli installert ved å gjemme seg inni andre programmer, og således bli installert uten mottagers viten. Er Netbus serveren først installert på en PC, har en angriper i praksis full tilgang til og kontroll over denne maskinen. Trojanske hester kan på denne måten sies å utnytte bakdører i systemer. Netbus har i de senere versjoner gjennomgått en kraftig forbedring mht. funksjonalitet, og man har i dag bl.a. følgende muligheter:

- Åpne/lukke cd-rom spilleren
- Vise valgfrie bilder
- Bytte om på musetastene
- Starte valgfrie programmer
- Kontrollere musens bevegelser
- Vise meldinger på skjermen
- Foreta shutdown på systemet, logge av bruker osv.
- Gå til valgfri URL med default nettleser.
- Lytte etter tastetrykk, og sende disse tilbake til inntrengeren.
- Dumpe skjermbildet og sende dette til inntrengeren.
- Upload og download av filer, samt endring/sletting av filer.
- Aktivisere mikrofonen og sende dette til inntrengeren for innspilling.
- Stjele bilder som web-kameraer fanger opp (i versjon 2.0)
- Koble ut enkelttaster eller hele tastaturet.
- Vise, fokusere og avslutte applikasjoner som kjører på systemet.

Med slik funksjonalitet er det enkelt å innse at Netbus og andre liknende trojaner utgjør en stor sikkerhetsrisiko for de maskiner som blir infisert.

Fra og med versjon 2.0 har Netbus blitt et kommersielt produkt, som av forfatteren blir markedsført som et fjernadministrasjons- og spionverktøy. Navnet har også blitt endret fra Netbus til Netbus Pro, angivelig i et forsøk på å bli kvitt hacker-assosiasjonene som blir forbundet med programmet. Derfor er det i versjon 2.0 da også bare mulig å foreta en synlig installasjon av serveren. Det finnes imidlertid hackede/endrede utgaver av versjon 2.0 Pro tilgjengelig på Internett som foretar en skjult installasjon av serveren, slik som tilfellet er med de tidligere utgaver. På tross av forfatterens nå tilsynelatende legitime hensikt med Netbus 2.0, er Netbus' posisjon som en av de mest utbredte og funksjonskraftige trojanske hester udiskutabel.

Netbus har siden starten kommet i flere versjoner, og har med den kommersielle versjon 2.0 Pro fått et meget innholdsrikt funksjonsgalleri. En kort oversikt over historikken til Netbus er som følger;

- Mars'98 Netbus 1.2 utgitt, svensk brukergrensesnitt.
- April'98 Netbus 1.5x utgitt, brukergrensesnitt oversatt til engelsk.
- 23.8.98 Netbus 1.60 utgitt, Netbus blir globalt kjent.
- 14.11.98 Netbus 1.70 utgitt, forbedret funksjonalitet.
- 19.2.99 Netbus 2.0 Pro utgitt, kommersiell, kraftig forbedret funksjonalitet.
- 5.4.99 Netbus 2.01 utgitt, enkelte tilleggsfunksjoner.

5.5.2 Brukergrensesnittet

Selve oppkoblingen mot en Netbus-infisert maskin er ytterst enkel: Man angir bare den infiserte maskinens IP-adresse og TCP portnummeret som Netbus serveren benytter i klient programmet. Eventuelt passord for Netbus serveren oppgis også her.

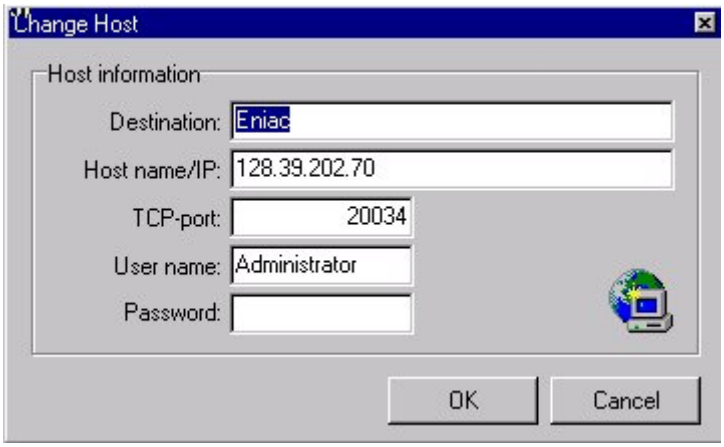


Fig. 4. Grensesnittet for å konfigurere en Netbus-oppkobling i versjon 2.0 Pro.

For å finne frem til infiserte maskiner skanner man IP adresser i ønskede områder for tilstedeværelsen av en Netbus server. Dette kan enten gjøres av Netbus klienten eller av egne programmer for å finne infiserte maskiner. I de tilfeller der Netbus serveren er passordbeskyttet, finnes det egne programmer utviklet for å knekke disse. Passord kan eksemplevis tenkes brukt i enkelte tilfeller der en bruker med hensikt har installert Netbus for å ha mulighet til å f.eks. fjernstyre en maskin på arbeidsplassen fra hjemmet, eller når man ønsker å hindre at andre skal få tilgang til en infisert maskin man selv har oppdaget (Netbus servere kan konfigureres fra klientsiden).

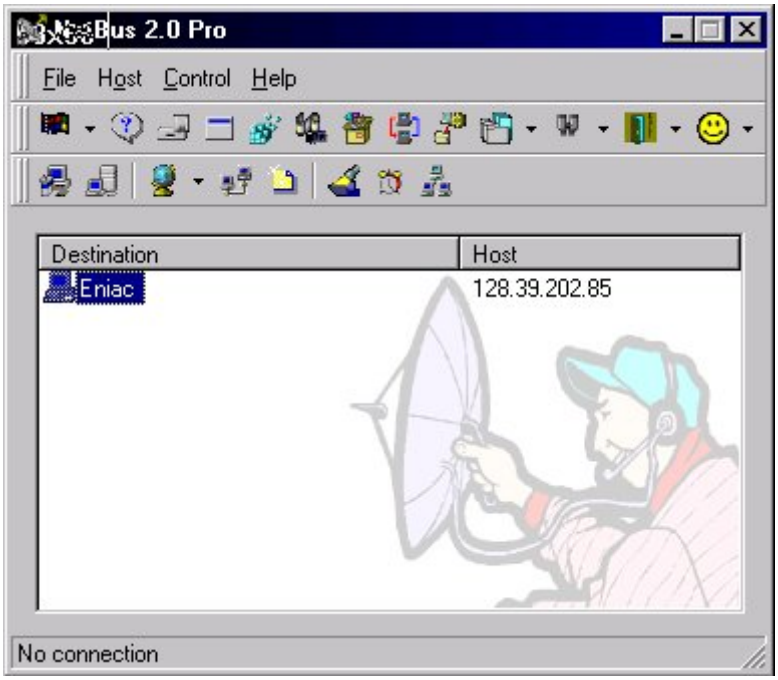


Fig. 5. Brukergrensesnittet for Netbus 2.0 Pro klienten.

6 Deteksjon av Netbus

6.1 Teknisk beskrivelse av Netbus

Netbus fungere etter klient/tjener-prinsippet, og baserer seg på TCP/IP protokollene. I motsetning til Back Orifice som benytter en svak form for kryptering av pakkene, er Netbus pakkene ukryptert. Netbus til og med versjon 1.6 benytter faste TCP porter for kommunikasjon. Dette er portene 12345 og 12346: Netbus serveren lytter på port 12345 etter en klient-forespørsel, og svarer via port 12346. Fra og med versjon 1.7 er det mulig å selv konfigurere hvilken port serveren skal benytte. Netbus 1.7 benytter som default de samme porter som de tidligere utgaver, mens versjon 2.0 Pro benytter port 20034 som default. Det er denne muligheten for å endre port som kompliserer deteksjonsfasen. Dersom det bare hadde blitt benyttet faste porter, ville et enkelt filter som overvåker trafikken på disse portene vært nok til å detektere Netbus.

6.2 Netbus signaturer

Det ble i kap.2 gitt en kort beskrivelse av arbeidsmetoden som ble benyttet for å komme frem til fungerende filtre for deteksjon av Netbus. En mer utfyllende beskrivelse av ideen bak deteksjon, signaturer og det praktiske arbeidet følger i de neste avsnitt.

6.2.1 Deteksjonshypotesen

Velkjente DoS angrep utnytter som tidligere nevnt som regel svakheter, feil eller mangler i implementasjoner av protokoller. Datapakkene som utveksles i forbindelse med et DoS angrep vil derfor som regel være av en slik karakter at man sjelden finner dem i naturlige sammenhenger. Som en følge av dette, vil det ofte være spesielle mønstre i datapakkenes headere som kan benyttes av et NIDS som ”signatur” for å kjenne igjen disse angrepene. Trojanske hester er derimot egne programmer, dog med en uønsket og utilsiktet funksjon. Deres skjulte funksjoner aktiviseres ikke før en bruker bevisst eksekverer dem. Programmet befinner seg da allerede fysisk på offerets PC, ofte spredd dit som vedlegg til E-post. Trojanske hester med fjernstyringsfunksjonalitet slik som Netbus og Back Orifice, består som nevnt tidligere av 2 applikasjoner: En server og en klient. Under en oppkobling utveksles det datapakker mellom disse to partene. Teorien er da at det er mulig å finne en eller annen form for mønster i trafikken mellom klient og server som kan tjene som signatur for å unikt

detektere en oppkobling. I motsetning til de ofte unaturlige pakkene som benyttes ved DoS angrep, er pakkene som utveksles under en Netbus oppkobling av mer ”normal” art. Det er derfor sannsynlig at man må lete blant nyttedata i pakkene for å finne en slik signatur, snarere enn i headerne.

6.2.2 Søken etter mulige signaturer

For å være i stand til å finne de eventuelle signaturer som kan detektere en Netbus oppkobling, ble det satt opp et separat, isolert nettverk for eksperimentering, som beskrevet i kap. 4. Baktanken med dette var å installere Netbus på en hhv. mål- og angrepsmaskin, og deretter kjøre Netbus forbindelser under kontrollerte forhold, samtidig som denne forbindelsen ble overvåket av en pakkesniffer. Til dette formål ble gratisprogrammet Hoppa Analyzer[7] benyttet. Siden Netbus tom. versjon 1.6 benytter faste TCP-porter, kan man strengt tatt detektere disse ved å lage et enkelt filter som overvåker disse portene. Ulempen med dette er at **all** trafikk på disse portene vil utløse en alarm, ikke bare trafikk relatert til Netbus. Det var derfor ønskelig å finne signaturer for disse versjoner også. Det var da naturlig å ta dem for seg i kronologisk rekkefølge. I de følgende avsnitt vil det bli presentert utdrag fra log-filer generert av Hoppa Analyzer. Angrepsmaskinen (Netbus klient) har her IP adresse **128.39.202.87**, mens målmaskinen (Netbus server) har IP adresse **128.39.202.85**.

6.2.2.1 Signatur for Netbus 1.2

Logging av trafikken mellom klient og server viser et hendelsesforløp under oppkobling som følger:

1. Klient sender en SYN pakke til server.
2. Server svarer med en SYN ACK pakke til klient.
3. Klient svarer med en ACK pakke til server.

Så langt altså en vanlig 3-way handshake oppkoblingsprosedyre. Det som så skjer, er at server sender en pakke med teksten ”NetBus.” i nyttedata. Følgende utdrag fra Hoppa Analyzer viser dette. Se vedlegg 18 for log av hele sesjonen. Loggene fra Hoppa Analyzer viser både verdiene i TCP-headeren og IP-headeren. Verdiene som står oppført under HEX data er innholdet i TCP-payload.

Utdrag av log for Netbus 1.2 (Vedlegg 18)

```
--- Packet received: 15:34:38.65 --- Length: 0061 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0047d
IP 16 bits Identification:       4100h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       24CDh
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d TCP 16 bits destination port:     1399d
TCP 32 bits sequence number: 00010E3Eh  TCP 32 bits ack number:       00010D8Dh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            61F9h  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 0D                                     NetBus.
```

En nærliggende tanke nå er å benytte nettopp denne tekststrengen som unik signatur på en Netbus 1.2 oppkobling. Dette kan imidlertid bare gjøres dersom ingen av de andre versjonene av Netbus inneholder nøyaktig denne tekststrengen på tilsvarende plass i nyttedata. Prosessen med logging av trafikk mellom klient og server måtte derfor repeteres med de andre Netbus versjonene. Likheter i resultater for de andre versjoner tom. 1.7 gjør at disse omtales samlet i neste avsnitt.

6.2.2.2 Signaturer for Netbus 1.5x - 1.7

Analyse av trafikken etter samme metode som ved Netbus 1.2 avdekket at også versjon 1.53, 1.6 og 1.7 benytter en 3-way handshake oppkoblingsprosedyre. Det som så skjer, er at server sender en pakke med hhv. ”NetBus 1.53 .”,”NetBus 1.60 .” og ”NetBus 1.70 .” i nyttedata. Følgende 3 utdrag fra log-filene viser dette.

Utdrag av log for Netbus 1.53 (Vedlegg 19)

```
--- Packet received: 14:53:07.41 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0053d
IP 16 bits Identification:       0D42h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        5885h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d TCP 16 bits destination port:      1373d
TCP 32 bits sequence number: 00010E37h  TCP 32 bits ack number:        00010CE0h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E13Ah  TCP 16 bits urgent pointer:        0000h
HEX data:                                                   ASCII data:
4E 65 74 42 75 73 20 31 2E 35 33 20 0D                      NetBus 1.53 .
```

Utdrag av log for Netbus 1.6 (Vedlegg 20)

```
--- Packet received: 14:39:31.77 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0053d
IP 16 bits Identification:       343Fh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        3188h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d TCP 16 bits destination port:      1356d
TCP 32 bits sequence number: 00010DCCh  TCP 32 bits ack number:        00010C5Ch
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E539h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                   ASCII data:
4E 65 74 42 75 73 20 31 2E 36 30 20 0D                      NetBus 1.60 .
```

Utdrag av log for Netbus 1.7 (Vedlegg 21)

```
--- Packet received: 14:24:49.31 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0053d
IP 16 bits Identification:       093Fh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        5C88h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d TCP 16 bits destination port:      1352d
TCP 32 bits sequence number: 00010DA5h  TCP 32 bits ack number:        00010C4Ah
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E575h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                   ASCII data:
4E 65 74 42 75 73 20 31 2E 37 30 20 0D                      NetBus 1.70 .
```

Man har nå kommet så langt at man har identifisert mønstre som kan tjene som potensielle signaturer for å unikt identifisere de ulike versjoner av Netbus. Det er imidlertid enkelte spørsmål som må avklares før man kan anta at disse mønstre vil identifisere Netbus med en stor grad av nøyaktighet:

- Kan det tenkes at endringer av server-parametre vil påvirke tekststrengen?
- Skal man eventuelt benytte **hele** tekststrengen som signatur?

For å kunne svare på det første spørsmålet må man gjenta prosessen med logging av forbindelsene med ulike server-parametre. Netbus 1.2 og 1.53 tilbyr ikke konfigurerbare server-instillinger, og følgelig bortfaller problemstillingen for deres vedkommende. Netbus 1.6 tilbyr passordbeskyttelse av serveren. Logging av forbindelsen med passordbeskyttet server avslørte da også at tekststrengen påvirkes av denne parameteren. Teksten i nyttedata endret seg fra ”NetBus 1.60 .” til ”NetBus 1.60 x.”, som vist i nedenforstående utdrag.

Utdrag fra log for Netbus 1.6 med passord

```
--- Packet received: 14:39:31.77 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0053d
IP 16 bits Identification:       343Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       3188h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d TCP 16 bits destination port:     1356d
TCP 32 bits sequence number: 00010DCCh  TCP 32 bits ack number:       00010C5Ch
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E539h  TCP 16 bits urgent pointer:       0000h
HEX data:                                                 ASCII data:
4E 65 74 42 75 73 20 31 2E 36 30 20 78 0D                 NetBus 1.60 x.
```

Eksperimentering med ulike passord viser at teksten kun er avhengig av om passord benyttes eller ikke, passordets konstruksjon påvirker ikke tekststrengen. Tekststrengen for Netbus 1.7 endrer seg på nøyaktig tilsvarende måte fra ”NetBus 1.70 .” til ”NetBus 1.70 x.” ved benyttelse av passord. Man ser her at ved å innskrenke signaturstrengen til respektive ”NetBus 1.60” og ”NetBus 1.70" vil man detektere servere både med og uten passord.

Når det gjelder spørsmålet om **hele** tekststrengen skal benyttes som signatur, ser man ved å kaste et blikk på historikken til Netbus fra avsnitt 7.5.1 at det i april’98 ble lansert versjon **1.5x** av Netbus. Dette betyr at selv om det er Netbus 1.53 som er mest utbredt av disse, og som var den eneste utgaven i 1.5 serien kandidaten klarte å oppdrive, finnes det sannsynligvis

andre versjoner i serien i sirkulasjon. Innskrenker man signaturstrengen fra ”NetBus 1.53 .” til ”NetBus 1.5” vil man også være i stand til å detektere andre versjoner i 1.5 serien, forutsatt at de følger ”normalen” og inneholder strenger som ”NetBus 1.51 .”, ”NetBus 1.52 .” osv. i nyttedata. Dette er det all grunn til å anta. Med samme begrunnelse kan det være hensiktsmessig å innskrenke signaturstrengen for Netbus 1.6 og 1.7 til hhv. ”NetBus 1.6” og ”NetBus 1.7”.

Man kan nå sammenfatte de ovennevnte oppdagelser i en tabell.

Netbus versjon	Signatur Hex verdi	ASCII symbol
Netbus 1.2	4E 65 74 42 75 73 0D	NetBus.
Netbus 1.5x	4E 65 74 42 75 73 20 31 2E 35	NetBus 1.5
Netbus 1.60	4E 65 74 42 75 73 20 31 2E 36	NetBus 1.6
Netbus 1.70	4E 65 74 42 75 73 20 31 2E 37	NetBus 1.7

Tabell 1. Oversikt over potensielle signaturer og deres ASCII verdier.

6.2.2.3 Signatur for Netbus 2.0 Pro

Det skulle vise seg å være noe mer problematisk å finne en egnet signatur for Netbus 2.0 Pro. Grunnen til dette, er at denne versjonen ikke sender navnet sitt etter at 3-way handshake er utført. Logger fra Netbus 2.0 Pro forbindelsen viser et noe annerledes hendelsesforløp enn for de tidligere versjoner: Etter at det har blitt foretatt en 3-way handshake oppkoblingsprosedyre, sender **klient** en pakke med **eksemplevis** følgende informasjon i nyttedata:
(Her har server-maskinen IP adresse 128.39.202.70)

Utdrag av log for Netbus 2.0 Pro (Vedlegg 22)

```
--- Packet received: 17:57:42.54 --- Length: 0086 --- Assigned number: 00009 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0072d
IP 16 bits Identification:       878Dh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       DE25h
IP source address: 128.39.202.87        IP destination address: 128.39.202.70
TCP 16 bits source port:         4379d  TCP 16 bits destination port:     20034d
TCP 32 bits sequence number: 0002971Dh  TCP 32 bits ack number:       000128F0h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            068Ah  TCP 16 bits urgent pointer:       0000h
HEX data:                                                 ASCII data:
42 4E 20 00 02 00 08 08 05 00 41 0C 69 1F 5D 28 5B BC 62 AE BN .......A.i.]([.b.
64 B7 6E 84 8C BC 68 AF 3E F2 93 E0                       d.n...h.>...
```

Server svarer så **eksempelvis** med følgende pakke:

Utdrag av log for Netbus 2.0 Pro (Vedlegg 22)

```
--- Packet received: 17:57:42.67 --- Length: 0070 --- Assigned number: 00011 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0056d
IP 16 bits Identification:       2A52h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        3B81h
IP source address: 128.39.202.70        IP destination address: 128.39.202.87
TCP 16 bits source port:         20034d TCP 16 bits destination port:      4379d
TCP 32 bits sequence number: 000128F0h  TCP 32 bits ack number:        0002973Dh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2218h
TCP 16 bits checksum:            69B9h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                 ASCII data:
42 4E 10 00 02 00 0C CE 05 00 41 0C 6B 1F 5D 28           BN........A.k.](
```

Dette kan tilsynelatende se ut som faste mønstre, om enn noe mindre selvforklarende enn ved de tidligere utgaver av Netbus. Ved eksperimentering med server parametre viser gjentatte repetisjoner av logge-prosessen imidlertid at enkelte verdier i nyttedata endrer seg hver gang, mens andre ser ut til å være faste. I pakken sendt av **klienten** observeres følgende i nyttedata:

- Byte 3 har alltid verdien 20 når det ikke benyttes passord på serveren. Ved passord er verdien avhengig av passordet.
- Byte 7 og 8 opptrer med forskjellige verdier hver gang, og ser ut til å være tilfeldig generert.

I pakken som **server** sender tilbake observeres følgende i nyttedata:

- Byte 7 og 8 opptrer med forskjellige verdier hver gang, og ser ut til å være tilfeldig generert.

Selv om disse observasjonene gjør det noe mer komplisert å finne et mønster som kan benyttes som signatur, kan man i dette tilfelle tenke seg at **summen** av de faste bytene kan fungere som signatur. Filteret som lages vil da basere seg på å maske ut verdiene på de tilsvarende plasseringer i IP pakkene (husk at TCP-pakkene er kapslet inn i IP pakker), disse verdiene summeres, og sammenliknes med Netbus 2.0 Pro signatur verdiene. Hvis verdiene er like, kan det med stor sikkerhet antas at Netbus 2.0 Pro er detektert. For større nøyaktighet er

det gunstig at **både** verdiene fra klient- og serverpakken må matches for at alarmen skal gå. Det er lite trolig at andre applikasjoner skal klare å tilfeldigvis få samme verdier på alle de faste plasseringene i nyttedata, eller at summen av dem tilfeldigvis skal bli lik signaturene. Fordi pakken fra klienten inneholder 29 byte med faste verdier, er det et spørsmål om filtereffektivitet om man skal la alle inngå i signaturen. Dette anses som ”overkill”, og det ble besluttet å kun la de 14 første faste bytene inngå i signaturen (14 fordi det er 14 faste bytes i server-pakken).

”Klient-signatur” blir derfor summen av følgende verdier, der de avmerkede verdier er utelatt:
42 4E 20 00 02 00 08 08 05 00 41 0C 69 1F 5D 28 5B = 24C
Tilsvarende blir ”Server-signatur” summen av verdiene:
42 4E 10 00 02 00 0C CE 05 00 41 0C 6B 1F 5D 28 = 203

6.2.3 Endelige signaturer

Man har nå kommet frem til en gruppe signaturer som trolig vil være i stand til å unikt kunne identifisere de ulike utgavene av Netbus. Om dette medfører riktighet kan man imidlertid ikke si med sikkerhet før filtrene er laget og grundig testet i praksis. Neste kapittel beskriver disse filtrene og implementeringen av dem. Før det sammenfattes de endelige signaturer i en tabell.

Netbus versjon	Signatur Hex verdi	ASCII symbol
Netbus 1.2	4E 65 74 42 75 73 0D	NetBus.
Netbus 1.5x	4E 65 74 42 75 73 20 31 2E 35	NetBus 1.5
Netbus 1.60	4E 65 74 42 75 73 20 31 2E 36	NetBus 1.6
Netbus 1.70	4E 65 74 42 75 73 20 31 2E 37	NetBus 1.7
Netbus 2.0 Pro	Klient: 24C , Server: 203	

Tabell 2. Oversikt over potensielle signaturer og deres ASCII verdier.

7 Implementering av filtre

7.1 Filter scenarier

Før implementeringen av filtre startet, var det et essensielt spørsmål som måtte avklares: **Hvor mange filtre skulle lages?** Ved å kaste et blikk på signaturene funnet i kap. 6, var det tilsynelatende 3 scenarier som måtte vurderes:

En mulighet var å lage **ett** filter som kikker etter alle signaturene funnet i kap. 6. Ulempen med dette er at brukeren av NFR systemet ikke har muligheten til velge hvilke Netbus versjoner det skal overvåkes for: Enten så ser man etter alle utgavene eller ingen av dem. Filteret ville i dette tilfelle trolig også blitt uoversiktlig for andre.

Siden det er store likheter i signaturene for Netbus 1.2 –1.7, var neste nærliggende alternativ å lage **to** filtre: Et filter som detekterer Netbus 1.2 –1.7, og et filter som detekterer Netbus 2.0 Pro. Dette alternativet gir noe større valgfrihet en det første alternativet, men det er fremdeles ønskelig med en større grad av frihet ved valg av aktive filtre. Dette ønsket fører frem til det tredje alternativet.

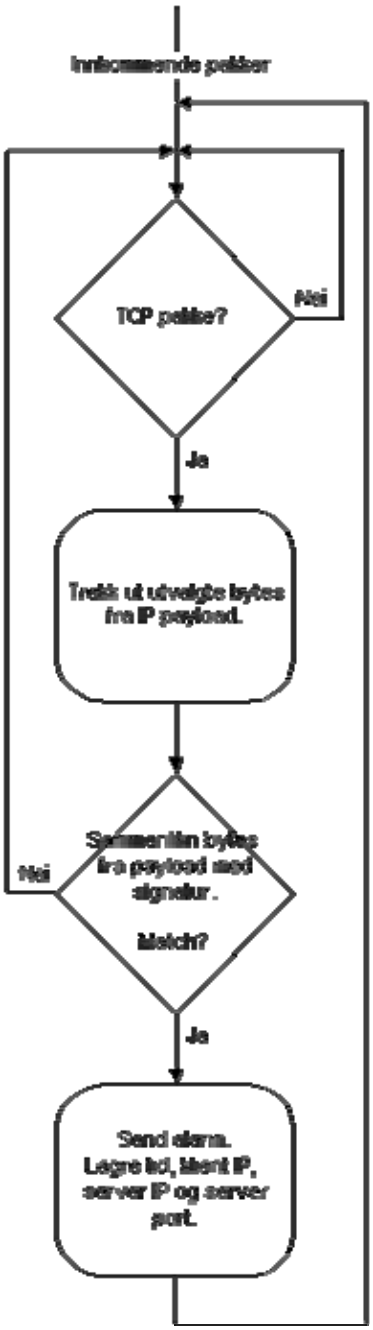
For å kunne tilby brukere av NFR full frihet til å velge hvilke versjoner av Netbus de ønsker å detektere, må det lages fem filtre: Et for hver versjon av Netbus. Dette vil medføre noe mer programmeringsarbeid, men belønningen blir kortere filtre som vil være lettere å modifisere i ettertid. Det er denne løsningen som har blitt valgt for det videre filterarbeidet.

7.2 Filtrene

I de følgende avsnitt vil de ulike filtre bli beskrevet. Likheter i filteroppbygningen for Netbus versjoner tom. 1.7 gjør at disse vil bli omtalt samlet, med utgangspunkt i filteret for Netbus 1.2. Variasjonene mellom de ulike versjoner vil så bli omtalt spesielt. Siden filtre er hovedkomponenten i såkalte backends som også består av en beskrivelsesfil og en konfigurasjosfil, må disse også lages. Det vil derimot ikke bli fokusert på innholdet av disse.

7.2.1 Filter for deteksjon av Netbus 1.2

Den prinsipielle virkemåten for filteret som detekterer Netbus 1.2 kan best illustreres ved et flytdiagram.



Videre følger en beskrivelse av filteret med kodehenvisninger. Se vedlegg 4 for en komplett kildekode for Netbus 1.2 filteret.

Filteret er lagd slik at det kun aktiviserer/trigger på TCP pakker, dette gjøres ved kodebiten

filter netbus_detect tcp ()

Deretter leses utvalgte verdier ut av pakken, og lagres i en lokal variabel. Tankegangen er at disse verdier skal sammenliknes med signaturen for Netbus som man har funnet i forrige kapittel. Koden som leser ut verdier er:

$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+ byte(ip.blob,24)+byte(ip.blob,25)+byte(ip.blob,26);

Denne koden trenger imidlertid noe forklaring: TCP protokollen benytter seg av den underliggende protokollen IP. Dette betyr at TCP pakkene kapsles inn i IP pakker før de overføres på nettverket. I N-Code har man mulighet til å benytte ulike pakke-variable for å trekke ut informasjon av pakker. En av disse pakke-variablene heter *ip.blob*, og trekker ut nyttedata (payload) fra IP pakker. Funksjonen *byte* returnerer en byte med en gitt offset fra en blob. Offset teller fra byte 0. Koden *byte(ip.blob,20)* vil derfor returnere verdien i byte **21** i IP payload. Fordi TCP header kan ha variabel lengde, angis lengden av headeren av feltet data offset. Nedenstående utdrag fra log-filen for Netbus 1.2 forbindelsen viser at dette feltet i dette tilfelle har verdien 5. Denne verdien betegner imidlertid antall 32-bits ord, dvs. bolker à 4 bytes. TCP headeren er i dette tilfelle 20 bytes. Vi ønsker derfor å lese ut byte 21 tom. 27 i alle IP pakker (som inneholder TCP pakker) for å sammenlikne disse med verdiene 4E 65 74 42 75 73 0D.
Siden offset teller fra byte 0, blir dette byte 20 tom. 26 ved benyttelse av funksjonen *byte*.

Utdrag fra TCP-header for Netbus 1.2 forbindelsen.

```
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            61F9h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 0D                                     NetBus.
```

Etter at verdiene fra TCP pakkene har blitt trukket ut og lagret i den lokale variabelen *temp_payload1*, sammenliknes summen av dem med den kjente signaturen for Netbus 1.2. Summen av signaturen for Netbus 1.2 blir desimaltallet 606 (25E Hex). Denne summen ligger i en annen lokal variabel kalt *server_response*.

Er disse to variable like, sendes en alarm til alert manager, samtidig som tidspunkt, klient IP, server IP og server port lagres i en egen recorder for Netbus 1.2.
Koden som utfører disse hendelser er:

```
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$server_response=606;                    # Den hexadesimale verdien 25E.

if ($temp_payload1==$server_response)
        {
        $klient_ip=ip.dest;
        $server_ip=ip.src;
        alert(_:NETBUS,_:NETBUS_12_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_12;
        }
```

7.2.2 Filtre for deteksjon av Netbus 1.5x – 1.7

For å detektere Netbus 1.5x – 1.7 benyttes akkurat den samme filtermetoden som for Netbus 1.2. Eneste forskjellen er at summen av signaturen er annerledes, samt at det er andre bytes som leses ut av IP payload. Summen av signaturene for disse versjoner er gitt av følgende tabell:

Netbus versjon	Signatur Hex verdi	Sum av signaturen
Netbus 1.5x	4E 65 74 42 75 73 20 31 2E 35	773 (305 Hex)
Netbus 1.60	4E 65 74 42 75 73 20 31 2E 36	774 (306 Hex)
Netbus 1.70	4E 65 74 42 75 73 20 31 2E 37	775 (307 Hex)

Tabell 3. Oversikt over signaturer og summen av dem.

Dette betyr at verdien av variabelen *server_response* i disse filtrene endres fra 606 til de respektive verdier gitt av tabellen. Som man ser, er det bare siste byte i signaturen som endres. I disse filtrene er det ønskelig å lese ut byte 21 tom. byte 30 i alle IP pakker som inneholder TCP pakker. Dette fører til at koden som leser ut verdier fra pakken i alle disse filtre blir:

```
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+
byte(ip.blob,24)+byte(ip.blob,25)+byte(ip.blob,26)+byte(ip.blob,27)+byte(ip.blob,28)+
byte(ip.blob,29);
```

Se vedlegg 7, 10 og 13 for komplett kildekode for disse filtre.

7.2.3 Filter for deteksjon av Netbus 2.0 Pro

Filteret for deteksjon av Netbus 2.0 Pro fungerer etter et liknende prinsipp som de andre filterne. Hovedforskjellen er imidlertid at dette filteret ser etter **to** signaturer og ikke en, slik

som er tilfelle med de andre filterne. Fra kap. 6 husker vi at det ble funnet en signatur i pakken som sendes av klienten etter 3-way handshake, og en annen signatur i pakken som server svarer med. For større nøyaktighet kreves begge signaturer detektert før man konstaterer at Netbus 2.0 Pro er detektert. På samme måte som de andre filtrene, er også dette filteret laget slik at det kun aktiviseres ved TCP pakker. Videre leses utvalgte verdier ut av pakken og lagres i lokale variable. Siden man her ser etter to signaturer, må man også trekke ut to ulike sett av bytes fra payload, for å kunne sjekke om man eventuelt har påvist klient eller server-pakken. Koden som trekker ut de to sett av bytes er som følger:

```
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,23)+byte(ip.blob,24)+
byte(ip.blob,25)+byte(ip.blob,28)+byte(ip.blob,29)+byte(ip.blob,30)+byte(ip.blob,31)+
byte(ip.blob,32)+byte(ip.blob,33)+byte(ip.blob,34)+byte(ip.blob,35)+byte(ip.blob,36);

$temp_payload2=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+
byte(ip.blob,24)+byte(ip.blob,25)+byte(ip.blob,28)+byte(ip.blob,29)+byte(ip.blob,30)+
byte(ip.blob,31)+byte(ip.blob,32)+byte(ip.blob,33)+byte(ip.blob,34)+byte(ip.blob,35);
```

Disse to variablene sammenliknes så med hhv. klient- og server signaturen, definert av følgende kode:

```
$client_fingerprint=588;          # Den hexadesimale verdien 24C.
$server_fingerprint=515;          # Den hexadesimale verdien 203.
```

Dersom *temp_payload1* er lik klient signaturen har man detektert pakken som sendes av klient rett etter 3-way handshake. En lokal variabel settes da til verdien 1. Filteret vil nå se etter svarpakken fra server for å kunne gi alarm om Netbus oppkobling. Dersom *temp_payload2* senere er lik server signaturen, settes en annen lokal variabel til verdien 1, og man har detektert pakken som sendes av server som respons på klient pakken. Koden som gjør dette er som følger:

```
if ($temp_payload1==$client_fingerprint)
        {
        $klient_request=1;
        $klient_ip=ip.src;
        }

if ($temp_payload2==$server_fingerprint)
        {
        $server_response=1;
        $server_ip=ip.src;
        }
```

For hver gang filteret eksekveres, dvs. hver gang det kommer en TCP pakke på nettverket, sjekkes det om **både** klient- og server signaturen/pakken er detektert. Er dette tilfelle, har man detektert en Netbus 2.0 Pro oppkobling, og en alarm genereres. Det lagres også informasjon om tid, IP adresser og serverport i en egen recorder, slik tilfellet er med de andre filtrene. Til slutt resettes variablene som holder rede på om signaturene er detektert. Koden som utfører alt dette er vist nedenfor.

```
if (($klient_request&&$server_response)!=0)
        {
        alert(_:NETBUS,_:NETBUS_2_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_2;
        $klient_request=0;
        $server_response=0;
        }
```

7.3 Testing av filtrene

Man har nå utviklet 5 filtre for deteksjon av de ulike versjoner av Netbus. Før man med stor grad av sikkerhet kan anta at disse filtre faktisk fungerer tilfredsstillende, må de testes i praksis. Det er da spesielt to faktorer som er av betydning:

1. Oppdager filtrene faktisk de Netbus versjoner de er konstruert for?
2. Forekommer det falske alarmer?

En utvidet testing av alle filtre mot de berørte Netbus versjoner avslører at filtrene ser ut til å detektere Netbus oppkoblinger med stor nøyaktighet. Etter flere dagers uttesting på testnettverket har deteksjonsprosenten vist seg å være 100%. Når det gjelder problemstillingen med falske alarmer, er det 2 tilfeller det dette kan tenkes å forekomme:

- Dersom signaturstrengen skulle forekomme først i nyttedata på et senere tidspunkt i en oppkobling. Man vil da få utløst en ekstra alarm på en allerede detektert oppkobling. I et forsøk på å finne ut om dette kunne inntreffe, ble det eksperimentert vel og lenge med **alle** funksjoner i de ulike Netbus utgaver. Ingen falske alarmer ble utløst i denne fasen, men dette gir ingen 100% garanti for at dette ikke **kan** tenkes å inntreffe under spesielle omstendigheter.
- Dersom summen av andre kombinasjoner av de utleste verdier fra IP pakkene **tilfeldigvis** skulle få samme verdi som de gitte signaturer. Sannsynligheten for at dette skal inntreffe er forsvinnende liten. Hvis dette særtilfelle allikevel skulle oppstå, vil man få en falsk alarm uten mulighet til å oppdage dette.

8 Diskusjon og Konklusjon

8.1 Diskusjon

Denne oppgaven har vist at NFR godt kan benyttes til å oppdage trojanske hester. Det er imidlertid noen begrensninger knyttet til dette. NFR er avhengig av å kjenne signaturen til de ulike trojanske hester for å være i stand til å detektere dem. Det kan tenkes tilfeller der det er umulig eller betydelig vanskeligere å finne faste mønstre i pakkene som kan benyttes som signaturer. Dette vil sannsynligvis være tilfellet dersom det benyttes kryptering, idet innholdet i nyttedata da vil være dynamisk.

8.2 Konklusjon

Som en følge av en økt utbredelse av datasystemer og det faktum at disse systemene i økende grad er blitt en del av Internett, har behovet for å kunne beskytte seg mot inntrengere og angrep økt betraktelig de siste årene. Den store utbredelsen man har sett av populære klient/tjener baserte trojanske hester som Back Orifice og Netbus bare forsterker dette inntrykket. Kombinasjonen av deres evne til å installere seg i det skjulte og på toppen av det hele kompromittere systemer totalt, gjør det spesielt attraktivt å utvikle filtre for deteksjon av denne typen trojanske hester. De resulterende filtre for deteksjon av Netbus har vist at NFR kan egne seg godt for deteksjon av slike klient/tjener baserte trojaner. Filterne detekterer de ulike versjoner av Netbus med en meget stor grad av nøyaktighet. Jeg har gjennom mitt arbeid vist en generell metode for å identifisere mønstre som kan tjene som signaturer. Arbeidet med å finne egnede signaturer kan imidlertid være en tidkrevende affære. Det er imidlertid enkelt å innse begrensningene til denne metodikken. Metoden vil trolig ikke føre frem dersom det benyttes kryptering av pakkene, men som modell og inspirasjon for å lage filtre for deteksjon av andre liknende trojanske hester har oppgaven forhåpentlig tjent sin hensikt.

9 Litteraturreferanser

[1] Robert Graham:
***FAQ: Network Intrusion Detection Systems**, 1999.*
http://www.ticm.com/kb/faq/idsfaq.html

[2] ***The Free On-line Dictionary of Computing***
http://www.instantweb.com/foldoc/foldoc.cgi?Trojan+horse

[3] AcidMeister:
***Trojans**, Digital Rebels magazine 02, 20.7.98.*
http://hackersclub.com/km/library/hack/rebelmag/trojans.htm

[4] P. Holbrook & J. Reynolds:
***RFC 1244, the Site Security Handbook**, Network Working Group, 1991.*
http://www.net.ohio-state.edu/hypertext/rfc1244/toc.html

[5] Marcus J. Ranum, Kent Landfield, Mike Stolarchuk, Mark Sienkiewicz, Andrew Lambert og Eric Wall:
***Implementing A Generalized Tool For Network Monitoring**, Network Flight Recorder, Inc.*
http://www.nfr.net/forum/publications/LISA-97.htm.

[6] Lawrence Berkeley National Labs Network Research Group:
***libpcap**.*
http://ftp.ee.lbl.gov/

[7] ***The Official Hoppa Homepage**.*
http://surf.to/hoppa

[8] Network Flight Recorder,Inc:
***NFR Architecture**, NFR Library on-line documentation.*
http://www.nfr.net/nfr/nfr-2.0.2-research/nfrlibrary/user-guide/overview/architecture.htm

[9] Norman ASA:
***File and system virus - description**, Norman technical.*
http://www.norman.no/corporate/virus/file_boot_virus.htm

[9] Norman ASA:
***Attack against weaknesses in the TCP/IP protocol**, Norman technical.*
http://www.norman.no/corporate/documents/wp_smurf.htm

[10] http://www.rootshell.com/

10 Vedlegg

Vedlegg 1: netbus.cfg - Konfigurasjonsfil for Netbus filter package

```
enabled=true
title=Netbus filters
```

Vedlegg 2: netbus.src - Alarm fil for Netbus filter package

```
_alerts _default_alerts 50000 - 65535 "User alerts"
{
	:NETBUS_12_DETECTED
	{
		NFR_FAC_ACK NFR_FAC_NFRLOG;
		SEV_WARNING;
		"Netbus 1.2 oppkobling fra ip $(1) mot ip $(2)";
	}

	:NETBUS_15x_DETECTED
	{
		NFR_FAC_ACK NFR_FAC_NFRLOG;
		SEV_WARNING;
		"Netbus 1.5x oppkobling fra ip $(1) mot ip $(2)";
	}

	:NETBUS_16_DETECTED
	{
		NFR_FAC_ACK NFR_FAC_NFRLOG;
		SEV_WARNING;
		"Netbus 1.6 oppkobling fra ip $(1) mot ip $(2)";
	}

	:NETBUS_17_DETECTED
	{
		NFR_FAC_ACK NFR_FAC_NFRLOG;
		SEV_WARNING;
		"Netbus 1.7 oppkobling fra ip $(1) mot ip $(2)";
	}

	:NETBUS_2_DETECTED
	{
		NFR_FAC_ACK NFR_FAC_NFRLOG;
		SEV_WARNING;
		"Netbus 2.0 Pro oppkobling fra ip $(1) mot ip $(2)";
	}
}

_alert_srcs _default_alert_srcs 50000 - 65535 "User alert sources"
{
	:NETBUS
	{
		"Netbus detektor";
		"Netbus detektor";
	}
}
```

Vedlegg 3: netbus_12.desc - Beskrivelsefil for netbus 1.2 filter

Dette filteret detekterer Netbus 1.2.

Vedlegg 4: netbus_12.nfr - Filter for Netbus 1.2

```
#
# Netbus 1.2 deteksjon
# Av Per Kristian Johnsen
# Email: pkjohnsen@hotmail.com
#

netbus_12_schema = library_schema:new(1,["time","ip","ip","int"],scope());

# Trigger paa tcp pakker.
filter netbus_detect tcp ()
{
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$server_response=606;                    # Den hexadesimale verdien 25E.

# Trekker ut utvalgte byte fra payload i en TCP-pakke.
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+byte(ip.blob,2
4)+byte(ip.blob,25)+byte(ip.blob,26);

if ($temp_payload1==$server_response)
        {
        $klient_ip=ip.dest;
        $server_ip=ip.src;
        alert(_:NETBUS,_:NETBUS_12_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_12;
        }
}

# Dette skaper den recorderen som skal brukes.
the_recorder_netbus_12=recorder("bin/histogram %c","netbus_12_schema");
```

Vedlegg 5: netbus_12.cfg - Konfigurasjonsfil for Netbus 1.2 filter

```
#
#

enabled=true
gui=histogram

#
num_columns=3

#
column_1_type=p_src_ip
column_2_type=p_dst_ip
column_3_type=p_dst_port

#
column_1_label=Netbus klient addresse
column_2_label=Netbus server addresse
column_3_label=Netbus server port

#
count_label=Netbus oppkoblingsforsok

#
rollover=300

#
suppress_all_time=false

#
sync_time=10

#
modified=true
origin=pkjohnsen@hotmail.com
title=Netbus 1.2 detection module

#
doalerts=true
alert_text=Nytt oppkoblingsforsok : $(1)
#
new_cell_alerts=false
#
cfversion=2
rollover_size=YES
rollover_size_val=1024000
rollover_time=YES
rollover_time_val=300000
#
archive_path=data/%p/%b/%y/%m%d/

data_label=
```

Vedlegg 6: netbus_15x.desc - Beskrivelsesfil for Netbus 1.5x filter

Dette filteret detekterer Netbus 1.5x.

Vedlegg 7: netbus_15x.nfr - Filter for Netbus 1.5x

```
#
# Netbus 1.5x deteksjon
# Av Per Kristian Johnsen
# Email: pkjohnsen@hotmail.com
#

netbus_15x_schema = library_schema:new(1,["time","ip","ip","int"],scope());

# Trigger paa tcp pakker.
filter netbus_detect tcp ()
{
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$server_response=773;                    # Den hexadesimale verdien 305.

# Trekker ut utvalgte byte fra payload i en TCP-pakke.
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+byte(ip.blob,2
4)+byte(ip.blob,25)+byte(ip.blob,26)+byte(ip.blob,27)+byte(ip.blob,28)+byte(ip.blob,29);

if ($temp_payload1==$server_response)
        {
        $klient_ip=ip.dest;
        $server_ip=ip.src;
        alert(_:NETBUS,_:NETBUS_15x_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_15x;
        }
}

# Dette skaper den recorderen som skal brukes.
the_recorder_netbus_15x=recorder("bin/histogram %c","netbus_15x_schema");
```

Vedlegg 8: netbus_15x.cfg - Konfigurasjonsfil for Netbus 1.5x filter

```
#
#

enabled=true
gui=histogram

#
num_columns=3

#
column_1_type=p_src_ip
column_2_type=p_dst_ip
column_3_type=p_dst_port

#
column_1_label=Netbus klient addresse
column_2_label=Netbus server addresse
column_3_label=Netbus server port

#
count_label=Netbus oppkoblingsforsok

#
rollover=300

#
suppress_all_time=false

#
sync_time=10

#
modified=true
origin=pkjohnsen@hotmail.com
title=Netbus 1.5x detection module

#
doalerts=true
alert_text=Nytt oppkoblingsforsok : $(1)

#
new_cell_alerts=false
cfversion=2
rollover_size=YES
rollover_size_val=1024000
rollover_time=YES
rollover_time_val=300000
#
archive_path=data/%p/%b/%y/%m%d/

data_label=
```

Vedlegg 9: netbus_16.desc - Beskrivelsesfil for Netbus 1.6 filter

Dette filteret detekterer Netbus 1.6.

Vedlegg 10: netbus_16.nfr - Filter for Netbus 1.6

```
#
# Netbus 1.6 deteksjon
# Av Per Kristian Johnsen
# Email: pkjohnsen@hotmail.com
#

netbus_16_schema = library_schema:new(1,["time","ip","ip","int"],scope());

# Trigger paa tcp pakker.
filter netbus_detect tcp ()
{
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$server_response=774;                    # Den hexadesimale verdien 306.

# Trekker ut utvalgte byte fra payload i en TCP-pakke.
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+byte(ip.blob,2
4)+byte(ip.blob,25)+byte(ip.blob,26)+byte(ip.blob,27)+byte(ip.blob,28)+byte(ip.blob,29);

if ($temp_payload1==$server_response)
        {
        $klient_ip=ip.dest;
        $server_ip=ip.src;
        alert(_:NETBUS,_:NETBUS_16_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_16;
        }
}

# Dette skaper den recorderen som skal brukes.
the_recorder_netbus_16=recorder("bin/histogram %c","netbus_16_schema");
```

Vedlegg 11: netbus_16.cfg - Konfigurasjonsfil for Netbus 1.6 filter

```
#
#

enabled=true
gui=histogram

#
num_columns=3

#
column_1_type=p_src_ip
column_2_type=p_dst_ip
column_3_type=p_dst_port

#
column_1_label=Netbus klient addresse
column_2_label=Netbus server addresse
column_3_label=Netbus server port

#
count_label=Netbus oppkoblingsforsok

#
rollover=300

#
suppress_all_time=false

#
sync_time=10

#
modified=true
origin=pkjohnsen@hotmail.com
title=Netbus 1.6 detection module

#
doalerts=true
alert_text=Nytt oppkoblingsforsok : $(1)
#
new_cell_alerts=false
#
cfversion=2
rollover_size=YES
rollover_size_val=1024000
rollover_time=YES
rollover_time_val=300000
#
archive_path=data/%p/%b/%y/%m%d/

data_label=
```

Vedlegg 12: netbus_17.desc - Beskrivelsesfil for Netbus 1.7 filter

Dette filteret detekterer Netbus 1.7.

Vedlegg 13: netbus_17.nfr - Filter for Netbus 1.7

```
#
# Netbus 1.7 deteksjon
# Av Per Kristian Johnsen
# Email: pkjohnsen@hotmail.com
#

netbus_17_schema = library_schema:new(1,["time","ip","ip","int"],scope());

# Trigger paa tcp pakker.
filter netbus_detect tcp ()
{
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$server_response=775;                    # Den hexadesimale verdien 307.

# Trekker ut utvalgte byte fra payload i en TCP-pakke.
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+byte(ip.blob,2
4)+byte(ip.blob,25)+byte(ip.blob,26)+byte(ip.blob,27)+byte(ip.blob,28)+byte(ip.blob,29);

if ($temp_payload1==$server_response)
        {
        $klient_ip=ip.dest;
        $server_ip=ip.src;
        alert(_:NETBUS,_:NETBUS_17_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_17;
        }
}

# Dette skaper den recorderen som skal brukes.
the_recorder_netbus_17=recorder("bin/histogram %c","netbus_17_schema");
```

Vedlegg 14: netbus_17.cfg - Konfigurasjonsfil for Netbus 1.7 filter

```
#
#

enabled=true
gui=histogram

#
num_columns=3

#
column_1_type=p_src_ip
column_2_type=p_dst_ip
column_3_type=p_dst_port

#
column_1_label=Netbus klient addresse
column_2_label=Netbus server addresse
column_3_label=Netbus server port

#
count_label=Netbus oppkoblingsforsok

#
rollover=300

#
suppress_all_time=false

#
sync_time=10

#
modified=true
origin=pkjohnsen@hotmail.com
title=Netbus 1.7 detection module

#
doalerts=true
alert_text=Nytt oppkoblingsforsok : $(1)
#
new_cell_alerts=false
#
cfversion=2
rollover_size=YES
rollover_size_val=1024000
rollover_time=YES
rollover_time_val=300000
#
archive_path=data/%p/%b/%y/%m%d/

data_label=
```

Vedlegg 15: netbus_2.desc - Beskrivelsesfil for Netbus 2.0 filter

Dette filter detekterer Netbus 2.0 Pro.

Vedlegg 16: netbus_2.nfr - Filter for Netbus 2.0

```
#
# Netbus 2.0 Pro deteksjon
# Av Per Kristian Johnsen
# Email: pkjohnsen@hotmail.com
#

netbus_2_schema = library_schema:new(1,["time","ip","ip","int"],scope());

# trigger paa tcp pakker.
filter netbus_detect tcp ()
{
# Netbus fingerprint funnet i TCP-payload vha. pakkesniffer.
$client_fingerprint=588;                    # Den hexadesimale verdien 24C.
$server_fingerprint=515;                    # Den hexadesimale verdien 203.

# Trekker ut utvalgte byte fra payload i en TCP-pakke.
$temp_payload1=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,23)+byte(ip.blob,24)+byte(ip.blob,2
5)+byte(ip.blob,28)+byte(ip.blob,29)+byte(ip.blob,30)+byte(ip.blob,31)+byte(ip.blob,32)+byte(ip.blo
b,33)+byte(ip.blob,34)+byte(ip.blob,35)+byte(ip.blob,36);

$temp_payload2=byte(ip.blob,20)+byte(ip.blob,21)+byte(ip.blob,22)+byte(ip.blob,23)+byte(ip.blob,2
4)+byte(ip.blob,25)+byte(ip.blob,28)+byte(ip.blob,29)+byte(ip.blob,30)+byte(ip.blob,31)+byte(ip.blo
b,32)+byte(ip.blob,33)+byte(ip.blob,34)+byte(ip.blob,35);

if ($temp_payload1==$client_fingerprint)
        {
        $klient_request=1;
        $klient_ip=ip.src;
        }

if ($temp_payload2==$server_fingerprint)
        {
        $server_response=1;
        $server_ip=ip.src;
        }

if (($klient_request&&$server_response)!=0)
        {
        alert(_:NETBUS,_:NETBUS_2_DETECTED,$klient_ip,$server_ip);
        record system.time,$klient_ip,$server_ip,tcp.connDport to the_recorder_netbus_2;
        $klient_request=0;
        $server_response=0;
        }
}

# Dette skaper den recorderen som skal brukes.
the_recorder_netbus_2=recorder("bin/histogram %c","netbus_2_schema");
```

Vedlegg 17: netbus_2.cfg - Konfigurasjonsfil for Netbus 2.0 filter

```
#
#

enabled=true
gui=histogram

#
num_columns=3

#
column_1_type=p_src_ip
column_2_type=p_dst_ip
column_3_type=p_dst_port

#
column_1_label=Netbus klient addresse
column_2_label=Netbus server addresse
column_3_label=Netbus server port

#
count_label=Netbus oppkoblingsforsok

#
rollover=300

#
suppress_all_time=false

#
sync_time=10

#
modified=true
origin=pkjohnsen@hotmail.com
title=Netbus 2.0 Pro detection module

#
doalerts=true
alert_text=Nytt oppkoblingsforsok : $(1)

#
new_cell_alerts=false

#
cfversion=2
rollover_size=YES
rollover_size_val=1024000
rollover_time=YES
rollover_time_val=300000
#
archive_path=data/%p/%b/%y/%m%d/

data_label=
```

Vedlegg 18: Hoppa Analyzer log av Netbus 1.2 oppkobling

```
*******************************************************************************
*   Hoppa Analyser Output File                                                *
*   Created 04-01-1999 at 15:34                                               *
*******************************************************************************

--- Packet received: 15:34:38.61 --- Length: 0058 --- Assigned number: 00000 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       612Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        0491h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1399d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010D8Ch  TCP 32 bits ack number:        00000000h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 02h SYN
TCP 16 bits window size:         2000h
TCP 16 bits checksum:            9FEDh  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 15:34:38.61 --- Length: 0060 --- Assigned number: 00001 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       4000h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        25D0h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:       1399d
TCP 32 bits sequence number: 00010E3Dh  TCP 32 bits ack number:        00010D8Dh
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 12h ACK SYN
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            8F66h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 15:34:38.61 --- Length: 0054 --- Assigned number: 00002 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       622Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        0395h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1399d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010D8Dh  TCP 32 bits ack number:        00010E3Eh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            A723h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------

--- Packet received: 15:34:38.65 --- Length: 0061 --- Assigned number: 00003 ---
```

```
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0047d
IP 16 bits Identification:       4100h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        24CDh
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:      1399d
TCP 32 bits sequence number: 00010E3Eh  TCP 32 bits ack number:        00010D8Dh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            61F9h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 0D                                     NetBus.
-------------------------------------------------------------------------------

--- Packet received: 15:34:38.81 --- Length: 0054 --- Assigned number: 00004 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       632Fh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                       80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        0295h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1399d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010D8Dh  TCP 32 bits ack number:        00010E45h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2231h
TCP 16 bits checksum:            A723h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------

*** End of file ***************************************************************
```

Vedlegg 19: Hoppa Analyzer log av Netbus 1.53 oppkobling

```
*******************************************************************************
*   Hoppa Analyser Output File                                                *
*   Created 04-01-1999 at 14:53                                               *
*******************************************************************************

--- Packet received: 14:53:07.37 --- Length: 0058 --- Assigned number: 00000 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       CB2Ch  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        9A93h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1373d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010CDFh  TCP 32 bits ack number:        00000000h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 02h SYN
TCP 16 bits window size:         2000h
TCP 16 bits checksum:            A0B4h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:53:07.37 --- Length: 0060 --- Assigned number: 00001 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       0C42h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        598Eh
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:      1373d
TCP 32 bits sequence number: 00010E36h  TCP 32 bits ack number:        00010CE0h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 12h ACK SYN
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            9034h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:53:07.37 --- Length: 0054 --- Assigned number: 00002 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       CC2Ch  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        9997h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1373d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010CE0h  TCP 32 bits ack number:        00010E37h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            A7F1h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------
```

```
--- Packet received: 14:53:07.41 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0053d
IP 16 bits Identification:       0D42h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       5885h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:      1373d
TCP 32 bits sequence number: 00010E37h  TCP 32 bits ack number:        00010CE0h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E13Ah  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 20 31 2E 35 33 20 0D                   NetBus 1.53 .
-------------------------------------------------------------------------------

--- Packet received: 14:53:07.52 --- Length: 0054 --- Assigned number: 00004 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0040d
IP 16 bits Identification:       CD2Ch  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       9897h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1373d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010CE0h  TCP 32 bits ack number:        00010E44h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         222Bh
TCP 16 bits checksum:            A7F1h  TCP 16 bits urgent pointer:       0000h
No data
-------------------------------------------------------------------------------

*** End of file ***************************************************************
```

Vedlegg 20: Hoppa Analyzer log av Netbus 1.6 oppkobling

```
*******************************************************************************
*   Hoppa Analyser Output File                                                *
*   Created 04-01-1999 at 14:39                                               *
*******************************************************************************

--- Packet received: 14:39:31.75 --- Length: 0058 --- Assigned number: 00000 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       2F28h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        3698h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1356d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010C5Bh  TCP 32 bits ack number:        00000000h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 02h SYN
TCP 16 bits window size:         2000h
TCP 16 bits checksum:            A149h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:39:31.76 --- Length: 0060 --- Assigned number: 00001 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       333Fh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        3291h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:       1356d
TCP 32 bits sequence number: 00010DCBh  TCP 32 bits ack number:        00010C5Ch
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 12h ACK SYN
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            9134h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:39:31.76 --- Length: 0054 --- Assigned number: 00002 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       3028h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        359Ch
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1356d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010C5Ch  TCP 32 bits ack number:        00010DCCh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            A8F1h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------
```

```
--- Packet received: 14:39:31.77 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0053d
IP 16 bits Identification:       343Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       3188h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:     1356d
TCP 32 bits sequence number: 00010DCCh  TCP 32 bits ack number:       00010C5Ch
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E539h  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 20 31 2E 36 30 20 0D                   NetBus 1.60 .
--------------------------------------------------------------------------------

--- Packet received: 14:39:31.95 --- Length: 0054 --- Assigned number: 00004 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0040d
IP 16 bits Identification:       3128h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       349Ch
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1356d  TCP 16 bits destination port:     12345d
TCP 32 bits sequence number: 00010C5Ch  TCP 32 bits ack number:       00010DD9h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         222Bh
TCP 16 bits checksum:            A8F1h  TCP 16 bits urgent pointer:       0000h
No data
--------------------------------------------------------------------------------

*** End of file ***************************************************************
```

Vedlegg 21: Hoppa Analyzer log av Netbus 1.7 oppkobling

```
*******************************************************************************
*   Hoppa Analyser Output File                                                *
*   Created 04-01-1999 at 14:25                                               *
*******************************************************************************

--- Packet received: 14:24:49.28 --- Length: 0058 --- Assigned number: 00000 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       9827h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        CD98h
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1352d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010C49h  TCP 32 bits ack number:        00000000h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 02h SYN
TCP 16 bits window size:         2000h
TCP 16 bits checksum:            A15Fh  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:24:49.28 --- Length: 0060 --- Assigned number: 00001 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       083Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        5D91h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:       1352d
TCP 32 bits sequence number: 00010DA4h  TCP 32 bits ack number:        00010C4Ah
TCP 4 bits data offset:             6h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 12h ACK SYN
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            9171h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 14:24:49.29 --- Length: 0054 --- Assigned number: 00002 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       9927h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        CC9Ch
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1352d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010C4Ah  TCP 32 bits ack number:        00010DA5h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            A92Eh  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------
```

```
--- Packet received: 14:24:49.31 --- Length: 0067 --- Assigned number: 00003 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0053d
IP 16 bits Identification:       093Fh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       5C88h
IP source address: 128.39.202.85        IP destination address: 128.39.202.87
TCP 16 bits source port:         12345d  TCP 16 bits destination port:      1352d
TCP 32 bits sequence number: 00010DA5h  TCP 32 bits ack number:        00010C4Ah
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            E575h  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
4E 65 74 42 75 73 20 31 2E 37 30 20 0D                   NetBus 1.70 .
-------------------------------------------------------------------------------

--- Packet received: 14:24:49.48 --- Length: 0054 --- Assigned number: 00004 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0040d
IP 16 bits Identification:       9A27h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       CB9Ch
IP source address: 128.39.202.87        IP destination address: 128.39.202.85
TCP 16 bits source port:         1352d  TCP 16 bits destination port:      12345d
TCP 32 bits sequence number: 00010C4Ah  TCP 32 bits ack number:        00010DB2h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         222Bh
TCP 16 bits checksum:            A92Eh  TCP 16 bits urgent pointer:       0000h
No data
-------------------------------------------------------------------------------

*** End of file ***************************************************************
```

Vedlegg 22: Hoppa Analyzer log av Netbus 2.0 Pro oppkobling

```
*******************************************************************************
*   Hoppa Analyser Output File                                                *
*   Created 11-05-1999 at 17:58                                               *
*******************************************************************************

--- Packet received: 17:57:42.54 --- Length: 0058 --- Assigned number: 00006 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       858Dh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        E041h
IP source address: 128.39.202.87        IP destination address: 128.39.202.70
TCP 16 bits source port:         4379d  TCP 16 bits destination port:      20034d
TCP 32 bits sequence number: 0002971Ch  TCP 32 bits ack number:        00000000h
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 02h SYN
TCP 16 bits window size:         2000h
TCP 16 bits checksum:            ECBDh  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 17:57:42.54 --- Length: 0060 --- Assigned number: 00007 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:           0044d
IP 16 bits Identification:       2952h  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        3C8Dh
IP source address: 128.39.202.70        IP destination address: 128.39.202.87
TCP 16 bits source port:         20034d  TCP 16 bits destination port:      4379d
TCP 32 bits sequence number: 000128EFh  TCP 32 bits ack number:        0002971Dh
TCP 4 bits data offset:             6h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 12h ACK SYN
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            C184h  TCP 16 bits urgent pointer:        0000h
HEX data:                                                ASCII data:
02 04 05 B4                                              ....
-------------------------------------------------------------------------------

--- Packet received: 17:57:42.54 --- Length: 0054 --- Assigned number: 00008 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:              5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:           0040d
IP 16 bits Identification:       868Dh  IP 3 bits Flags:                      2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                        80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:        DF45h
IP source address: 128.39.202.87        IP destination address: 128.39.202.70
TCP 16 bits source port:         4379d  TCP 16 bits destination port:      20034d
TCP 32 bits sequence number: 0002971Dh  TCP 32 bits ack number:        000128F0h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                  00h
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            D941h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------
```

```
--- Packet received: 17:57:42.54 --- Length: 0086 --- Assigned number: 00009 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0072d
IP 16 bits Identification:       878Dh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       DE25h
IP source address: 128.39.202.87        IP destination address: 128.39.202.70
TCP 16 bits source port:         4379d  TCP 16 bits destination port:     20034d
TCP 32 bits sequence number: 0002971Dh  TCP 32 bits ack number:       000128F0h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2238h
TCP 16 bits checksum:            068Ah  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
42 4E 20 00 02 00 08 08 05 00 41 0C 69 1F 5D 28 5B BC 62 AE BN .......A.i.]([.b.
64 B7 6E 84 8C BC 68 AF 3E F2 93 E0                      d.n...h.>...
--------------------------------------------------------------------------------

--- Packet received: 17:57:42.56 --- Length: 0042 --- Assigned number: 00010 ---
MAC destination: FF:FF:FF:FF:FF:FF      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0806h
Protocol: ARP
ARP 16 bits hardware type:    0001h  ARP 16 bits protocol type:         0800h
ARP 8 bits length of MAC address:  06h  ARP 8 bits lenght of IP address:     04h
ARP 16 bits operation code:      0001h ARP_REQUEST
ARP MAC source address:          00:E0:2C:02:2C:56
ARP IP source address:           128.39.202.87
ARP MAC destination address:     00:00:00:00:00:00
ARP IP destination address:      128:39:202:1
No data
--------------------------------------------------------------------------------

--- Packet received: 17:57:42.67 --- Length: 0070 --- Assigned number: 00011 ---
MAC destination: 00:E0:2C:02:2C:56      MAC source: 00:10:4B:31:41:6D
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         00h  IP 16 bits Total length:          0056d
IP 16 bits Identification:       2A52h  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       3B81h
IP source address: 128.39.202.70        IP destination address: 128.39.202.87
TCP 16 bits source port:         20034d  TCP 16 bits destination port:      4379d
TCP 32 bits sequence number: 000128F0h  TCP 32 bits ack number:       0002973Dh
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
TCP 6 bits control flags : 18h ACK PUSH
TCP 16 bits window size:         2218h
TCP 16 bits checksum:            69B9h  TCP 16 bits urgent pointer:       0000h
HEX data:                                                ASCII data:
42 4E 10 00 02 00 0C CE 05 00 41 0C 6B 1F 5D 28             BN........A.k.](
--------------------------------------------------------------------------------

--- Packet received: 17:57:42.80 --- Length: 0054 --- Assigned number: 00012 ---
MAC destination: 00:10:4B:31:41:6D      MAC source: 00:E0:2C:02:2C:56
Frametype: Ethernet II, Protocol field: 0800h
Protocol: IP
IP 4 bits IP version:               4h  IP 4 bits Header length:             5h
IP 8 bits Type of service:         10h  IP 16 bits Total length:          0040d
IP 16 bits Identification:       898Dh  IP 3 bits Flags:                     2h
IP 13 bits Fragment Offset:      0000h  IP 8 bits TTL:                      80h
IP 8 bits Protocol type:     TCP = 06h  IP 16 bits Header Checksum:       DC45h
IP source address: 128.39.202.87        IP destination address: 128.39.202.70
TCP 16 bits source port:         4379d  TCP 16 bits destination port:     20034d
TCP 32 bits sequence number: 0002973Dh  TCP 32 bits ack number:       00012900h
TCP 4 bits data offset:             5h  TCP 6 bits reserved                 00h
```

```
TCP 6 bits control flags : 10h ACK
TCP 16 bits window size:         2228h
TCP 16 bits checksum:            D921h  TCP 16 bits urgent pointer:        0000h
No data
-------------------------------------------------------------------------------
```

*** End of file **********