Nettverksover-våkning med Snort

Nettverksovervåkning med Snort

av

Geir Solli og Dag Atle Isene

HiVE Høgskolen i Vestfold 2003

Sammendrag

I dette prosjektet har vi tatt for oss Snort, som er en programvarepakke for overvåkning av nettverkstrafikk.

I vårt opprinnelige prosjektmål, hadde vi definert følgende målsetning:

Oppgaven går ut på å gjøre seg kjent med dataprogrammet SNORT og implementere dette programmet sammen med en overvåkningsfunksjon og filtrering toveis.
Overvåknings/filtreringssystemet skal bygges og demonstreres på en vertsmaskin med Linux og som innehar ruter funksjonalitet mellom 2 forskjellige LAN.
Oppgaven går ut på å kvalifisere og demonstrere denne ruter funksjonaliteten med en vertsmaskin i HVE's datalab (C258) mellom 2 LAN."

Det viste seg at Snort ikke egner seg til denne type oppgaver, og vi begrenset vårt prosjektmål til å sette opp Snort som et rent nettverksovervåknings system. Vi har også implementerte flere grensesnitt, som gjør det mulig og administrere og konfigurere Snort via web.

Resultatet av vårt arbeid er et komplett open- source Nettverks Overvåknings System, som er et godt alternativ til, ofte dyre, tilsvarende kommersielle produkter.

Forord

Som en del av fagplanen ved ingeniørutdanningen ved Høgskolen i Vestfold inngår et hovedprosjekt i den avsluttende fasen av utdanningen. Her skal studentene planlegge, gjennomføre og dokumentere tverrfaglig prosjekter. Prosjektene bør fortrinnsvis utvikles i nært samarbeide med næringen/ industrien, og være egnet til å gi faglig fordypning. Arbeidet bør inneholde en stor grad av «nyskapning» eller springe ut fra en ide eller skisse framsatt av prosjektveileder eller næringens kontaktperson.

Vårt arbeid har vært utført i gruppe med to studenter under veiledning. Før gjennomføringen startet ble prosjektmålsetningen med rammebetingelser godkjent av avdelingen. Forslagsstiller til vårt prosjekt var veileder Hans Jørgen Alker, i samarbeid med IT-avdelingen på Høgskolen i Vestfold.

* Frist for innlevering av prosjektsøknad var 03.12.02, og avdelingens godkjennelse av prosjektet hadde tidsfrist til 08.01.03.
* Innlevering av prosjektplan hadde en frist til 22.01.03, og presentasjon av prosjektet på web hadde frist til 01.02.03.
* Innlevering av ferdig prosjektrapport hadde frist til 06.06.03, men på grunn av dødsfall i familien fikk vi utsatt vår innleveringsfrist til 20.06.03.
* Utstilling av vårt arbeid er også en del av det obligatoriske arbeidet. Dette foregikk på Høgskolen i Vestfold, avdeling Borre, 11. og 12.06.03, HVExpo.

Vi vil takke vår veileder Hans Jørgen Alker og studentassistent Magnus Olstad Hansen for god hjelp gjennom hele prosjektet. Vi vil også rette en stor takk til teamleder for data, Helge Herheim for sin fleksibilitet og forståelse.

Geir Solli, Borre 20.06.03 Dag Atle Isene, Borre 20.06.03

--- ---

Innholdsfortegnelse

Innledning

Vi har i prosjektet gjort oss kjent med Snort, i Linux miljø, og implementert dette programmet sammen med en overvåkningsfunksjon. Vi har benyttet Snort som er et Network Intrusion Detection System, og implementert i dette en serverfunksjonalitet for pakkelogging og alarmering. Vi har også benyttet eksisterende web grensesnitt for overvåkning og administrasjon av systemet. I oppsettet av Snort har vi tilpasset regler for detektering av uønsket trafikk, samt konfigurert Snort til å fungere i HVE's data- lab (C258).

Videre var det meningen å lage policy regler som begrenser trafikk mellom inter – LAN. Da Snort viser seg ikke å være egnet til et slikt formål, lot prosjektet seg vanskelig gjennomføre slik det opprinnelig var tenkt.

Vår erfaring fra Linux begrenser seg til et enkelt kurs i nettverk, hvor vi benyttet Linux til enkle kommandolinje baserte nettverksoppgaver. Vi skulle gjerne hatt bedre bakgrunn fra dette operativsystemet, noe som står høyt på ønskelisten til mange studenter.

Hva er Snort?

Snort er en gratis programvare fortrinnsvis for Linux, men kan også brukes i andre operativ systemer. Snort er et lettvekts open- source Network Intrusion Detection System, heretter kalt NIDS, som kan utføre realtime trafikkanalyser og pakke logging på IP nettverk. Snort kan utføre protokollanalyser, innholds- søk/treff og kan brukes for å oppdage en mengde angrep og prober, slik som buffer overflows, port skanninger (også stealth), CGI angrep, DOS angrep, OS fingerprinting (begrepene er forklart i kapittel 8).

Snort bruker et fleksibelt regel språk for å beskrive trafikk som skal plukkes opp eller passere, men også en deteksjons motor som tar i bruk en modulær plug- in arkitektur. Snort har også en realtime alarmerings evne som innlemmer alarmerings mekanismer for syslog, en bruker spesifisert fil, en UNIX sokkel, eller WinPopop meldinger til Windows klienter.

Snort kan brukes på tre forsjellige måter. Den kan brukes som en enkel pakke sniffer, en pakke logger (nyttig til nettverks feilsøking etc.), eller som en komplett NIDS.

Kapittel 1 Snort - den planleggende fasen

Før vi startet installasjon av Snort måtte vi bestemme oss for hvordan vi ville ha den begrepsmessige topologien.

Vi måtte også planlegge hvordan vi skulle sette opp Snort, og hva som trengtes for å bli kjent med programvaren.

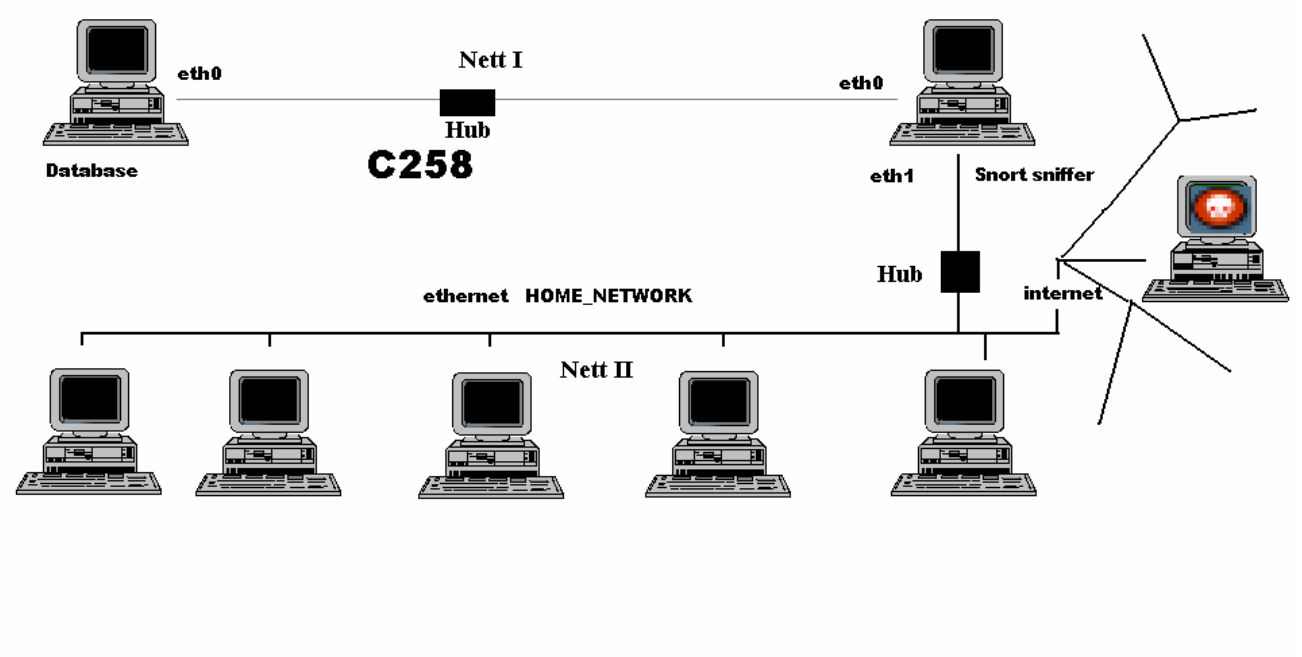
1.1 Begrepsmessig topologi

For på en enkel måte kunne bli kjent med Snort, dens funksjoner og muligheter, satt vi Snort opp som en NIDS. Dette innebar at vi ville kun ha én maskin til overvåkning på ett enkelt nett- segment. Vi avgjorde også i den planleggende fasen å implementere en database, som vi kunne logge trafikk og alarmer på.

Vi tenkte oss følgende oppkobling: Maskin A som Database, heretter kalt A, Maskin B som Snort ”sniffer”, heretter kalt B, med intern trafikken mellom disse på et ”lukket” nett, heretter kalt Nett I.(Se Figur 1.1.1) B, Snort ”sniffer”, måtte ha to nettverkskort, da den også skulle være koblet til nettet den skulle overvåke, heretter kalt Nett II

Da nettverksovervåkning ikke lar seg gjøre rett på en svitsjet port, slik som i vår data- lab C258, bestemte vi oss for å koble nettet som skulle overvåkes, et utvalg av maskiner på data-laben, sammen med en sammenkobler, heretter kalt hub. Trafikken som skulle gå mellom A og B, måtte også ha en hub, da disse ikke kunne kobles direkte til hverandre uten å bruke en krysset nettverks- kabel.

Vi bestemte oss også for å kjøre HTTPS (HyperText Transfer Protocol Secure), som gir oss en sikker overføring på nett I.



Figur 1.1.1 Begrepsmessig IDS topologi.

1.2 Valg av programvare

Før vi startet installasjon av programvare, måtte vi kartlegge hva slags versjonstyper av de forsjellige programvare- pakkene som fungerte best i sammen. Her fikk vi god hjelp av dokumentasjonen som ligger tilgjengelig på http://www.snort.org.

* Snort i seg selv har kommet til en versjon 2.0, som er den beste og mest stabile versjonen. Snort versjon 2.0, fungerer best med operativ systemet Redhat 9.0, og er også best dokumentert med et slikt oppsett.

* Under vår kartlegging av programvarepakker, bestemte vi oss for å installere en MySQL database (Avsnitt 7.2) for logger og alarmer, samt en web basert applikasjon som er spesielt utviklet for overvåkning og administrasjon av logger og alarmer, generert av NIDS eller en brannmur. Denne programvaren heter ACID (Analysis Console for Intrusion Detection), og er også en gratis open- source applikasjon for Linux.

* ACID trengte også en server som den skulle gå på, og vi valgte og legge denne på en Apache server. (Avsnitt 7.1)

* Vi fant ut at det finnes et webgrensesnitt for administrasjon av Unix baserte systemer som heter Webmin. Denne har også en modul for administrasjon av Snort og dens konfigurasjon. Praktisk talt kan ”alle” funksjoner i Snort styres herfra. (Avsnitt 7.4) Vi bestemte oss også for å implementere denne.

1.3 Snort 2.0 og Redhat 9.0

Under installasjon av de valgte programvare- versjonen fikk vi tidlig problemer med oppsettet. Vi installerte Redhat 9.0 på maskin A og B, og startet implementasjonen av annen programvare. Vi erfarte tidlig det at ting ikke var enkelt med disse versjonene. Under installasjon av Net_SSLeay, som kreves for å kjøre SSL (Secure Sockets Layer) kobling mot Webmin, oppsto det så store versjonskonflikter at vi måtte stoppe installasjonen.

Det viste seg at det ikke var mulig og finne en versjon av Net_SSLeay som er tilpasset Redhat 9.0, og vi slet lenge med dette problemet, og hadde flere til og hjelpe oss, deriblant vår gode hjelper, studentassistent Magnus Hansen.

Vi bestemte oss, etter mye prøving og feiling, å starte fra bunnen av med Snort 1.8.7 og Redhat 7.3.

1.4 Oppsummering av Programvarepakker

Programvare på A (Database):

- Apache Web Server for ACID (Avsnitt 7.1)
- MySQL server (Avsnitt 7.2)
- ACID (Avsnitt 7.3)

Programvare på B (Snort sniffer):

- MySQL klient
- Snort
- Webmin (Avsnitt 7.4)
- SSL (Avsnitt 7.6)

Vi har sett at det fort kan oppstå problemer. Det har vært svært tidkrevende å sette seg inn i NIDS topologien, og vi har brukt en ”Learning by Doing” teknikk for å komme videre i arbeidet.

Kapittel 2 Snort Installasjon

I dette kapittelet blir installasjonsprosessen kun fortalt på en enkel måte for å gi innblikk i hvordan vi har satt opp Snort.

Installasjonen av Snort som en komplett NIDS, er grundig beskrevet i ”Installasjonsmanual for Snort”, vedlegg 1.

2.1 Oppsett av Snort database

2.1.1 Redhat

Maskinen ble satt fra bunnen med et enkelt oppsett av Redhat Linux 7.3. Maskinen skulle inneholde MySQL database, Apache server og Acid konsollen.

2.1.2 Apache

Vi startet med å hente ned Apache serveren og installerte denne. Det neste som skulle gjøres var å installere og konfigurer MySQL databasen.

2.1.3 MySQL

MySQL ble satt opp med en lokal bruker, og det ble opprettet en database kalt snort, satt opp med bruker rettigheter for maskin B.

2.1.4 Acid

Vi erfarte ganske snart at Acid krever installasjon av PHP og en supplerende MySQL modul. Disse ble lastet ned og installert, og vi kunne fortsette med installasjonen av Acid. Dette gikk forholdsvis smertefritt, men vi har erfart at for å få Acid opp å gå, så måtte vi ha med Adodb, et databasebibliotek for PHP, samt at vi også fikk med oss et ANCI C bibliotek kalt GD, for dynamisk kreasjon av bilder.

2.2 Oppsett av Snort sensor

2.2.1 Redhat

Som på maskin A, ble også denne satt opp med Redhat Linux 7.3. Denne skulle ha Snort og Webmin installert.

2.2.2 Snort

For å få Snort til å gå slik vi hadde tenkt, med overføring av alarmer til database, trengtes en MySQL klient. Snort ble installert og konfigurert på enklest mulig måte, med et automatisk oppstarts skript, for en enkelt test for å se at den rapporterer til Acid. Dette fungerte uten de helt store problemene.

2.2.3 Webmin

Som tidligere nevnt, kreves Net_SSLeay for å kjøre en SSL kobling mot Webmin. Installasjonen av denne gikk greit. Webmin, med tilhørende Snort IDS Admin modul, ble også installert og sjekket.

Snort fungerte nå som vi ville og den manglet kun konfigurering og justering av regel filer.

Kapittel 3 Konfigurering av Snort

3.1 snort.conf filen

Etter at Snort er installert, må ”/etc/snort/snort.conf” (se vedlegg 2) filen konfigureres etter behov. Martin Roesch har laget ”Snort User Manual”, som vi har oversatt, ”Brukermanual For Snort” (vedlegg 3). Denne beskriver hva slags muligheter som finnes i Snort, og gav oss en god pekepinn på de valgene vi ville ha med i vårt oppsett.

3.1.1 Snort Variabler

Først må forskjellige variabler, som HOME_NET, EXTERNAL_NET og DNS_SERVERS defineres, slik at de passer med nettverks- beskrivelsen. Her er det svært viktig å definere variablene riktig, eller så vil ikke Snort fungere på riktig måte. Vi har definert HOME_NET variabelen til å inneholde en komma separert liste over de maskinene som inngikk i vårt nettverksoppsett, beskrevet i figur 3.1.1.1. EXTERNAL_NET variabelen er definert til ”alle andre adresser”.

Når man bruker Snort i et sammensatt miljø, med en sensor og mange forsjellige grensesnitt som skal overvåkes, kan det bli vanskelig å definere HOME_NET og EXTERNAL_NET variablene. Da kan begge variablene settes til ”any”. Du mister da en ”tidlig filtrering”, ved å ikke definere alle nettverks områdene (Range) i det store intern- nettet, men ytelsestapet i Snort blir ikke så stort, ved at Snort slipper å sjekke gjennom en stor liste med adresser for hver pakke.

For å slippe “falske” meldinger om portscans, definer DNS_SERVERS variabelen til å inneholde IP adresser på dns- servere, samt andre nettverksnoder, som kan trigge Snorts portscan modul.

Du kan også definere dine egne variabler her som du kan refererer til i dine egne regler. Dette er spesielt nyttig om du skal bruke ”pass rules” som er tilpasset ditt nettverks oppsett.

Definer alle andre variabler til passende verdier, eller som i figur 3.1.1.1

```
var HOME_NET [10.100.101.30,10.100.101.27,128.39.114.141,128.39.114.148]
var EXTERNAL_NET !$HOME_NET
# DNS_SERVERS holds the addresses of "noisy" computers like DNS or NWM
# to be ignored from portscans
var DNS_SERVERS [1.1.1.1/32,2.2.2.2/32]
var SMTP_SERVERS $HOME_NET
```

Figur 3.1.1.1 Nettverksvariabler

3.1.2 Snort Preprosessorer

Det neste som må gjøres er å sette opp preprosessorer (se avsnitt 3.5). Her er det viktig å velge de riktige preprosessorene. Jo flere preprosessorer du har, jo flere triggede alarmer får du, og det er viktig å huske på at dette går på bekostning av ytelse.

Her er det også viktig å sjekke med ”Brukermanual For Snort”, vedlegg 3, fordi noen av de tilgjengelige preprosessorene er man fra rådet fra å bruke. Om man bestemmer seg for å bruke noen av disse, er det smart å sjekke om det finnes nye oppdateringer.

Preprosessorene *minifrag* og *stream* er erstattet med *stream4* og *defrag* er erstattet av *frag2*.

frag2 er den nye IP defragmenterings prosessoren som ble introdusert med Snort versjon 1.8 og er mer effektiv med hensyn til minnebruk, enn den gamle *defrag/minifrag*

Fra ”Brukermanual For Snort”: *Stream4 modulen gir TCP stream sammensettingen og kraftig analysemuligheter for Snort. Robuste stream sammensettingsmuligheter tillater Snort å ignorere ”statusløse” angrep slik som stick og snot produserer. Stream4 gir storbrukere mulighet til å spore mer enn 256 samtidige TCP streams. Stream4 skal være i stand til å håndtere 64,000 samtidige TCP koblinger.*

stream4 modulen består av to preprosessorer, *stream4* og *stream4_reassemble*, som begge må inkluderes.

Det finnes mange forsjellige valgmuligheter for begge preprosessorene, men vi bruker kun, for *stream4 - detect_scans*, for å gi alarmer på portskanning, og *detect_state_problems* for å bli informert når det oppstår handlinger som ubestemte RST pakker, data på SYN pakker og når det oppstår ubestemte sekvensnummer på pakker.

Med *stream4_reassemble* kan vi bruke *port all* valget som gjør at under tilbakesettingen (reassemble) av pakker sjekkes alle porter, istedenfor bare de forhåndsdefinerte. Dette er vel en slags ”føre var” teknikk, som faktisk virker negativt inn på CPU bruken på Snort sensoren, men det anbefales å bruke denne innstillingen.

To andre preprosessorer vi bruker er *portscan* og *portscan-ingnorehosts,* som fanger opp portskanning, og bestemmer hvilke verter som skal ignoreres med hensyn til portskanning.

portscan definerer vi til å se etter alle nettverk som bruker formen 0.0.0.0/0. Vi definerer også antallet port- numre som kan nåes, samt definisjon av deteksjonsperioden, i sekunder. I tillegg legger vi til den komplette stien til portskann- loggfilen.

Med *protscan-ignorehosts* blir vi kvitt noen av de merkelige alarmene fra verter som ”snakker for mye”, slik som dns servere.

Noen av preprosessorene som ikke er nevnt i ”Brukermanual For Snort”, men som vi skal bruke, er *unidecode,* som er en erstatning for *http_decode* og normaliserer http og UNICODE angrep, *rpc_decode* for å normalisere rpc trafikk på en gitt port, *bo* for å sjekke for Back Orifice trafikk og *telnet_decode* for å normalisere telnet strenger.

Her er preprosessor delen av */etc/snort/snort.conf*:

```
preprocessor frag2
preprocessor stream4: detect_scans detect_state_problems
preprocessor stream4_reassemble: ports all
preprocessor unidecode: 80 8080
preprocessor rpc_decode: 111
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor portscan: 0.0.0.0/0 6 3 /var/log/snort/portscan.log
preprocessor portscan-ignorehosts: $DNS_SERVERS
```

Figur 3.1.2.1 Preprosessorer

3.1.3 Snort Output Moduler

Den neste delen av konfigurasjonsfilen er konfigureringen av output modulene, som vi bruker for å sende alarmer til databasen.

Som tidligere nevnt har vi tatt i bruk ACID, og må derfor sette opp Snort til å logge til en database. Database output modulen krever følgende parametere:

log | alert

Logg til alarm område. Det er også mulig å bruke logg området (log | log), men for å få med portscan alarmer inn i databasen, må vi bruke alert.

mysql|postgrsql|odbc|oracle|mssql

Type database.

user=<username>

Her defineres brukernavnet for databasen.

password=<password>

Passordet for den oppgitte brukeren.

dbname=<database name>

Navnet på databasen som det skal logges inn på.

host=<hostname>

Her definerer man vertsnavnet på maskinen som databasen ligger på.

sensor_name=<sensor name>

Her kan man legge inn navnet på sensoren, hvis man skal skille mellom forsjellige sensorer, hvis man har flere som logger til én database.

Figur 3.1.3.1 Snort Output Moduler

Her er output modul delen av */etc/snort/snort.conf:*

```
output alert_syslog: LOG_AUTH LOG_ALERT LOG_PID

output database: alert, mysql, user=snort password=davidoffen dbname=snort
host=10.100.101.27 sensor_name=sensor1
```

Figur 3.1.3.2 Output modulen

Hvis man bruker mer enn én fysisk Snort sensor, og man vil logge til en database, anbefales det bruk av en sentralisert database, på ulike maskiner. Du kan da samsvare alarmdata, med én enkel konsoll, og få bedre oversikt for å oppdage angrep.

3.2 Snort regel filer

Reglene er en virtuell del av Snort. Reglene er delt opp i forsjellige kategorier, og de forsjellige kategoriene kommer sammen med Snort installasjonen. Reglene er å finne i /etc/snort, med filetternavnet *.rules. Formatet på reglene har i versjon 1.8.* blitt forandret for å gjenspeile klassifikasjonstypen. I tillegg kan det defineres prioritet på klassetypene.

Konfigurasjonen av kategorierer blir gjort i /etc/snort/classification.config. Normalt trenger man ikke gjøre noe med denne, siden den er forhånds konfigurert sammen med Snort reglene.

/etc/snort/classification.config vises i figur 3.2.1

```
#
       # config classification:shortname,short description,priority
       #
       #config classification: not-suspicious,Not Suspicious Traffic,0
       config classification: unknown,Unknown Traffic,1
       config classification: bad-unknown,Potentially Bad Traffic, 2
       config classification: attempted-recon,Attempted Information Leak,3
       config classification: successful-recon-limited,Information Leak,4
       config classification: successful-recon-largescale,Large Scale
Information Leak,5
       config classification: attempted-dos,Attempted Denial of Service,6
       config classification: successful-dos,Denial of Service,7
       config classification: attempted-user,Attempted User Privilege
Gain,8
       config classification: unsuccessful-user,Unsuccessful User
Privilege Gain,7
       config classification: successful-user,Successful User Privilege
Gain,9
       config classification: attempted-admin,Attempted Administrator
Privilege Gain,10
       config classification: successful-admin,Successful Administrator
Privilege Gain,11

       # added from vision18.conf
```

```
       # classification for use with a management interface
       # low risk
       config classification: not-suspicious,policy traffic that is not
suspicious,0
       config classification: suspicious,suspicious miscellaneous
traffic,1
       config classification: info-failed,failed information gathering
attempt,2
       config classification: relay-failed,failed relay attempt,3
       config classification: data-failed,failed data integrity attempt,4
       config classification: system-failed,failed system integrity
attempt,5
       config classification: client-failed,failed client integrity
attempt,6
       # med risk
       config classification: denialofservice,denial of service,7
       config classification: info-attempt,information gathering attempt,8
       config classification: relay-attempt,relay attempt,9
       config classification: data-attempt,data integrity attempt,10
       config classification: system-attempt,system integrity attempt,11
       config classification: client-attempt,client integrity attempt,12
       config classification: data-or-info-attempt,data integrity or
information gathering attempt,13
       config classification: system-or-info-attempt,system integrity or
information gathering attempt,14
       config classification: relay-or-info-attempt,relay of information
gathering attempt,15
       # high risk
       config classification: info-success,successful information
gathering attempt,16
       config classification: relay-success,successful relay attempt,17
       config classification: data-success,successful data integrity
attempt,18
       config classification: system-success,successful system integrity
attempt,19
       config classification: client-success,successful client integrity
attempt,20
```

Figur 3.2.1 Klassifisering

Klassifikasjon og regel filene er inkludert i */etc/snort/snort.conf* filen.. I figur 3.2.2 vises det hvordan regelfilene blir inkludert i /etc/snort/snort.conf.

```
# Include classification & priority settings
       include /etc/snort/classification.config

       include /exploit.rules
       include /scan.rules
       include /finger.rules
       include /ftp.rules
       include /telnet.rules
       include /smtp.rules
       include /rpc.rules
```

```
include /rservices.rules
include /backdoor.rules
include /dos.rules
include /ddos.rules
include /dns.rules
include /netbios.rules
include /web-cgi.rules
include /web-coldfusion.rules
include /web-frontpage.rules
include /web-iis.rules
include /web-misc.rules
include /sql.rules
include /x11.rules
include /icmp.rules
include /shellcode.rules
include /misc.rules
include /policy.rules
include /info.rules
#include /icmp-info.rules
include /virus.rules
include /local.rules
```

Figur 3.2.2 Inkluderte regel filer

Når du er ferdig med å sette opp */etc/snort/snort.conf* filen, skal Snort startes ved å kalle opp */etc/rc.d/init.d/snortd start,* og eventuelle feil som dukker opp i filen */var/log/messages*, må rettes opp. Du kan også sjekke status på Snort ved å skrive */etc/snort/snord status.*

3.3 /etc/rc.d/init.d/snortd

I */etc/rc.d/init.d/snortd* bør du sjekke, i det minste, linjen som definerer hvilket nettverkskort som skal brukes som ”sniffer”. Bytt ut INTERFACE=”eth0” med det nettverkskortet du bruker. Dette kan være et annet nettverkskort, ethernet (ethX),pppx eller ipppx. Denne filen inneholder det automatiske oppstartsskriptet for Snort.

3.4 /etc/snort/snort-check

Dette skriptet blir brukt for å lage winpopups via en smbclient, eller for å sende mail til en gitt e- post adresse. Dette har vi ikke testet, men har tatt det med, for eventuelt videre utvikling.

Her er */etc/snort/snort-check* filen.

```
#!/bin/sh
```

```
# Script to be run from within swatch to send alerts in multiple formats
# inspired from script on www.snort.org by Bill Richardson
# extended to read a file called "hosts" with names of
# workstation to send a winpopup, syntax is the same as with snortd option
-M
# Poppi, 02.05.2001

# Prerequisites:
# Samba set up correctly
# Change the following variables according to your system (for RedHat 7.x
user it should be ok)

# hostfile holds the name of the file containing the workstation for
winpopups
hostfile="/etc/snort/hosts"

# recipientfile holds the addresses of all recipients in a single file,
# seperated by newline
recipientfile="/etc/snort/recipients"

# if a recipient file exists
if [ -s "$recipientfile" ] ; then
  # generate the recipientlist with email adresses.
  for i in `cat $recipientfile` ; do
    recipients="$recipients "$i
  done

  echo "$*" | mail -s "Snort-Alert!!!" "$recipients"
fi

# if a hostfile exists, send winpopups
if [ -s "$hostfile" ] ; then
  for i in `cat $hostfile` ; do
    echo "Snort-Alert! $*" | smbclient -M $i > /dev/null 2>&1
  done
fi
```

Figur 3.4.1 Snort-check

3.4.1 /etc/snort/hosts

I denne filen legges alle navnene på de vertene som skal få Snort meldingen, en per linje.

```
hve1022
hve2232
hve2231
```

Figur 3.4.1.1 Verter

3.4.2 /etc/snort/recipients

I filen /etc/snort/recipients kan du legge inn epost adressene til de som ønsker å motta Snort alarmer, en adresse per linje.

```
bjarne@hive.no
kaare@hive.no
```

Figur 3.4.2.1 Epost adresser

Hvis en av disse filene utelates, vil meldingsfunksjonen kobles ut.

Når alle avsnitt i kapittel 3 er utført, er Snort ferdig konfigurert. Det vil gjenstå justeringer av regler, og eliminering av alarmer som ikke er ønsket rapportert, se kapittel 4, for testing.

Kapittel 4 Testing av Snort

4.1 Enkel testing

Testing av Snort installasjonen kan gjøres veldig enkelt, slik som i eksemplet i avsnitt 4.2 og 4.3. I virkeligheten, i et sammensatt nettverk, vil testing og justering nødvendigvis ta tid. Våre erfaringer, fra den testingen vi har gjort, viser at det vil lønne seg og sette opp Snort til å fungere på en slik måte som man i utgangspunktet har tenkt seg, og bruke tid på justeringer etter hvert.

4.1.1 ”Falske” alarmer

Man må belage seg på å bruke litt tid og analysere alarmer og eliminere disse, ved å redefinere regler, om alarmene er ”falske”. De ”falske” alarmene kan komme fra DNS servere som ikke er tatt med i /etc/snort/snort.conf (se avsnitt 3.1), eller for eksempel fra andre verter som ”bråker”, ved å sende ut ping. Disse falske alarmene kan komme lenge etter at man har justert Snort, ved eksempelvis at en ny DNS server blir lagt til.

4.1.2 E-post/SMS varsling

Etter at man er ferdig med å sette opp og justere Snort, vil det være fornuftig og sette opp snort-check, som er omtalt i avsnitt 3.4, for å varsle alarmer via e-post, slik at man kan oppdage et angrep i en tidlig fase, og gjøre mothandlinger.

Det finnes i dag også løsninger som gjør det enkelt og tilsvarende kostnadsfritt og sende e-post til mobiltelefon, ved hjelp av sms. Dette vil føre til at man har et konstant øye med sikkerheten i nettet.

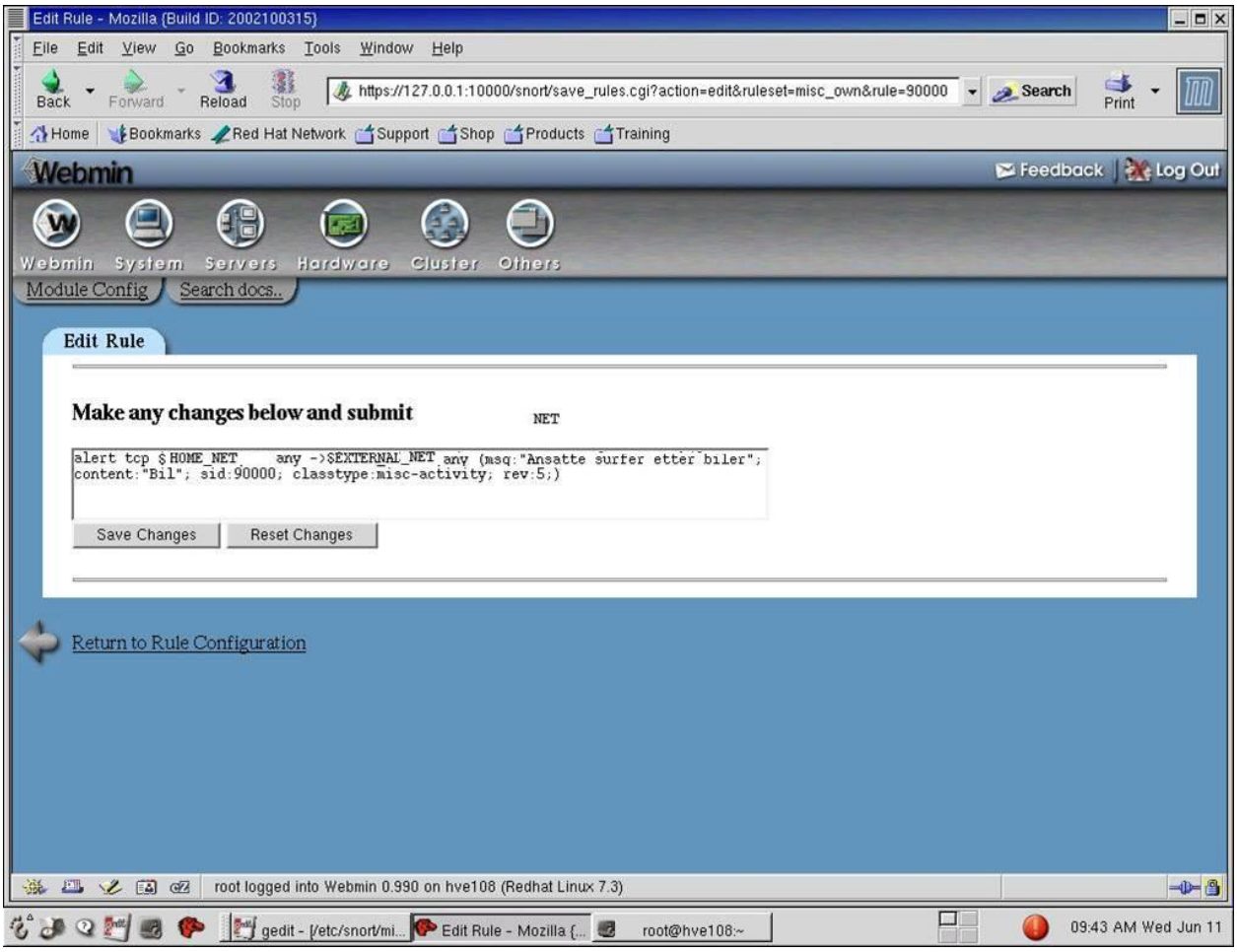
4.1.3 Oppdateringer av regler

Det er ansett som svært viktig å følge med i utviklingen av nye angrep, samt oppdatere regler for å varsle disse. Ondsinnede hackere gjør sitt beste for å ”lure” overvåkningssystemer og ligger alltid ett skritt foran. Det er skrevet mange artikler, og bøker om menneskets onde sinn, men det er noe vi ikke skal ta opp her.

4.2 Test av Snort – Alarm på bakgrunn av en regel

Vi kan illustrere en test av Snort ved å referer til en enkel regel. Denne regelen varsler om en pakke innholder en gitt ASCII streng.

Regelen er definert på følgende måte:

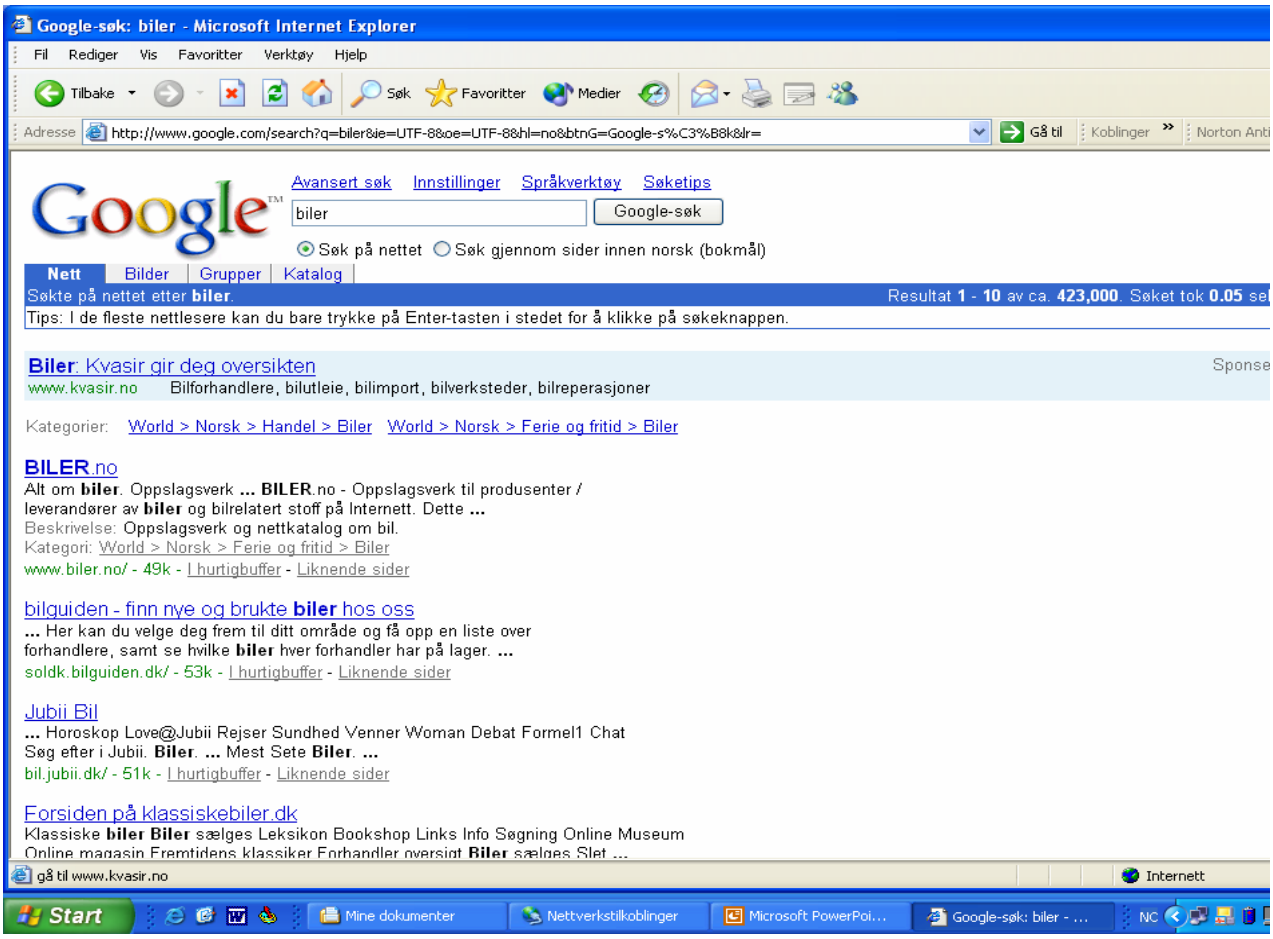


Figur 4.2.1

Forklaring til figur 4.2.1:

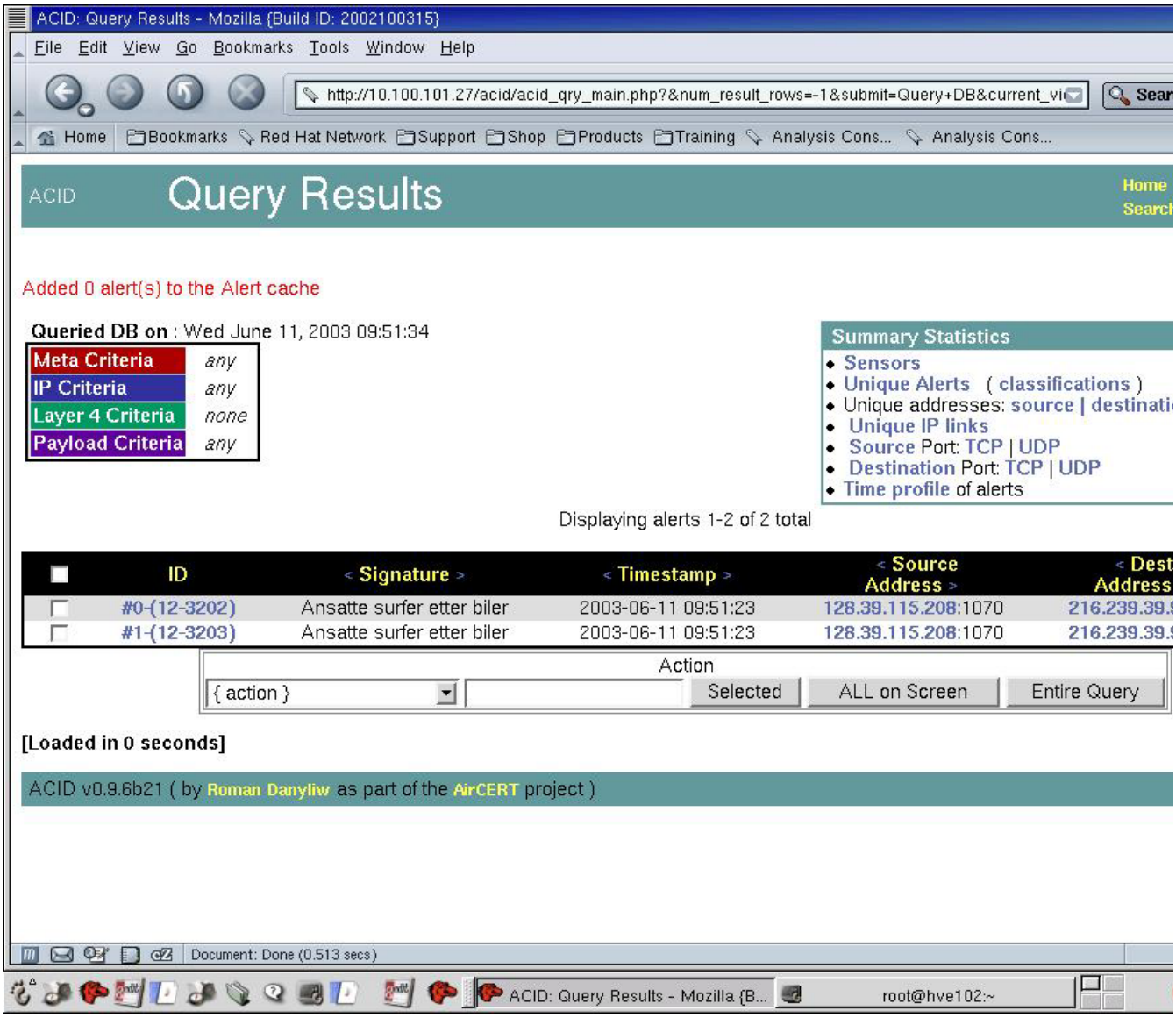
Gi alarm med signatur ”Ansatte surfer etter biler” (msg), hvis det finnes ”Bil” (Content) i innholdet av en pakke, på TCP protokollen fra $HOME_NET på hvilken som helst port (any) -> (til) $EXTERNAL_NET på hvilken som helst port (any).

På en maskin innenfor $HOME_NET definisjonen, blir det åpnet en søkemotor, og søkt etter biler, se figur 4.2.2.



Figur 4.2.2

Når vedkommende trykker på søk knappen, blir det sendt en pakke som inneholder ASCII strengen ”bil”. Denne blir ”plukket opp” av deteksjonsmotoren i Snort, og det blir generert en alarm, med signaturen, ”ansatte surfer etter biler”, Dette vises i web grensesnittet til databasen ACID. Figur 4.2.3.



Figur 4.2.3

I samsvar med regelen kan vi her se at alarm med signaturen ”Ansatte surfer etter biler” dukker opp i databasen. Vi kan også se at det har blitt sendt en pakke med ASCII strengen ”bil” fra 128.39.115.208 port 1070 til 216.239.39.99 port 80.

4.3 Test av Snort – Alarm på bakgrunn av en preprosessor

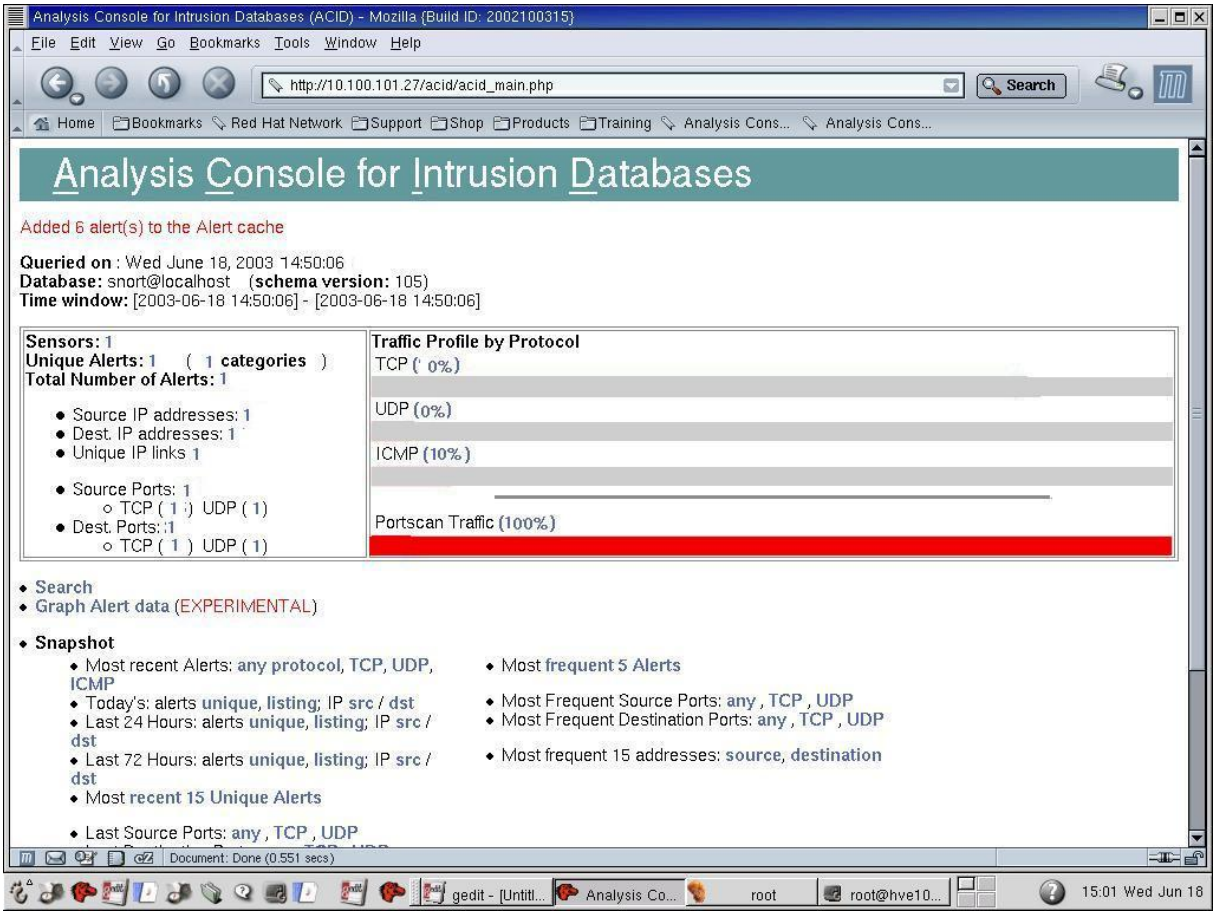
Vi kan illustrere en test av Snort ved å demonstrere en portscan. La oss tenke oss vi sitter på en maskin kalt maskin I. Denne har IP adresse 128.39.114.122. Denne starter en portscan på en av maskinene i nettverket som er definert som $HOME_NET.

Inntrengeren har laget et skript som sjekker alle porter på en gitt maskin Fra kommandolinjen starter han dette skriptet.

```
[root@hve2026 script]# ./ps.sh 128.39.114.142
Tester port 333...
Stopped                 ./ps.sh 128.39.114.142
[root@hve2026 script]#
```

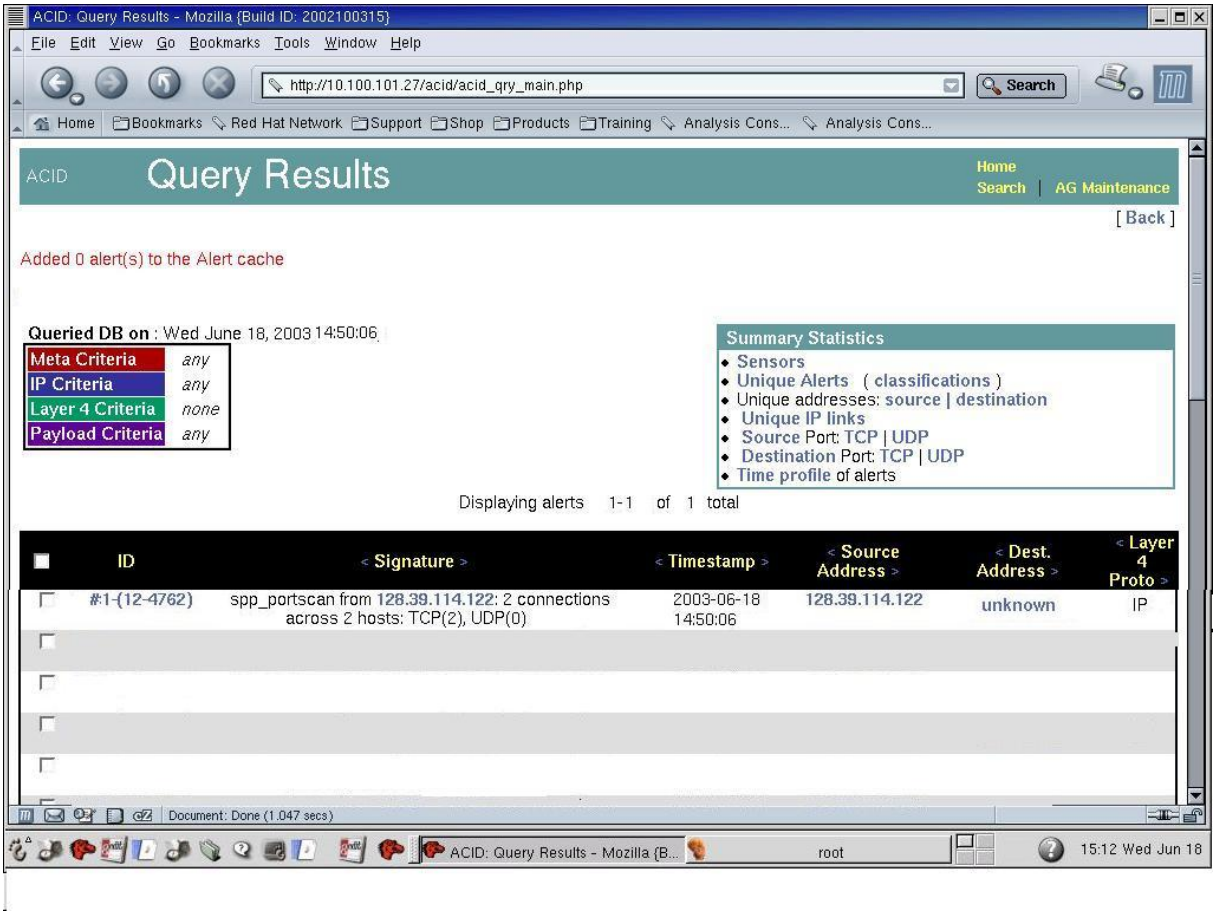
Figur 4.3.1

Snort vil da på grunnlag av preprosessoren portscan, som er definert i avsnitt 3.1.2, gi følgende alarm til databasen. Vi sjekker med ACID grensenittet mot databasen:



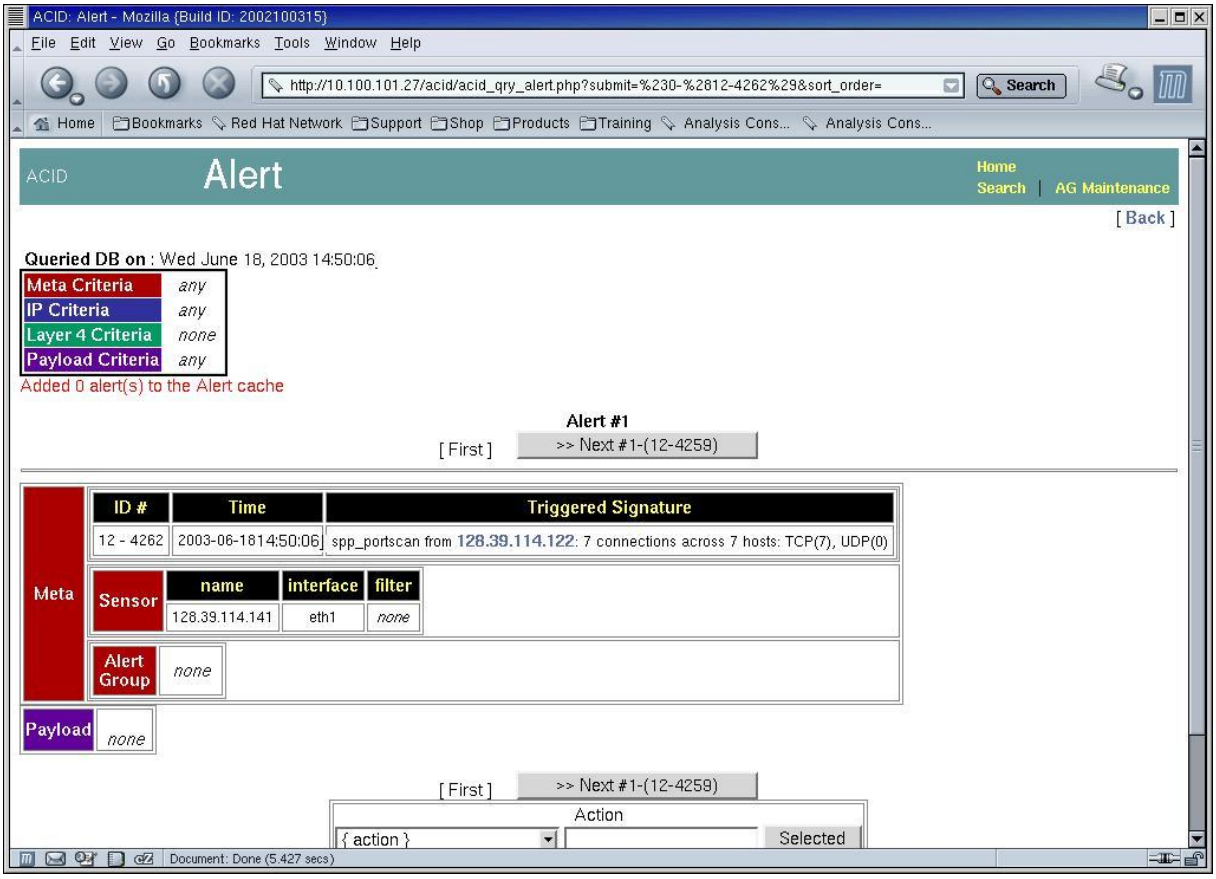
Figur 4.3.2

Her kan vi se at portscan trafikk er meldt med alarmer. Vi går videre inn for å se hvor trafikken kommer fra. Trykker på *Total Numers of Alerts: 1*. Videre får vi følgende bilde:



Figur 4.3.3

Her kan vi se at alarm nummer #1 har en portscan signatur. Vi trykker på alarm id *#1-(12-4762),* og får opp dette bildet:



Figur 4.3.4

Dette bildet gir oss informasjonen vi trenger å vite, her har vi på en maskin innen for $HOME_NET definisjonen fått skannet portene. Portskanningen skjedde fra 128.39.114.122.

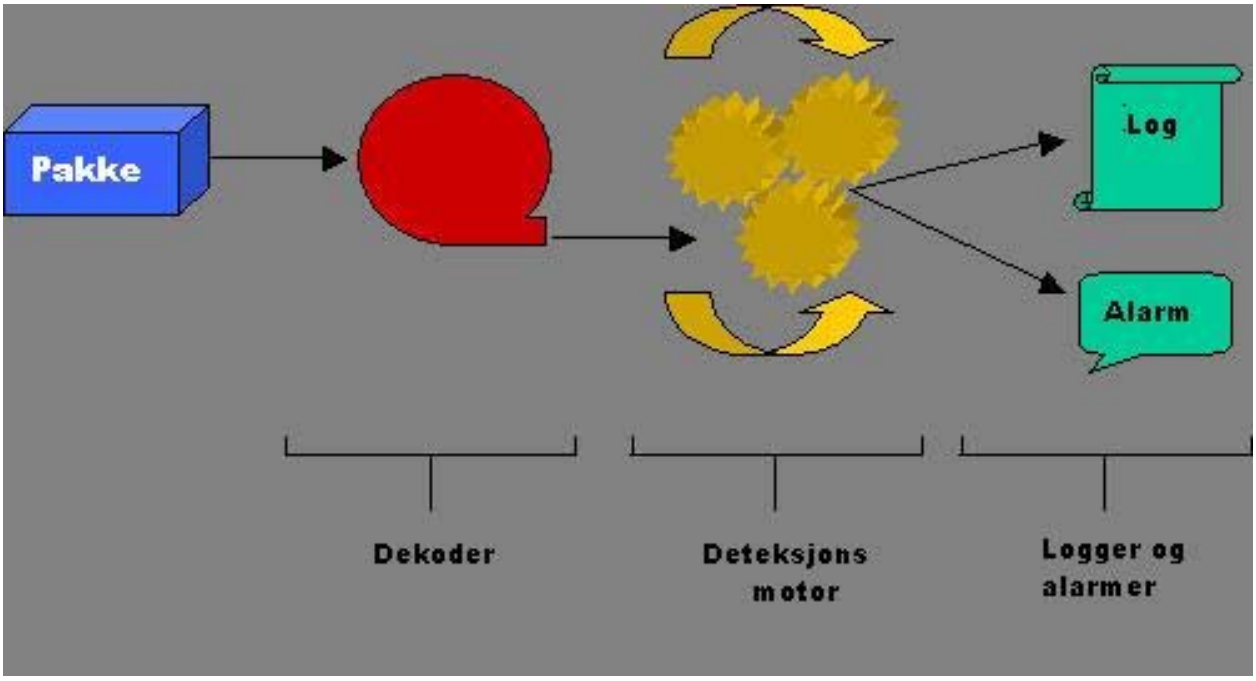
Vi ser at det virket som det skulle.

Kapittel 5 Snorts interne struktur

5.1 Snorts arkitektur

Snorts oppbygning består av tre prinsipielle komponenter. En enkel representasjon av disse komponentene er vist i figur 5.1.1. Disse kan beskrives som følger:

- Pakkedekoder: Snort pakkedekoder støtter Ethernet, SLIP og PPP protokollene. Pakkedekoderen utfører det nødvendige arbeidet med å forbrede data, før de sendes til deteksjonsmotoren.
- Deteksjonsmotoren: denne er Snorts ”hjerte”. Enkelt forklart er denne ansvarlig for å analysere hver pakke, basert på Snort reglene. Deteksjonsmotoren skiller Snort regler inn i ”regelhodet”, og ”regelvalg”. De vanligste variablene slik som kilde og destinasjons IP og portnummer blir identifisert med ”regelhodet”. ”Regelvalget” er definert ved detaljer som for eksempel TCP flagg, ICMP kode typer, spesielt innhold, payload størrelse etc. Deteksjonsmotoren analysere hver pakke basert på disse Snort regel filene. Den første regelen som passer med den dekodede pakken, trigger den handlingen som er spesifisert i regelen. En pakke som ikke passer med noen av Snort reglene, blir kassert. Nøkkelkomponenter i deteksjonsmotoren er plugin modulene slik som portscan modulen. Plugin moduler forsterker funksjonaliteten i Snort ved at man enkelt kan legge til nye analyserings metoder.
- Logger/alarmer: logging og alarmer er to separate underordnede komponenter. Logging tillater deg å logge informasjon samlet av pakkedekoderen, og gjør at den blir presentert på en leselig måte. Du kan konfigurere alarmer til å sendes til syslog, enkel fil, eller database. Du kan også sette alarmfunksjonen ut av drift, for testing og lignende.



Figur 5.1.1 Snorts interne struktur

5.2 Regel- fil formatet

Snort bruker et ”enkelt lettvekts regelbeskrivelses språk” som er fleksibelt og kraftig. Det finnes enkle retningslinjer som man må huske på når man lager Snort regler.

De fleste Snort reglene er skrevet i en enkel linje. Dette var påkrevd i versjonene tidligere enn 1.8. I dagens versjon av Snort kan reglene spenne over flere linjer ved å legge til en ”backslash” \ til enden av linjen.

Snort regler er oppdelt i to logiske seksjoner, regel hodet og regel opsjon. Regel hodet inneholder regelens handling, protokoll, kilde og bestemmelsesstedets IP adresse og nettmaske, og kildes og bestemmelsesstedets port informasjon. Regelens opsjons seksjon inneholder alarm meldinger og informasjon om hvilke deler av pakken som skal inspiseres, for å bestemme om regelens handling skal utføres.

```
alert tcp any any -> 192.168.1.0/24 111 (content:”|00 01 86 a5|”; \
msg:    ”mountd access”;)
```

Figur 5.2.1: En enkel Snort regel

Teksten frem til den første parentesen er reglens hode, og det som står inne i parentesen er reglens opsjon. Ordet før det som står i anførsel er opsjons nøkkelord. Merk deg at regelens opsjons seksjon ikke er spesielt nødvendig for noen regler, de bare brukes for å lage mer bestemte definisjoner av pakker som skal samles, alarmeres eller kastes. Alle elementene som danner en regel må være sanne for at den indikerte regel handlingen skal skje. Når de er satt sammen, kan de sees på som en logisk OG (AND) statement. På samme tid, kan de forsjellige reglene i et Snort regel bibliotek sees på som den danner en stor logisk ELLER (OR) statement.

Alle detaljer for regler i Snort, er beskrevet i ”Brukermanual For Snort”, vedlegg 3, kapittel 2.

5.3 Hvordan fungerer Preprosessorer

Preprosessorer ble introdusert i versjon 1.5 av Snort. De tillater funksjonaliteten i Snort å bli utvidet ved å tillate brukere og programmererer å droppe modulære ”programtillegg” inn i Snort ganske enkelt. Preprosessor kode er kjørt før deteksjonsmotoren blir kalt opp, men etter at pakkene har blitt dekodet. Pakkene kan modifiseres eller analyseres igjennom denne mekanismen.

Preprosessorer er lastet og konfigurert ved å bruke Preprosessor nøkkelordet. Formatet på preprosessordirektivet i Snort regel fil er:

```
preprocessor <name>: <options>

preprocessor minfrag: 128
```

Figur 5.3.1 Preprocessor Directive Format eksempel

Alle detaljer om preprosessorer i Snort, er beskrevet i ”Bruker Manual For Snort”, vedlegg 3, kapittel 2, avsnitt 2.4.

Kapittel 6 Konklusjon

6.1 Opprinnelig prosjektmål

Vår opprinnelige målsetning for dette prosjektet, var å kunne, ved hjelp av Snort, begrense trafikk mellom to Inter- LAN. Det viser seg at Snort ikke egner seg til å gjøre handlinger på bakgrunn av alarmer fra deteksjonsmotoren, utover det å sperre for IP adresser som genererer alarmer. Dette er mulig ved hjelp av forsjellige programpakker som legges ”utenpå” Snort. Grunnen til at vi ikke har nevnt dette tidligere i rapporten, er at vi ønsket en rød tråd gjennom hele beskrivelsen av vårt arbeid.

I arbeidet med å komme frem til en løsning som begrenser trafikk, vil det ikke være hensiktsmessig og sperre IP adresser på denne måten. Muligheten for å oppnå denne effekten, slik vi ser det, ligger i å stenge for pakker som genererer alarmer. Dette må også skje umiddelbart etter at alarmene er generert, for at ønsket effekt skal oppnåes.

6.2 Oppgaven

Det opprinnelige prosjektmålet har også vært vårt mål og ønske gjennom hele prosjektet. Da det viste seg at Snort ikke var egnet til vårt formål, var det hardt og jobbe videre med et redefinert mål. Til tross for at det opprinnelige målet har vært beskrevet som ”å bli kastet på sjøen med armer og bein bundet”, har vi igjennom hele prosjektet hatt troen på å nå målet. Til tross for vår begrensede kunnskap om Linux, syntes vi det har gått greit, og vi har på ”kjøpet” fått med oss en god dose med Linux kunnskap.

6.3 Videre arbeid med dette prosjektet.

Det vil kunne la seg gjøre og utvikle programvare, som på bakgrunn av alarmer fra Snort, stopper pakker enkeltvis. Da vi startet prosjektet så vi for oss at denne muligheten lå innebygd i Snort, eller enkelt kunne legges til. Tidsmessig har vi ikke tro på at vi kunne rukket å lage en slik applikasjon i vårt prosjekt. Hadde vi hatt tidligere kjennskap til denne programvaren, ville det sikkert latt seg gjøre. Vårt forslag til faglærer i Nettverksfaget, som også er vår veileder, Hans-Jørgen Alker, er å kjøre nettverks- lab på grunnlag av vår Installasjonsmanual for Snort,

vedlegg 1. Her vil det også kanskje være mulighet for å en enkel presentasjon av Snort inn i undervisningen.

6.4 Alternativt bruk av NIDS

Det finnes mange måter man kan bruke en NIDS på. Slik som var utgangspunktet i vårt prosjekt, å begrense trafikk ved og lage filtre som stoppet pakker på bakgrunn av alarmer generert av Snort.

Om man lager et filter som stopper trafikken på bakgrunn av innhold i pakker, kan man også bruke Snort, som et effektivt filter for å stoppe uønsket trafikk, som kan være så mangt. Man kan for eksempel hjelpe foreldre med en av de største bekymringer for internettbruk, å sperre for uønsket innhold på internettsider.

Supplerende informasjon

Kapittel 7 Programpakker for en komplett NIDS, med Snort

For å få til den løsningen vi har tenkt oss, er det fem primære programpakker vi trengte. De fem er Apache web server, MySQL database server, ACID, Webmin og Snort.
De tre første brukes hovedsakelig til maskinen med databasen og de to siste til maskinen med sensoren.

7.1 Apache Web Server

Dette er web serveren som brukes av mesteparten av websidene som aksesseres på Internet. Hensikten med å bruke Apache er for å hoste ACID web baserte konsollen.

7.2 MySQL Server

MySQL er en SQL basert database server for et mangfold av plattformer (operativsystemer) og er den plattformen som har best systemstøtte i å lagre Snort alarmer. Alle IDS alarmer trigget fra sensoren blir lagret i MySQL databasen.

7.3 Analysis Console for Intrusion Databases (ACID)

ACID er en web basert applikasjon for å vise brannmurer og/eller IDS varslinger. Dette er hvor all informasjonen fra sensoren blir lagt ut for visning.

7.4 Webmin

Webmin er et web basert grensesnitt for å administrere Unix baserte servere. Det forsyner et grafisk grensesnitt til de fleste servicer og konfigurasjons muligheter som er tilgjengelig på skall nivå. Webmin er skrevet i Perl og nye moduler blir utviklet hele tiden. Det finnes også

en Snort modul som er installert slik at Snort kan administreres grafisk. Her kan regler opprettes og redigeres, samt at alle Snort konfigurasjoner kan justeres.

7.5 Snort

Snort er en lettvekter i nettverk intrusjon påvisning (Intrusion Detection System), som kan utføre sanntid trafikk-analyser og pakke logging på IP-nettverk.
Snort kan også konfigureres i to andre modus enn IDS. Disse to er ”sniffer” og ”pakke-logger”. Dette er mindre kompliserte modus enn IDS som vi har konfigurert Snort til.

7.6 SSL – Secure Sockets Layer

Grunnen til at man bruker SSL Protokoll er primært for å sørge for funksjonsstabilitet og pålitelighet mellom to kommuniserende applikasjoner. Protokollen er sammensatt av to lag. På det laveste laget, over en pålitelig transport protokoll (f.eks TCP) finner vi SSL Record protokollen. SSL Record protokollen blir brukt til innkapsling av protokoller på et høyere nivå. SSL Handshake protokoll er en slik innkapslet protokoll. Den tillater at tjener og klient verifiserer hverandre og blir enige om en kryptisk algoritme og kryptografiske nøkler før applikasjon protokollen sender eller mottar sine først data.

Kapittel 8 NIDS – forklaringer

8.1 Hva er og hvordan virker en NIDS?

En NIDS kan enten være en software eller hardware basert løsning, som er designet for å detektere uautorisert bruk av, eller angrep på, et datamaskin system eller et nettverk. En NIDS ser etter uautorisert forsøk på å oppnå tilgang til et system, gradvis få flere privilegier på et uautorisert system, eller for å minske tilgjengeligheten til et system enten fra innsiden eller fra utsiden av organisasjonen eller fra Internet. En NIDS er bare en del av en sammensatt og overlappende sikkerhets politikk.

NIDS kommer i mange former, med forsjellige måter for overvåkning og analysering av tilgjengelig data. NIDS overvåker handlinger på tre forsjellige nivåer; nettverk, vert og applikasjon. De kan analysere disse handlingene ved å bruke to teknikker; signatur deteksjon og uregelmessighets deteksjon (anomali). Vi skal bare ta for oss signatur deteksjon, som Snort bruker. Noen NIDS har evnen til å gjøre handling når et angrep er oppdaget, men dette er noe man må gjøre med stor aksomhet, med tanke på lovlighet og tilgjengelighet. Snort har begrensede muligheter på dette område se avsnitt.

Signatur deteksjon er den mest brukte i kommersielle NIDS produkter, men uregelmessighets deteksjon (anomali) er nyere og øker i omfang.

8.2 Signatur- basert deteksjon

Signatur basert deteksjon ser etter aktivitet som ligner en predefinert streng som beskriver et unikt bestemt angrep. Signatur basert NIDS må være spesielt programmert for å detektere hvert angrep. Denne teknikken er ekstremt effektiv mot kjente angrep, og må konstant oppdateres for å holde følge med på nye angrepsmetoder.

8.3 Angrep

8.3.1 Buffer Overflow angrep

Et buffer overflow oppstår i et program når programmet lagrer mer informasjon i en array, enn det er reservert plass for. Dette fører til at nabo områdene i bufferet blir overskrevet, noe som igjen fører til at dataene blir korrupte. Buffer overflows er alltid programfeil som typisk blir introdusert til et program fordi programmereren ikke klarte å få til at informasjonen som kopieres inn i programmet ikke kan være større enn en gitt verdi. Uheldigvis, er buffer overflow programfeil å se ganske ofte, på grunn av vanlig og farlig programmerings praksis. Når et program er sårbart for buffer overflow, kan denne være gjemt i programmet i flere år. Dette potensialet gjør at programmet blir et mål for et plutselig angrep som utnytter sårbarheten, for å oppnå uautorisert tilgang til et system.

En buffer overflow kan skje, som en tilfeldighet, ved eksekveringen av et program. Når dette skjer, er det lite trolig at det vil føre til et sikkerhets kompromiss for systemet. I de fleste tilfeller vil det føre til at informasjonen i området som blir rammet av overflow blir ødelagt, og at programmet krasjer eller lager et åpenbart ukorrekt resultat. På en annen side, i et buffer overflow angrep, bruker den som angriper sårbarheten for å lage korrupt informasjon, som er gjennomtenkt og designet for å angripe kode som tidligere er plantet av angriperen. Hvis det gjennomføres slik det var tenkt, kan den som angriper ta kontorollen over programmet. Når kontrollen er overført til koden, gis det uautorisert tilgang til angriperen. Typisk vil angrepskoden ligge som et skall, noe som tillater den som angriper å kjøre kommandoer på et system.

8.3.2 Port scans

En port scanner er software som samler inn informasjon om portene på en datamaskin står åpne. Når en port er åpen på en maskin ,fortrinnsvis en server, vil det være mulig for en annen maskin å koble seg til.

Internet er som kjent basert på IP protokollen, denne bruker port notasjon for å koble maskiner sammen. Hver gang du besøker en web side, blir en port på maskinen din åpnet og koblet til porten på serveren hvor dokumentet ligger. Vanligvis, på klient siden, blir det brukt

en tilfeldig port for å koble til serveren. På server siden blir det vanligvis brukt en standard port for å lytte etter innkommende koblinger fra klient porter. Det er vanlig for web servere å lytte på port 80 for web forespørsler, port 25 for e- post overføring, port 110 for å laste ned mail etc.

En port scanner vil si deg informasjon om alle portene på en maskin. Resultatet kan gi deg en god pekepinn på hva slags system som blir kjørt på maskinen, ved å sjekke opp port numrene mot hva slags programmer som normalt kjøres på disse portene.

Det er mange forsjellige måter for å utføre en port scanning. Den enkleste metoden er å skrive et program som kjører en full forbindelse på alle porter på en maskin fra 1 til 65535. De fleste hackere foretrekker å bruke ”stealth scan” metoder, som ikke lager en full forbindelse til serveren, og av den grunn ikke blir logget hele tiden. I de fleste tilfeller tillater denne metoden hackere å sjekke portene på en maskin uten å vise mistenkelig aktivitet på målets logg filer. En av Snorts fordeler er at denne type trafikk vil være synlig.

Den mest brukte metoden for å gjøre et ”stealth scan” er syn-ack metoden. Normalt, for at en kobling skal gjøres mellom to porter, må klienten sende en SYN pakke til serverens port , og serveren svarer med en ACK pakke til klienten og til slutt sender klienten er SYN pakke tilbake til serveren for å komplettere ”handshake”. Etter denne prosessen logger vanligvis serveren koblingen som opprettet, til den IP adressen som koblingens er gjort på. En vanlig ”stealth skanner” vil enkelt utelate den siste SYN pakken, som vil gjøre at serveren tror at koblingen ble droppet. En annen vanlig metode for angriperen å gjemme seg på, eller å utføre en port scan gjennom en brannmur, blir kalt ftp bounce angrep. Siden FTP protokollen tillater en klient å spesifisere port og IP adresse for å koble seg til for å overføre filer, kan angriperen spesifisere port og IP adresse som den vil skanne istedenfor sin egen .

8.3.3 CGI angrep

Enhver Internet server er mål for et angrep mot feil som er mulig og utnytte, og de fleste servere som har mye trafikk er satt opp og testet av eksperter. Hovedproblemet med CGI script er at de hver og en fungerer som en miniatyr web- server, tilgjengelige for alle, og nye

CGI script blir skrevet hver dag av nybegynnere og programmerere som ikke er opptatt av sikkerhet. Alle disse CGI scriptene kan inneholde sikkerhetshull som er mulig å utnytte.

CGI script kan benyttes til å oppnå tilgang til informasjon om systemet som kan brukes av hackere for å bryte seg inn. Hackere vil ofte prøve å skaffe seg en vanlig bruker login for å plyndre seg gjennom mapper eller se igjennom leselige /etc filer for å kanskje skaffe seg root tilgang. Et CGI script med sikkerhetshull kan brukes på samme måte, uten bruker login, for å skaffe den samme informasjonen.

Et annen og farligere CGI sikkerhetshull oppstår i CGI programmer som prosesserer informasjon fra et skjema (eng. Form) . Skjemaer er nyttige og kan brukes på en sikker måte, men noen CGI programmer bruker skjema informasjon på en måte som ikke er sikker, og kan bli lurt til å eksekvere andre programmer (spesifisert av en hacker), mens den prosesserer skjema informasjonen. Å eksekvere andre programmer kan kanskje ikke se ut som det er så farlig hvis CGI programmet er kjørt med ”nobody” tilgang, men på mange, eller de fleste systemer, er dette kritisk for sikkerheten, der hackeren kan:

- Sende passwd filen eller annen /etc informasjon på mail, eller kople (mappe) system mapper til seg selv.
- Starte en telnet daemon på en høy port slik at de kan logge seg inn.
- Suge til seg så mye CPU at web serveren ikke lenger klarer de oppgaver den skal gjøre.
- Fjerne eller endre server logg filer

8.3.4 OS fingerprinting

Operativ system fingerprinting er evnen til å identifisere et fjerntliggende operativsystemer ved å analysere måten det reagerer på visse typer pakker. Det kan sammenlignes med mennesket fingeravtrykk, fordi hvert enkelt operativsystem har sin egen måte å reagere på i de enkelte situasjoner.

8.3.5 DoS angrep

Et DoS (Denial of Service) angrep skjer når en maskin prøver å stoppe en annen maskin fra å tilby services til sine klienter. Den aktuelle metoden kan variere fra å enkelt krasje maskinen

til å oversvømme den nettverks tilkobling. DoS angrep utføres ofte mot store web servere, for å blokkere disse for andre besøkende.

DDos angrep: Distributed Denial of Service. Et DoS angrep hvor en hacker tar i bruk flere maskiner og bruker disse for å oversvømme en bestemt IP adresse med forespørsler, for å sette dette systemet ut av spill. Ved å bruke flere maskiner til et angrep, forsterkes angrepet og det blir vanskeligere å identifisere hackeren.

Referanser

Snort:	http://www.snort.org
	http://www.linuxdoc.org/HOWTO/Snort-Statistics-HOWTO/
	http://www.dpo.uab.edu/~andrewb/snort/manpage.html
	http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/snort/snort/
ACID	http://acidlab.sourceforge.net/
MySQL	http://www.mysql.com/
Webmin	http://www.webmin.com/
Redhat	http://www.redhat.com/
	http://www.redhat.com/support/resources/howto/rhl73.html
	https://rhn.redhat.com/help/basic/
Diverse søk	http://www.google.com

Vedlegg

Vedlegg 1: ”Installasjonsmanual for Snort”, som en komplett NIDS.

Vedlegg 2: ”snort.conf”

Vedlegg 3: ”Brukermanual for Snort”, skrevet av Martin Roesch, oversatt av Geir Solli og Dag Atle Isene.

Ønskes mer informasjon om oppgaven, henvend deg til Høgskolen i Vestfold, avdeling RI.