ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

DIPLOMSKI RAD br. 1708

RASPODIJELJENI SUSTAV ZA SPRIJEČAVANJE MREŽNIH Nnapada

Igor Požgaj

Zagreb, veljača 2008.

Zahvaljujem svima koji su mi na bilo koji način pomogli u izradi ovog diplomskog rada. Posebno zahvaljujem mr.sc. Stjepanu Grošu i prof.dr.sc. Vladi Glaviniću na stručnom vodstvu, te obitelji i mojoj djevojci Meliti na pruženoj podršci.

Sažetak

Uz standardne tehnike zaštite računalnih mreža često se koriste i sustavi čija je osnovna namjena prepoznavanje sumnjivih aktivnosti i napada na mreži. Cilj ovoga rada je prikazati funkcije i način rada takvih sustava, opisati arhitekturu tipičnog sustava za otkrivanje mrežnih napada, te prikazati najčešće probleme kod njihovog korištenja kao i neka od mogućih rješenja.

Kao praktični dio rada razvijen je jednostavni program s osnovnim funkcijama sustava za prepoznavanje i sprječavanje mrežnih napada, Korišteni su standardni alati otvorenog koda te programski jezik Python. Funkcionalnost programa obuhvaća osluškivanje mrežnog prometa, prepoznavanje raznih vrsta napada, njihovo zapisivanje u datoteke dnevnika, te sprječavanje daljnjih napada. Sprječavanje napada moguće je obavljati ručno definiranjem odgovarajuće sigurnosne politike ili automatski zadavanjem određenih pravila unutar programa.

Abstract

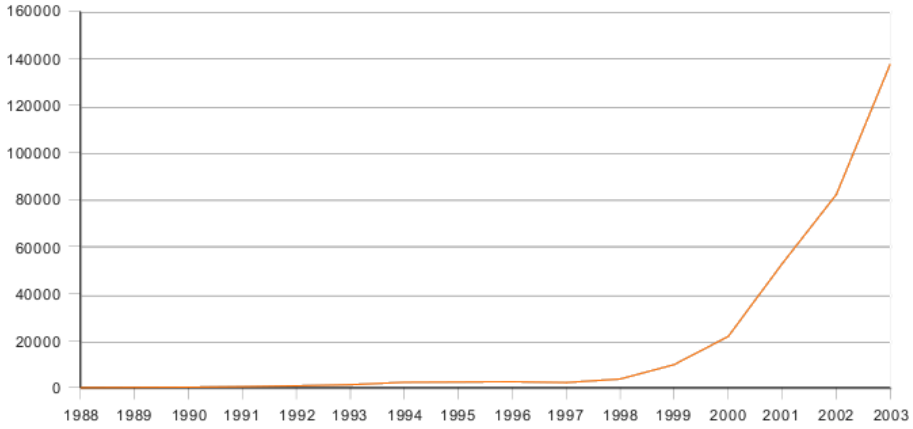
In addition to standard techniques used for protecting the computer networks, systems which main goal is detection of attacks and suspicious actions are also commonly used. Goal of this thesis is to elaborate tasks and functionality of such systems, describe the typical architecture of network intrusion detection system, and to show the most common problems of using such systems as well as solutions for that problems.

Practical part of the thesis includes developed application with basic functionalities of intrusion detection and prevention system. Program is developed in Python programming language and by using the common open source tools. Functionalities of application include network traffic sniffing, detection of various kinds of network attacks, logging of attacks and prevention of further attacks. Prevention of attacks is possible by manually defining the security policy, or it can be taken automatically by using special commands inside the application.

Sadržaj

1. Uvod

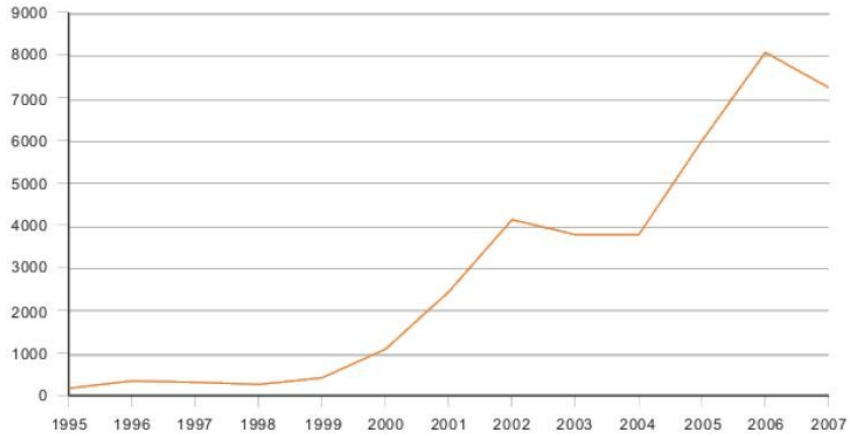
Računalna sigurnost je, zahvaljujući sve bržem rastu računalne industrije, postala nezaobilazna tema u projektiranju bilo kojeg informacijskog sustava. Novi propusti i metode napada na razne sustave otkrivaju se gotovo svakodnevno. Prema podacima preuzetih od CERT-a (eng. *Computer Emergency Response Team*) broj prijavljenih sigurnosnih incidenata u razdoblju od petnaest godina porastao je gotovo 1000 puta, kao što prikazuje slika 1.1, a praktički se udvostručuje svake sljedeće godine [7]. Razlozi za ovakav drastičan porast broja sigurnosnih incidenata su mnogobrojni. Posljednjih godina pristup Internetu je sve jednostavniji i jeftiniji, a broj korisnika se neprestano povećava. Nadalje, tržištem trenutno dominira vrlo mali broj operacijskih sustava, pa pronalaženjem propusta u bilo kojem od njih napadač automatski dobiva veliki broj potencijalnih žrtava na kojima može iskoristiti pronađeni propust. Brzi razvoj tehnologije često izbacuje na tržište nekompletna i neprovjerena rješenja koja na kraju rezultiraju velikim brojem sigurnosnih propusta. Često navođeni primjer je slučaj WEP protokola (eng. *Wired Equivalent Privacy*) u bežičnim mrežama. Dvije godine nakon objave pronađeni su ozbiljni propusti u protokolu, a u samo nekoliko idućih mjeseci pojavio se niz alata za iskorištavanje njegovih nedostataka (npr. *airsnort*). Uza sve navedeno, popularizacijom Interneta informacije o novim propustima se trivijalno i vrlo brzo šire među velikim brojem ljudi, a pronalaženje alata za napade na razne informacijske sustave svodi se na jednostavno upisivanje ključnih riječi u tražilicu, te preuzimanje gotovih alata s jedne od mnogobrojnih *crackerskih* stranica. Zbog toga znanje i vještine napadača postaju sve manje važni faktori, te sve veći udio u populaciji napadača čine takozvani *script kiddies,* napadači koji koriste već gotove alate bez detaljnog znanja o alatu koji koriste ili o propustu koji iskorištavaju.



Slika 1.1: Prijavljeni sigurnosni incidenti [CERT]

Struktura napada se tijekom vremena također drastično promijenila. Prema podacima iz CERT-a, prije desetak godina većinu prijavljenih incidenata su sačinjavale prijave vezane uz štetno djelovanje virusa, pogađanje korisničkih lozinki raznim metodama pretraživanja (eng. *brute force password guessing*), te iskorištavanje dobro poznatih propusta u sustavima. Porastom kompleksnosti informacijskih sustava uvelike se povećao i broj sigurnosnih propusta, što je jasno vidljivo i na slici 1.2. Danas većinu prijava CERT-u

sačinjavaju razni tipovi napada s mreže u rasponu od jednostavnog prikrivenog skeniranja pristupnih vrata (eng. *stealth port scanning*) pa sve do vrlo ozbiljnih napada poput raspodijeljenih napada uskraćivanjem usluge (eng. *distributed denial of service*). U vrijeme kada je gotovo svako računalo priključeno na Internet i zbog svih prethodno navedenih razloga, mrežna sigurnost i razvoj sigurnosnih rješenja za obranu od napada s mreže čine jedno od najbrže razvijajućih područja računalne industrije. Posebnu ulogu u tome ima i razvoj alata za obranu ranjivih mrežnih usluga, među kojima važnu ulogu imaju i sustavi za otkrivanje napada sa mreže.



Slika 1.2: Pronađeni sigurnosni propusti

Paralelno s razvojem raznih metoda napada na informacijske sustave razvijali su se i obrambeni mehanizmi i tehnike. Jedna od prvih, a danas vjerojatno i najčešće korištenih metoda je korištenje mrežne sigurnosne stijene (eng. *firewall*). Sigurnosna stijena je u svojoj osnovi programsko ili sklopovsko rješenje koje djeluje kao filtar paketa i služi za osnovno nadziranje prometa između raznih područja povjerenja (eng. *zones of trust*), čime je moguće u potpunosti zabraniti ili selektivno propuštati promet prema određenim grupama računala. Tipična takva područja koja se koriste su *lokalna mreža* (eng. *local network*) s najvećim stupnjem povjerenja, demilitarizirano područje (eng. *demilitarized zone*) s manjim stupnjem povjerenja, te *Internet* s najmanjim stupnjem povjerenja. Prve implementacije sigurnosne stijene su bile statične i nisu pratile stanje veze (eng. s*tateless packet filter*) pa su mogle blokirati promet samo u ovisnosti o mrežnim adresama i pristupnim vratima (eng. *port*). Danas je takav pristup uglavnom manje zastupljen i koriste se sigurnosne stijene koje mogu pratiti i stanje veze (eng. *stateful packet filter*). Praćenje stanja veze omogućuje izradu puno složenijih pravila jer se svakoj vezi pridružuje i jedno od mogućih stanja. Neka od moguća stanja veze su *nova* (eng, *new*), *uspostavljena* (eng. *established*), *povezana* (eng. *related*) itd. Ova tehnika također omogućava prevođenje adresa u sigurnosnoj stijeni (eng. *network address translation*, NAT) kao i bolje funkcioniranje s problematičnim mrežnim uslugama poput FTP-a (eng. *file transfer protocol*).

Glavni nedostatak zaštite sigurnosnom stijenom je što većina takvih rješenja radi na mrežnom (L3) i prijenosnom (L4) sloju, čime je moguće propuštanje ili zabranjivanje prometa samo u ovisnosti o podacima dostupnim na tim slojevima. Moderne metode zaštite zahtijevaju analizu prometa na slojevima višim od prijenosnog, posebice na aplikacijskom (L7) sloju. Primjerice, paket koji dolazi na poslužitelj i sadrži štetni

programski kôd koji će napadaču omogućiti pristup ljusci operacijskog sustava (eng. *shell*), sigurnosna stijena neće moći odbaciti jer su na mrežnom i prijenosnom sloju dostupne samo informacije o mrežnim adresama odnosno pristupnim vratima, pa sigurnosna stijena ne može znati ništa o sadržaju paketa. Za puno veći stupanj nadzora prometa koji prolazi mrežom potrebno rješenje koje će između ostalog moći analizirati i sadržaj samoga paketa na aplikacijskom sloju, te na osnovu tih informacija odlučiti da li će paket biti propušten ili odbačen. Upravo je to jedna od funkcija sustava za otkrivanje napada sa mreže.

Ponekad se za analizu sigurnosti na određenoj mreži koriste i sustavi za privlačenje napadača (eng. *honeypots*). Sustav za privlačenje napadača je računalni sustav koji se nalazi u istoj mreži na kojoj se nalazi i stvarni poslužitelj čiju sigurnost treba ispitivati, a čija je osnovna namjena zavarati napadača tako da povjeruje da radi sa stvarnim poslužiteljem. Cilj ove tehnike je mogućnost pravovremenog dobijanja informacije o napadu, te dobijanje informacija o tehnikama koje napadač koristi da bi ostvario taj napad. Sustav uobičajeno ima pokrenute usluge koje napadači najčešće napadaju (*telnet, ssh, ftp, httpd, nfs, samba*), a sigurnosni mehanizmi na njemu su puno slabiji od onih na stvarnom poslužitelju. Često se i pristupna vrata navedenih mrežnih usluga sa stvarnog poslužitelja preusmjeravaju na pristupna vrata sustava za privlačenje napadača, pod uvjetom da se ne koriste za neki korisni promet. Računalni sustav za privlačenje napadača mora biti dobro izoliran od stvarnog poslužitelja iz razloga da napadač nema mogućnost napada na ostala računala u mreži ukoliko uspije kompromitirati napadnuto računalo. Ovakvi sustavi su vrlo često usko objedinjeni sa sustavima za otkrivanje napada sa mreže.

Danas se, zahvaljujući vrlo brzom razvoju virtualizacijskih tehnika i alata (VMW*are, Xen, QEMU, kvm* itd.) najčešće koriste već gotova rješenja koja obuhvaćaju sva tri navedena aspekta zaštite i nadziranja sigurnosti (sigurnosnu stijenu, sustav za otkrivanje napada sa mreže i sustav za privlačenje napadača). Prednost takvog pristupa je mogućnost vrlo brzog postavljanja sustava za nadzor i zaštitu u gotovo bilo koju mrežnu topologiju, te uz vrlo nisku cijenu i troškove održavanja. Takva rješenja su često objavljenja pod raznim licencama otvorenog kôda i razvijaju se gotovo svakodnevno, pa predstavljaju ozbiljnu konkurenciju i vrlo skupim komercijalnim i vlasničkim (eng. *proprietary)* proizvodima.

2. Sustavi za otkrivanje mrežnih napada

2.1. Definicija i funkcionalnost

Sustavi za otkrivanje napada (eng. *intrusion detection systems*) su dio skupa tehnologija za podizanje ukupne razine sigurnosti informacijskih sustava. Njihov rad se zasniva na prikupljanju informacija s čitavog niza mrežnih i računalnih izvora, te analiziranju tih informacija s ciljem otkrivanja eventualnih nedozvoljenih aktivnosti i zlouporabe sustava na kojem se nalaze. Sustavi za otkrivanje napada (u daljnjem tekstu koristi se u literaturi uobičajena kratica *IDS*) prate i analiziraju mrežni promet i različite dijelove operacijskog sustava, međutim ne poduzimaju nikakve akcije koje bi spriječile štetni ili neželjeni promet. Ukoliko se takav promet uoči, najčešće se samo zabilježi u dnevnik (eng. *log*), te se obavještavaju nadležne osobe putem različitih mehanizama obavještavanja. U literaturi se često razlikuju sustavi koji otkrivaju napade od onih koji otkrivaju pogrešno korištenje sustava. Uobičajeno se napadom smatra štetna aktivnost koja na sustav djeluje s lokacije izvan sustava, a iste takve aktivnosti koje djeluju unutar samog sustava se smatraju pogrešnim korištenjem. Napadi se od pogrešnog korištenja dodatno razlikuju i po svojem intenzitetu odnosno brzini pojavljivanja, važnosti napadnutog objekta itd. Na primjer, zaposlenik prijavljen za rad na jednom od računala u lokalnoj mreži koji više puta za redom unese krivo korisničko ime kod prijavljivanja na neku mrežnu uslugu u korporacijskom intranetu krivo koristi sustav. Isti postupak, ali s računala izvan tvrtkine lokalne mreže te koji se pojavljuje stotinjak puta unutar jedne minute, smatra se napadom. U većini implementacija ovakvih programa najčešće se ne pravi razlika između napada i pogrešnog korištenja sustava već se oboje tretiraju jednako, kao napad.

Rad IDS-a zasniva se na nadziranju pojedinih dijelova operacijskog sustava, te analiziranju zaglavlja i sadržaja paketa na raznim slojevima mrežnog stoga. Obje vrste podataka sadrže informacije o aktivnostima korisnika, pa je na osnovu analize njihova sadržaja moguće prepoznati neuobičajene aktivnosti i napade. Pri analiziranju mrežnog prometa IDS ni na koji način ne smije usporavati mrežni promet – njegov rad mora biti u potpunosti nevidljiv korisnicima mreže i svih sustava na mreži. Iz razloga što analiza cjelokupnog prometa na visoko propusnim mrežama može biti prilično zahtjevan posao, primjenjuju se posebne metode, npr. mrežne priključnice (eng. *network taps*), da bi se izbjegle takve vrste problema. Analizu mrežnog prometa moguće je obavljati na računalu posebno namijenjenom samo svrsi prepoznavanja i praćenja napada na određeni poslužitelj, aplikaciju ili grupu računala u mreži. Iako je moguće smjestiti IDS na isto računalo na kojem je instaliran i poslužitelj koji se želi nadzirati, najčešće se ipak nalazi na zasebnom računalu namijenjenom posebno prepoznavanju napada. U literaturi se ovakvo računalo namijenjeno isključivo radu IDS-a ili nekih od njegovih komponenti naziva *senzor* [5].

Za prepoznavanje štetnih aktivnosti na računalnom sustavu ili mreži, IDS koristi dvije uobičajene metode: *prepoznavanje potpisa* i *prepoznavanje neuobičajenog ponašanja*. Kod prepoznavanja potpisa (eng. *signatures detecting, pattern matching*) sadržaj dnevnika ili mrežnih paketa šalje se u posebni pretraživački program (eng. *detection engine*) gdje se uspoređuje s unaprijed poznatim i definiranim potpisima napada ili s unaprijed definiranim uobičajenim ponašanjem sustava. Potpis napada je skup uzoraka na temelju kojih sustav

može otkriti da pročitani paket predstavlja potencijalni napad na sustav. S druge strane, sustavi koji koriste prepoznavanje neuobičajenog ponašanja rade na načelima prikupljanja i statističke obrade podataka o uobičajenom radu sustava. Statistička obrada podataka služi da bi sustav za otkrivanje napada odredio uobičajeno korištenje sustav poput najčešće korištenih protokola, pristupnih vrata, adresa i vremena pristupa uslugama. Kada se primijeti znatnije odstupanje od tako sakupljenih i statistički obrađenih podataka, IDS će smatrati da je u tijeku napad na sustav. Sustav za otkrivanje koji koristi prepoznavanje neuobičajenog ponašanja u pravilu koristi više sistemskih resursa, ali će nakon početnog razdoblja učenja moći prepoznati i neke od novih vrsta napada, dok je sustav koji koristi prepoznavanje potpisa učinkovit samo toliko koliko je ažurna njegova baza podataka s potpisima napada. Usporedba pojedinih važnijih karakteristika sustava koji su zasnovani na prepoznavanju potpisa, te onih koji koriste prepoznavanje neuobičajenog ponašanja, prikazana je tablicom 2.1.

Prepoznavanje potpisima	Prepoznavanje neuobičajenog ponašanja
Manji broj lažnih prepoznavanja	Mogući veći broj lažnih prepoznavanja
Proizvođač IDS mora održavati i redovito izdavati potpise novih otkrivenih vrsta napada	Prilagodljiviji sustav, može otkriti i vrste napada koje se ne nalaze u bazi s potpisima napada
Brža obrada, zahtjeva manje računalnih resursa	Zahtjeva više računalnih resursa, sporija obrada
Nije potrebno početno razdoblje "učenje" sustava da bi prepoznavao napade, koriste se već gotovi potpisi napada	Potrebno je početno razdoblje "učenja" sustava za prepoznavanje napada, može predstavljati problem u dinamičnim sredinama
Teško održavanje pravila	Manje pravila i lakše održavanje

Tablica 2.1: Karakteristike različith načina prepoznavanja napada

Potrebno je i napomenuti da većina besplatnih kao i komercijalnih IDS-ova danas koristi kombinaciju obje navedene metode prepoznavanja, ali se ipak veći naglasak stavlja na metodu prepoznavanja potpisa. Primjer takvog programa je i sustav za otkrivanje napada sa mreže Snort koji je opisan kasnije u nastavku rada.

2.2. Vrste IDS-a

Za razliku od klasifikacije sustava za prepoznavanje napada po načinu na koji prepoznaju napade, u literaturi se IDS-ovi najčešće klasificiraju prema svojoj namijeni. Po toj podjeli možemo ih razvrstati u dvije osnovne skupine:

- računalno temeljeni IDS
- mrežni IDS

Računalno temeljeni IDS (eng. *host based intrusion detection system - HIDS*) je sustav namijenjen analiziranju mrežnog prometa koji je usmjeren prema samom računalu na kojem je postavljen IDS, te analiziranju datotečnog sustava, procesa i drugih važnijih dijelova operacijskog sustava. Vrste napada koje može otkrivati ovakav sustav su vrlo

raznolike. Rad HIDS-a se zasniva na prikupljanju potpisa ili sažetka (eng. *hash, digest*) te praćenju dozvola na kritičnim sistemskim datotekama, nadziranju rada s datotečnim sustavom, nadziranju procesa, memorije itd. Važan dio HIDS-a je baza podataka koja sadrži informacije o ranjivim programima i uslugama na osnovu kojih sustav može procijeniti rizičnost pojedinih pokrenutih aplikacija, te dati ukupnu ocjenu o sigurnosti. Primjeri često korištenih HIDS programa su *nessus* [19] i *samhain* [20], ali i manji alati za otkrivanje štetnog programskog kôda poput *chkrootkit*, *rkhunter* i *tripwire*.

Mrežni IDS (eng. *network intrusion detection system – NIDS*) je sustav namijenjen prikupljanju i analiziranju mrežnog prometa koji nije nužno stvoren ili usmjeren prema računalu na kojem je postavljen IDS, to jest prema IDS senzoru. Cilj NIDS-a je nadzirati rad ranjivih mrežnih usluga i aplikacija, pratiti rad korisnika na mreži te paziti na dobro poznate propuste u mrežnim protokolima i porukama koje se razmjenjuju. NIDS u svojoj osnovi djeluje vrlo slično kao i aplikacijska (L7) sigurnosna stijena ili aplikacijski prilaz (eng. *application level gateway*). Za razliku od sigurnosne stijene, NIDS ne može utjecati na stanje veze pa se njegove aktivnosti svode na prepoznavanje i upozoravanje na napade. Nadalje, način rada i funkcija NIDS-a se bitno razlikuje od aplikacijske sigurnosne stijene koja je primarno namijenjena selektivnom propuštanju mrežnog prometa koji stvaraju programi koristeći skup točno određenih mrežnih protokola. Glavna tema ovog rada su mrežni IDS-ovi pa će se pod pojmom IDS, ukoliko nije drukčije eksplicitno navedeno, u nastavku rada podrazumijevati mrežni IDS (NIDS).

Mrežni IDS-ovi mogu otkriti veliki broj različitih vrsta napada na mreži zahvaljujući već unaprijed poznatim potpisima napada ili podacima o unaprijed ustanovljenom uobičajenom korištenju sustava. Glavne vrste potpisa napada vezane su uz nadziranje karakterističnih binarnih ili tekstualnih nizova znakova u mrežnim paketima. Najčešće se nadziru sadržaj paketa (eng. *payload*) na aplikacijskom sloju te kompletna zaglavlja paketa (eng. *packet header*) na mrežnom i prijenosnom sloju. Primjer niza znakova koji se može tražiti, te koji će nakon otkrivanja aktivirati upozorenje je niz

```
cat "+ +" > /.rhosts
```

koji ukoliko se izvrši u ljusci operacijskog sustava UNIX može učiniti sustav vrlo izložen daljnjim napadima. Nizovi znakova koji se također često traže su "`cgi-bin`", "`ifs`", "`aglimpse`" i drugi. Druga vrsta potpisa traži sumnjive pakete koji stižu na računalo i cilj im je pristupiti karakterističnim pristupnim vratima (NetBIOS, SunRPC, telnet, FTP itd.), a na lokalnom računalu nisu nužno pokrenute mrežne usluge na tim pristupnim vratima. Posljednja vrsta potpisa zadužena je za otkrivanje neuobičajenih kombinacija zastavica u zaglavljima mrežnih paketa. Najpoznatiji primjer programa koji je koristio propust s neuobičajenim kombinacijama zastavica je program Winnuke koji šalje pakete na NetBIOS pristupna vrata s uključenim zastavicama URG (eng. urgent) i OOB (eng. out of band). Zbog propusta u operacijskom sustavu Microsoft Windows takvi paketi su uzrokovali trenutno rušenje sustava. Sličan primjer zloćudnih paketa su i paketi koji u zaglavlju imaju uključenu i SYN i FIN zastavicu čime napadač istovremeno pokušava uspostaviti i zatvoriti TCP vezu. U ovu klasu potpisa pripadaju i potpisi kojima je cilj otkriti razne vrste skeniranja pristupnih vrata (syn, connect, xmas, ymas itd.). Najpoznatiji programi koji mogu otkriti sve prethodno navedene vrste napada su Snort, Axent, Prelude, Cisco Secure IDS, ISS, Shadow itd.

2.3. Arhitektura IDS-a

IDS je logički podijeljen na nekoliko zasebnih komponenti. Logička podjela može odgovarati i stvarnoj fizičkoj realizaciji sustava, međutim češći je slučaj gdje su pojedine komponente udružene u kompleksnije cjeline ili su dodatno razdvojene na još nekoliko komponenti. Pojedine komponente sustava međusobno komuniciraju s zajedničkim ciljem učinkovitog otkrivanja pojedinih vrsta napada te stvaranja sigurnosnih upozorenja i njihovog spremanja u dnevnik. Komunikacija između pojedinih komponenti ostvarena je razmjenom poruka. Primarna namjena razdvajanja cjelokupnog sustava na više međusobno ovisnih komponenti je jednostavnost razvoja, bolja podjela zadataka unutar sustava te mogućnost korištenja modula iz različitih IDS sustava u jednoj raznolikoj (heterogenoj) cjelini. Pojedine komponente mogu se smještati i na zasebnim računalima čime se ostvaruje mogućnost izrade sustava koji će maksimalno iskoristiti paralelni rad (npr. *clustering* rješenja) i dati najbolja radna svojstva. Nažalost, arhitektura pojedinačnih IDS rješenja u stvarnosti dosta varira od proizvođača do proizvođača, bilo zbog nedovoljne primjene standarda ili namjere proizvođača da onemogući korištenje pojedinih komponenti i od konkurentskih proizvoda (npr. da se uz pretprocesor jednog proizvođača može koristiti sustav otkrivanja drugog proizvođača). Glavna prepreka komunikaciji komponenti raznih proizvođača je uglavnom nedostatak primjene standardnog formata poruka i primjene standardiziranih protokola za razmjenu tih poruka. Umjesto otvorenih i slobodnih češće se koriste zatvoreni i vlasnički formati te poruke, što predstavlja dodatni problem kod međusobne suradnje različitih komponenti, ali i kod procesa standardizacije. U ovome smislu programi otvorenog koda više pridonose standardizaciji protokola i formata poruka jer je njihova specifikacija kao i implementacija javno dostupna, pa je moguće i zajedničko korištenje različitih komponenti. Primjerice, moguće je kombinirati dva vrlo popularna IDS sustava otvorenog koda, Snort i Prelude, tako da Snort obavlja funkcije otkrivanja napada, a otkrivene napade predaje na daljnju obradu programu Prelude.

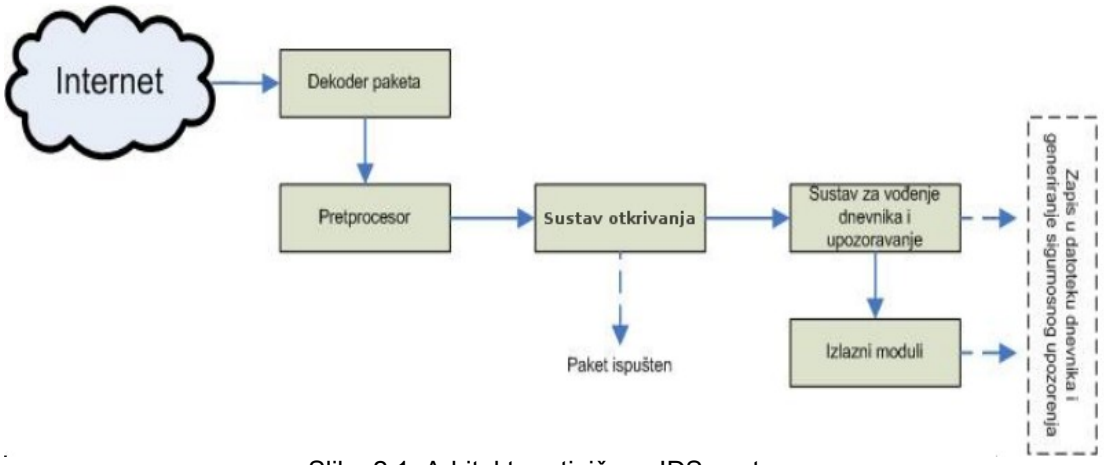
Tipične komponente od kojih se sastoji sustav za otkrivanje napada s mreže su sljedeće:

- dekoder paketa
- pretprocesor
- sustav otkrivanja
- sustav za vođenje dnevnika i upozoravanje
- izlazni moduli

Kao što je već navedeno, moguća su i grupiranja pojedinih komponenti sa sličnim funkcijama u jedinstveni modul. Primjerice dekoder paketa i pretprocesor mogu biti dio jedinstvenog modula za pripremanje mrežnih paketa, a izlazni modul može biti ugrađen u sustav za vođenje dnevnika i upozoravanje, npr. u slučaju da će se koristiti isključivo zapisivanje upozorenja u datoteku dnevnika.

Međusobnu ovisnost pojedinih komponenti IDS-a prikazuje slika 2.1. Tok paketa kroz sustav prikazan je punom strelicom. Potencijalne akcije koje IDS može poduzeti na paketima prikazane su isprekidanom crtom. Na putu od dekodera paketa do izlaznih modula paket može biti ispušten, odnosno za njega se neće stvoriti nikakva sigurnosna upozorenja, ili će se stvoriti odgovarajuće sigurnosno upozorenje i zapis u dnevniku.

Moguće je da u nekim implementacijama IDS-ova neke od navedenih komponenti nisu zasebno realizirane ili se uopće niti ne koriste. Komponente koje se u pravilu nalaze u svakom IDS-u su dekoder paketa, pretprocesor i sustav otkrivanja. Za sustav otkrivanja ponegdje se koristi i naziv *engine*.



Slika 2.1: Arhitektura tipičnog IDS sustava

Sve navedene komponente čine IDS, odnosno IDS senzor. Cjelokupni sustav za nadzor mreže može se sastojati od jednog ili više senzora. Ponegdje se kao posebna vrsta IDS senzora izdvaja i konzola za nadziranje rada sustava. Konzola za nadziranje sustava služi za praćenje rada (eng. *monitoring*) i upravljanje pojedinim dijelovima sustava, primarno IDS senzorima i njihovim funkcijama. U većini implementacija IDS-ova, konzola za nadziranje sustava nije ništa drugo nego posebno izdvojeni senzor ili još češće svaki senzor ima mogućnost upravljanja ostalim senzorima, tj. ne pravi se razlika između nadzornog (aktivnog) i ostalih senzora. Funkcionalnost nadzornog senzora najčešće se može koristiti putem posebnog programa s grafičkim korisničkim sučeljem ili putem web aplikacije. Primjer nadzornog programa je Site Protector Console u slučaju ISS IDS-a.

2.3.1. Dekoder paketa

Svi paketi koji dolaze s mreže prvo ulaze u dekoder paketa. Zadaća ove komponente IDS-a je pripremanje paketa s različitih mrežnih sučelja prije daljnjeg prosljeđivanja pretprocesoru. Dekoder paketa omogućuje da se ostatak sustava koristi na jednak način bez obzira na vrstu mrežnog sučelja koja se koristi na ulazu u IDS, na primjer Ethernet, SLIP ili PPP. Pošto u IDS dolaze sirovi paketi (eng. *raw packets*) koji će biti različiti ovisno o vrsti sučelja s kojeg dolaze, nužno je ostalim dijelovima sustava osigurati jedinstveni format dekodiranog paketa. Bez ovakvog pristupa u projektiranju IDS-a, sustav bi bio vrlo ograničen jer bi se sve komponente kojima dekoder dalje prosljeđuje pakete morali dizajnirati za neku specifičnu vrstu paketa. Primjerice, morali bi zasebno biti implementirani pretprocesori za SLIP odnosno PPP sučelja. Umjesto toga, nakon dekodiranja sadržaja dalje se prosljeđuje i koristi vlastiti format paketa.

2.3.2. Pretprocesor

Pretprocesor je vrlo važna komponenta IDS-a koja može preurediti i mijenjati tekstualne ili binarne nizove znakova u paketu prije nego što ih sustav otkrivanja počne detaljnije analizirati. Neke vrste IDS-a su tako dizajnirane da čak i pretprocesor može stvarati sigurnosna upozorenja nakon što otkrije pogreške u pojedinim dijelovima paketa (primjerice Snort). Pogreške koje na paketima može otkriti pretprocesor su vrlo raznolike i u osnovi se svode na elementarnu provjeru zaglavlja u primljenim paketima. IDS može imati i više od jednog pretprocesora. U takvom modularnom slučaju svaki od pretprocesora obavlja uže specijalizirani dio posla koji bi inače obavljao cijeli pretprocesor. U različitim kombinacijama raspodijeljenih i centraliziranih IDS arhitektura obično svaki od senzora ima svoj vlastiti pretprocesor. Prednost IDS-ova koji sadrže više pretprocesora je puno lakši razvoj i nadogradnja sustava, te smanjenje zahtjeva nad pojedinim senzorima u slučaju da je arhitektura IDS-a takva da se pojedini pretprocesori izvode na različitim računalima.

Uloga pretprocesora može se najbolje opisati na jednostavnom primjeru otkrivanja napada na hipotetsku ranjivu mrežnu aplikaciju ili primjerice na web poslužitelj. Možemo pretpostaviti da će svaki bolji sustav za otkrivanje napada sadržavati pravilo koje će stvarati sigurnosno upozorenje ako se tijekom nadgledanja HTTP veze (eng. *session*) u jednom od paketa otkrije pokušaj pristupa važnoj sistemskoj datoteci poput administratorske IIS datoteke `scripts/iisadmin`. Ukoliko sustav otkrivanja doslovce uspoređuje staze do datoteka s onima navedenima u bazi potpisa, napadač može zaobići otkrivanje napada na nekoliko vrlo jednostavnih načina. Najjednostavniji takvi načini su korištenje zamjenskih ili *escape* znakova, korištenje apsolutnih staza, te korištenje oznaka "." za pristup tekućem direktoriju i ".." za pristup roditeljskom direktoriju. Tako na primjer svaka od sljedećih staza pokazuje na istu datoteku.

- `scripts/./iisadmin`
- `scripts/examples/../iisadmin`
- `scripts\iisadmin`
- `scripts/.\iisadmin`

Problem se može i dodatno zakomplicirati korištenjem Unicode (UTF-8 ili UTF-16) heksadecimalnih kodova u URI (eng. *uniform resource identifier*). Web poslužitelj će uvijek moći otkriti da se radi o stazi do datoteke *`scripts/iisadmin`*, međutim ukoliko IDS tumači staze doslovno onda neće moći otkriti ovakve vrste napada. Uloga pretprocesora je rješavanje upravo ovakvih i sličnih vrsta problema.

Pretprocesor se također koristi za defragmentaciju paketa. Kada se mrežom prenosi preveliki paket, isti se uobičajeno razdvaja na više dijelova – fragmenata. Na primjer, najveći neprekinuti niz podataka koji se može prenositi u Ethernet mrežama je 1500 okteta. Iznos ove veličine se nadzire pomoću MTU (eng. *maximum transfer unit*) vrijednosti mrežnog sučelja. Ukoliko se mrežom želi prenositi paket čija je veličina veća od vrijednosti MTU, paket prvo mora biti razdvojen na dijelove veličine manje ili jednake vrijednosti MTU. Nakon primanja svih dijelova paketa, primatelj podataka može ponovno sastaviti izvorni paket i očitati izvornu poruku. Fragmentacija paketa je jedan od uobičajenih načina kojim napadači pokušavaju zaobići otkrivanje napada i to ne samo kod IDS-a. Napadač u ovu svrhu može koristiti određena polja u zaglavljima paketa, poput

zastavice DF (eng. *don't fragment*). Prije nego što IDS počne provjeravati da li primljeni paket ispunjava neki od uvjeta iz baze potpisa, prvo ga mora ponovno sastaviti iz svih njegovih primljenih dijelova. U slučaju prethodnog primjera s pristupom važnim sistemskim datotekama, jedan dio URI-a se može nalaziti u jednom, a ostatak u drugom dijelu paketa. Kada bi se baza potpisa počela pretraživati za svaki dio paketa zasebno, sustav otkrivanja ne bi mogao otkriti napad.

Pretprocesor može riješiti sve prethodno navedene probleme. U većini danas dostupnih IDS-ova pretprocesor je iznimno važna komponenta i obavlja funkcije defragmentacije paketa, dekodiranja URI-a te praćenja i ponovnog sastavljanja (eng. *reassembly*) TCP veza bez kojih IDS ne bi mogao korektno raditi. Potrebno je naglasiti da je pojam pretprocesora prilično široko definiran, te da u praktičnim implementacijama sustava za otkrivanje napada s mreže modeli pretprocesora znaju znatno odstupati od modela opisanog u ovom poglavlju.

2.3.3. Sustav otkrivanja

Sustav otkrivanja je najvažniji dio IDS-a. Njegova uloga je provjeravanje da li primljeni paketi ispunjavaju neki od uvjeta iz baze potpisa i pravila tj. da li je sadržaj paketa u dozvoljenim granicama rada sustava u slučaju da je riječ o sustavu koji prepoznaje napade prepoznavanjem neuobičajenog ponašanja. Baza potpisa i pravila organizirana je u interne strukture podataka u obliku povezanih lista (eng. *linked lists*). Pravila su u listama grupirana tako da ako se pronađe određeni uzorak u ulaznim podacima onda se aktivira čitavi niz ispunjenih pravila. Time se izbjegava nepotrebno ponovno ispitivanje za isti uvjet, ali se i omogućava izrada složenijih nizova pravila. Primjerice, ako postoji pedeset pravila za otkrivanje niza "`/etc/shadow`" unutar paketa, mogu se grupirati unutar jedne liste. Umjesto da se paket pretražuje za svako od tih pedeset pravila, pretraživati će se samo jednom za prvi element liste, te ukoliko zadovoljava njegove uvjete automatski će se aktivirati svako drugo pravilo iz te liste.

Ako paket ispunjava bilo koje od pravila u bazi potpisa, poduzet će se odgovarajuća akcija koja može biti stvaranje sigurnosnog upozorenja, stvaranje zapisa u dnevniku ili oboje. U slučaju da paket nije zadovoljio niti jedno od pročitanih pravila, sustav će ga smatrati ispravnim korištenjem sustava te ga neće dalje obrađivati.

Sustav otkrivanja je vremenski najkritičnija i najosjetljivija komponenta IDS-a. Ovisno o tome koliko je brzo računalo na kojem se nalazi, te o broju pravila i potpisa, vremenski odziv na različite vrste paketa može jako varirati. Ovo može biti naročito važno ukoliko se od sustava zahtjeva da radi u stvarnome vremenu (eng. *real time*) gdje vrijeme prepoznavanja nikad ne smije prekoračiti unaprijed zadanu vremensku granicu. Problem posebno dolazi do izražaja kada sustav radi na mreži s iznimno velikom količinom prometa ili u slučajevima gdje je baza potpisa velika. Primjerice, ako je baza potpisa vrlo velika moguće je da će doći do zagušenja sustava čak i ako IDS radi na mreži s relativno malim prometom, npr. na 10 Mbit vezi. U takvim uvjetima sustav neće moći raditi u stvarnome vremenu, a moguće su i situacije gdje će neki paketi biti propušteni, a da uopće nisu bili pretraživani za tragove napada. U najgorem slučaju može se dogoditi i da se nestručno postavljen IDS u potpunosti zaguši i da prekine komunikaciju na pojedinim segmentima

mreže. Kao rješenje ovog problema moguće je primijeniti samo zapisivanje paketa, a tek naknadno napraviti pretraživanje njihovog sadržaja na drugom poslužitelju. Primjerice, ovakav pristup može se ostvariti kombinacijom programa Snort i Barnyard, gdje Snort stvara zapise, a Barnyard ih naknadno analizira.

Otkrivanje napada može se obavljati na raznim dijelovima paketa:

- zaglavlja na mrežnom sloju (IP, IPX, ICMP, itd.)
- zaglavlja na prijenosnom sloju (TCP, UDP, itd.)
- zaglavlja na aplikacijskom sloju (DNS, FTP, SNMP, SMTP, NNTP itd.)
- sadržaj paketa (eng. *payload*) na aplikacijskom sloju

Sustav otkrivanja može raditi na nekoliko načina. U prvom se načinu za paket koji zadovoljava jedan od uvjeta odmah stvara sigurnosno upozorenje i zapis u dnevniku, a paket se prestaje dalje obrađivati. Posljedica ovakvog načina rada je činjenica da će se stvarati samo jedno sigurnosno upozorenje za paket iako možda zadovoljava uvjete više pravila. Ovakva tehnika će doprinijeti rasterećenju sustava otkrivanja zato jer se neće morati pretraživati ostatak potpisa, međutim predstavlja ozbiljno ograničenje cjelokupnog sustava ukoliko bi paket zadovoljavao više pravila, a prvo pronađeno je najmanje važnosti. Djelomično rješenje ovog problema je i prethodno navedeno grupiranje pravila u povezane liste, međutim to samo umanjuje problem jer će još uvijek postojati grupe međusobno isključivih pravila koja mogu biti aktivirana istim paketom. Drugo rješenje uzima u obzir i ovakve probleme te svakome pravilu pridružuje i odgovarajuću težinu u ovisnosti o opasnosti koju paket može uzrokovati. Nakon što se pronađe prvo odgovarajuće pravilo za paket, otkrivanje se nastavlja dalje sve dok nisu pretražena sva pravila u bazi potpisa. Nakon što se pretraži cijela baza i sve liste pravila, može se dogoditi da paket zadovoljava i više pravila. Ovisno o tome kako je sustav postavljen, kao konačni rezultat pretrage se mogu prikazati sva zadovoljena pravila ili samo ona najvišeg prioriteta. Kao dodatna optimizacija moguće je da su pravila već unaprijed poredana po vrijednosti prioriteta, pa nema potrebe za pretraživanjem cijele liste ukoliko se dođe do zadovoljavajuće razine prioriteta. U tom slučaju prekida se s daljnjim pretraživanjem ostatka liste pravila.

2.3.4. Sustav za vođenje dnevnika i upozoravanje

Ovisno o tome što sustav otkrivanja pronađe unutar paketa, sustav za vođenje dnevnika i upozoravanje može stvoriti sigurnosno upozorenje te odgovarajući zapis u dnevniku. Samo fizičko spremanje zapisa u npr. datoteku ili bazu podataka obavljaju izlazni moduli. Dnevnik sadrži podatke primljene od sustava otkrivanja i obično se sastoji od sljedećih podataka o aktiviranom pravilu te paketu koji ga je aktivirao:

- vrijeme kada je pravilo aktivirano
- izvorišna i odredišna IP adresa paketa
- identifikacijski broj pravila
- opis pravila i referenca na detaljnije informacije
- sadržaj i zaglavlje paketa

- oznaka komponente koja je otkrila napad
- prioritet otkrivenog napada

Većina IDS sustava pruža dodatne opcije za nadziranje ovog dijela sustava čime se omogućuje filtriranje određenih vrsta upozorenja ili ograničavanje brzine njihovog zapisivanja. Mogućnost filtriranja i ograničavanja broja zapisa kod stvaranja upozorenja je važna jer će u većini slučajeva IDS stvarati veliki broj lažnih upozorenja iste vrste, te bez razloga puniti dnevnik i stvarati dodatni bespotrebni mrežni promet. Filtriranjem i ograničavanjem može se jednostavno ograničiti broj zapisivanja određene vrste pravila u jedinici vremena ili ga u potpunosti zabraniti.

2.3.5. Izlazni moduli

Izlazni moduli su zaduženi za formatiranje podataka primljenih od sustava za vođenje dnevnika i upozoravanje. Ovaj dio IDS-a određuje format zapisa, primjerice u datoteci ili bazi podataka, te definira metodu dojave sigurnosnog upozorenja. Izlazni moduli su vrlo važna komponenta IDS-a jer omogućavaju rad IDS-a na raznorodnim okolinama. Na primjer, sustav otkrivanja i ostale kritične komponente se mogu nalaziti na Linux operacijskom sustavu, dok se zahvaljujući izlaznom modulu koji omogućava stvaranje zapisa pomoću SMB poruka dnevnici mogu čuvati na računalima koja koriste drukčiji operacijski sustav poput Microsoft Windowsa. Dodatni razlog za različite vrste izlaznih formata dnevničkih zapisa je činjenica da se u većini slučajeva datoteke zapisa ne čitaju direktno nego se dalje analiziraju s različitim dodatnim alatima. Korištenjem dodatnih alata za analizu moguće je dobiti detaljniji uvid u najčešće napade i njihovu strukturu.

Neki od uobičajenih i najčešćih korištenih formata datoteka dnevnika te načina dojavljivanja su sljedeći:

- tekstualne datoteke
- XML (eng. *Extensible Markup Language*) datoteke
- postavljanje SNMP varijabli
- korištenje usluga operacijskog sustava (npr. *syslog*)
- korištenje sustava za upravljanje relacijskim bazama podataka (RDBMS)
- slanje SMB (eng. *Service Message Block*) poruka
- slanje upozorenja elektroničkom poštom

2.4. Položaj u različitim mrežnim topologijama

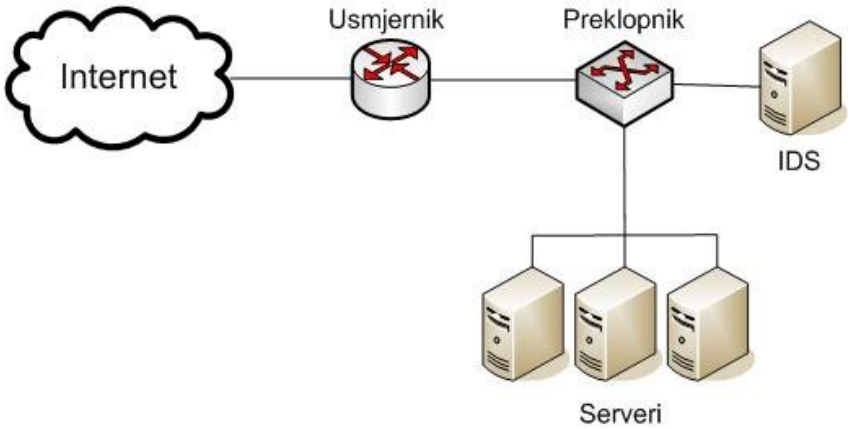
2.4.1. Smještaj IDS-a

Ovisno o mrežnoj topologiji na koju se postavlja, IDS senzor može biti smješten na jedno ili više mjesta u mreži koja se nadzire. Smještaj također ovisi o prirodi napada koju je potrebno otkrivati: napade s lokalne mreže (*unutrašnje napade*), napade izvan lokalne mreže (*vanjske napade*) ili oboje. Na primjer, ako želimo otkrivati samo vanjske napade, a mreža se sastoji od samo jednog usmjernika (eng. *router*) preko kojega lokalna mreža s nekoliko segmenata ima pristup Internetu, najbolje je IDS postaviti odmah iza usmjernika jer će tako sav promet koji dolazi s Interneta u lokalnu mrežu morati proći kroz usmjernik pa samim time i kroz IDS senzor. Ako mreža ima više izlaza na Internet, tada se IDS mora nalaziti na svakoj izlaznoj točki iz mreže. Ako želimo otkrivati i unutrašnje napade, tada je IDS senzore potrebno postaviti na svaki zasebni mrežni segment kako bi senzori imali pristup cjelokupnom prometu koji prolazi tom mrežom. U većini slučajeva nema potrebe za nadziranjem svakog pojedinačnog segmenta nego se nadziru samo pojedini povjerljivi i kritični dijelovi mreže, primjerice segment na kojem se nalazi sigurnosna stijena ili neki važniji aplikacijski poslužitelj, baza podataka itd. U pravilu se IDS senzor u lokalnoj mreži postavlja iza svakog usmjernika i sigurnosne zaštitne stijene. Mjesta koja se obično izbjegavaju su zalihosni (redundantni) dijelovi mreže poput paralelnih mrežnih veza, nakupina računala (eng. *clusters*), višestrukih preklopnika, sustava s *failover* rješenjima protiv ispada pojedinih mrežnih kartica itd.

Posebnu pažnju kod postavljanja IDS senzora i dimenzioniranja mreže treba obratiti na način na koji će se obrađivati prikupljeni podaci. Ovi problemi posebice dolaze do izražaja u složenijim konfiguracijama IDS-a gdje je potrebno postaviti više od jednog IDS senzora, te kada isti moraju međusobno komunicirati. U takvim slučajevima najčešće je potrebno odabrati rješenje koje će biti kompromis između potrebne procesorske snage te količine prometa koja će se izmjenjivati između pojedinih IDS senzora. Podaci iz pojedinih senzora mogu se obrađivati *raspodijeljeno* ili *centralizirano.* Kod raspodijeljenog pristupa podaci se obrađuju u onim senzorima u kojima su i prikupljeni, dok se kod centraliziranog pristupa podaci sakupljeni u senzorima šalju u centralni senzor ili *analizator* gdje će biti podrobnije ispitani. Prednost centraliziranog pristupa projektiranju IDS-a je lakše upravljanje cjelokupnim sustavom, međutim takvo rješenje znatno povećava količinu mrežnog prometa koji se razmjenjuje između pojedinih senzora i centralnog senzora tj. konzole, te zahtjeva znatno veću procesorsku moć centralnog senzora koji će morati obrađivati događaje iz cijelog sustava. Raspodijeljeni je pristup znatno teže realizirati i održavati, ali se zato smanjuje količina mrežnog prometa koji se izmjenjuje između pojedinih senzora u sustavu, a za razliku od centraliziranog sustava potrebna je puno manja procesorska snaga za obradu prikupljenih podataka. Dodatna činjenica koja ide u prilog raspodijeljenom pristupu u oblikovanju IDS je bolja otpornost na ispade u sustavu. Pošto su pojedine komponente u raspodijeljenom sustavu potpuno samostalne i ne ovise o niti jednom drugom elementu sustava, ukoliko jedan od senzora u sustavu prestane s radom, ostatak sustava može i dalje ispravno raditi. Centralizirani sustav nije tako fleksibilan i otporan je samo na ispade senzorskih računala, a ukoliko se pokvari centralni senzor ili nadzorna konzola, tada će prestati ispravno funkcionirati i ostatak sustava. U praksi se najčešće javlja kombinacija oba prethodno navedena pristupa pri čemu raspodijeljeni

senzori prikupljaju podatke i na njima obavljaju neke osnovne akcije (npr. pretprocesiranje mrežnih paketa), te ih zatim prosljeđuju jednom ili više centralnih senzora koji će ih podrobnije analizirati te poduzeti potrebne akcije ovisno o njihovom sadržaju. Svim komponentama u takvim sustavima može se upravljati s jednog mjesta putem posebne nadzorne konzole. Nadzorna konzola je specijalizirani senzor koji uz mogućnosti otkrivanja napada ima i mogućnost upravljanja drugim senzorima, primjerice mijenjanje njihovih postavki za vrijeme rada sustava ili pokretanje i gašenje senzora. Moguće je da nadzorna konzola ima isključivo upravljačke mogućnosti te da ne može otkrivati napade.

Kao što je već navedeno, pravilan smještaj IDS-a je od ključne važnosti za njegovo pravilno djelovanje. Odabirom ispravne pozicije možemo djelovati na niz aspekata cijelog sustava, počevši od količine prometa koji će biti potrebno analizirati, pa sve do procesorske snage koja će biti potrebna za analizu dobivenih podataka. Tipični smještaj IDS-a u jednostavnoj lokalnoj mreži prikazuje slika 2.2. U slučaju da se zahtjeva da mreža mora sadržavati i demilitarizirano područje, također je potrebno smjestiti jedan od IDS senzora i u taj segment mreže, iako bi u pravilu postavke za stvaranje sigurnosnih upozorenja u tom području trebale biti puno blaže od onih u privatnim dijelovima mreže, što zbog same prirode i namjene tog područja u kojem se obično ne nalaze ključni sustavi za poslovanje nego sustavi čiji ispad neće uzrokovati pad cijele mreže, te zbog očekivanoga većeg broja prepoznatih napada nego u unutrašnjoj mreži. U protivnome je moguće zagušenje IDS senzora u demilitariziranom području zbog prevelikoga broja otkrivenih potencijalnih napada.



Slika 2.2: Tipičan smještaj IDS-a

2.4.2. Rješavanje problema s preklopnicima

Poseban problem u smještaju IDS-a i pojedinih senzora predstavlja mrežna oprema koja na bilo koji način blokira neku vrstu prometa. Primjer takve mrežne opreme je preklopnik (eng, *switch*). Problem se može pojaviti u slučajevima poput onoga koji prikazuje slika 2.2. Preklopnik će pakete koje dobije na jedan od izlaza proslijediti na točno onaj izlaz koji je priključen na onaj segment mreže na kojemu se nalazi ciljna adresa (preklopnik radi sa adresama na podatkovnom sloju, tj. s MAC adresama). Naravno, izuzetak od ovoga je početna faza rada uređaja u kojemu još nisu popunjene sve tablice s podacima o

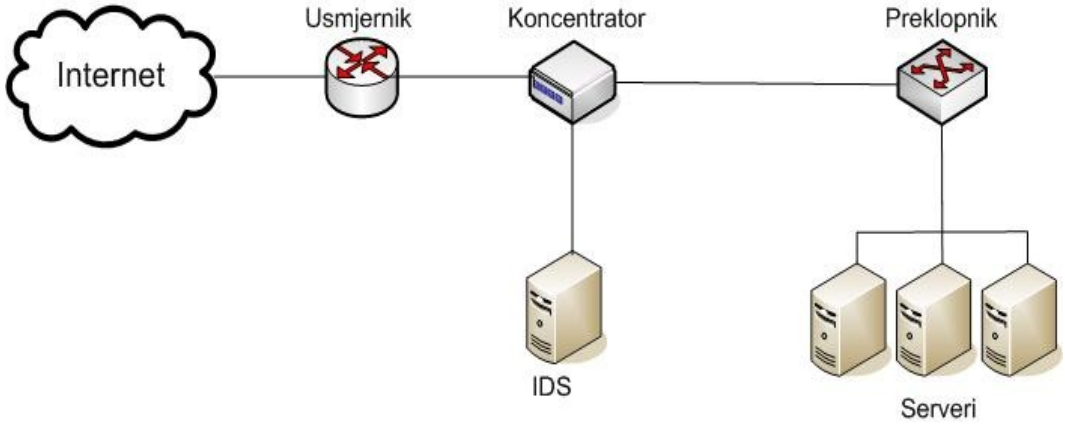
usmjeravanju prometa. Ovo predstavlja ozbiljno ograničenje u radu IDS-a jer je u tom slučaju potrebno postavljati senzore na svaki od izlaza preklopnika, tj. na svako zasebno područje vidljivosti što bitno uvećava troškove i vrijeme potrebno za postavljanje, ali i održavanje takvog rješenja.

Većina preklopnika (npr. tvrtke Cisco) su tako dizajnirani i mogu raditi u takvome načinu rada da se TX/RX (eng. *transmit* i *receive*) linije s jednog ili više izlaza mogu zrcaliti ili *miroririrati* na drugi izlaz preklopnika. Takav izlaz na koji se zrcali se obično naziva SPAN (eng. *Switch Port Analyser*) ili zrcalni izlaz (eng. *mirrored port*). Ukoliko se senzor priključi na takav izlaz, a preklopnik je postavljen tako da zrcali izlaz koji je priključen prema usmjerniku na SPAN izlaz, sav promet koji prolazi od usmjernika k poslužiteljima kroz preklopnik dolaziti će i do IDS senzora, ne ometajući pri tome normalnu komunikaciju prema poslužiteljima. Na taj način se izbjegava potreba za postavljanjem dodatnih senzora u svaki pojedini segment mreže na koji je spojen preklopnik. Korištenje SPAN izlaza je vrlo jednostavno i ne zahtjeva nikakvo dodatno mijenjanje postojeće mrežne topologije, uz uvjet da ga uređaj podržava. Priključivanjem IDS senzora na SPAN izlaz preklopnika nije potrebno mijenjati konfiguracije ostalih mrežnih komponenti poput tablica prosljeđivanja u usmjernicima ili pravila u sigurnosnim stijenama. Međutim, uz sve dobre osobine, korištenje SPAN izlaza ima i niz nedostataka. Na jednom preklopniku se može nalaziti samo jedan SPAN izlaz. Ukoliko je potrebno nadzirati više od jednog izlaza preklopnika tada se na SPAN izlaz umjesto samo jednog mora zrcaliti promet sa čitavog raspona izlaza. Ovakvo zrcaljenje prometa sa više izlaza nije poželjno jer može vrlo brzo zakrčiti i preopteretiti SPAN izlaz i znatno smanjiti radna svojstva (performanse) rada preklopnika. Problem je pogotovo izražen ako se nadzire promet u dvosmjernoj vezi (eng. *full duplex link*). Nadalje, bez dodatnih promjena u konfiguraciji IDS-a, ovakav način rada preklopnika omogućava napadaču napad direktno na IDS senzor spojen na SPAN izlaz, osim ako je zaštićen radom u prikrivenom (eng. *stealth*) načinu rada kako je opisano u poglavlju o zaštiti IDS-a. Još jedan nedostatak ovakvog rješenja je i činjenica da će preklopnik odbacivati pakete za koje izračunato zaštitno CRC polje ne odgovara onome u primljenom paketu. IDS inače analizira i ovakve pakete jer napadači koriste razne tehnike u kojima namjerno stvaraju pakete sa krivim CRC poljima kako bi zaobišli mehanizme zaštite i otkrivanja ili otkrili njihovu prisutnost. Ipak, ovakve pakete je moguće vidjeti na samom preklopniku, s podacima o adresi napadača, vremenu napada i broju takvih paketa.

Drugo rješenje problema s preklopnicima je korištenje dodatne specijalizirane mrežne opreme. Jedno od najdostupnijih i najjeftinijih rješenja je upotreba koncentratora (eng. *hub*). Koncentrator se koristi u paričnim mrežama i radi na takav način da će sve pakete koje dobije na jedan od svojih izlaza proslijediti na sve ostale izlaze, tj služi kao obnavljač signala sa više izlaza (eng. *multiport repeater*). Ovakav način rada koncentratora može se iskoristiti za slanje preslike prometa prema IDS-u pri čemu je potrebno napraviti takvu topologiju mreže u kojoj će se koncentrator nalaziti između usmjernika i preklopnika, a IDS se priključi na dodatni izlaz koncentratora kao što prikazuje slika 2.3.

U ovom slučaju sav promet koji prolazi od usmjernika prema preklopniku, zahvaljujući koncentratoru biti će proslijeđen i preklopniku i IDS senzoru. Potrebno je i napomenuti da je ovakva topologija mreže pogodna samo za otkrivanje vanjskih napada, jer IDS može vidjeti samo promet koji dolazi s Interneta. Zbog načina rada preklopnika IDS neće vidjeti promet koji međusobno razmjenjuju poslužitelji koji se nalaze iza preklopnika. U slučaju

da je potrebno nadzirati promet i na tom segmentu mreže, potrebno je dodati još jedan IDS senzor na već prije navedeni SPAN izlaz preklopnika. Korištenje koncentratora za rješavanje problema sa preklopnicima ima niz prednosti. Koncentratori s četiri izlaza koji se koriste u ovakvim konfiguracijama su relativno jeftini uređaji i vrlo se jednostavno postavljaju u već postojeću mrežnu topologiju. Postavljanjem koncentratora također nije potrebno mijenjati postavke usmjernika i sigurnosnih stijena. Međutim i korištenje koncentratora ima svoje nedostatke. Ako je veza između usmjernika i preklopnika dvosmjerna, zbog učestalih kolizija paketa propusnost veze može postati bitno smanjena što će znatno smanjiti radna svojstva cijele mreže. Kolizije će naročito doći do izražaja ako se uz usmjernik, preklopnik i IDS senzor na četvrti izlaz koncentratora spoji i nadzorna konzola IDS-a što će bitno utjecati na tok podataka između usmjernika i koncentratora. U zadnje vrijeme koncentratori se sve manje koriste u ovu namjenu, te se češće koriste specijalizirani mrežni uređaji konstruirani upravo za ovakve namjene. Primjer takvog uređaja je mrežna priključnica (eng. *network tap*).



Slika 2.3: Korištenje koncentratora u mrežama sa preklopnicima

2.4.3. Mrežna priključnica

Za rješavanje svih prethodno navedenih problema kod rada IDS-a s preklopnicima danas se najčešće koriste posebni mrežni uređaji dizajnirani upravo tako da zaobiđu sve nedostatke prethodno navedenih pristupa. Mrežna priključnica (eng. *network tap*) je uređaj namijenjen preslikavanju prometa u Ethernet, 802.11, FDDI i ATM mrežama. Funkcionalnost uređaja odvija se na fizičkom i podatkovnom sloju, a dodatna je pogodnost što može raditi i na žičanim (bakrenim) i na optičkim medijima. Na tržištu su dostupne dvije vrste mrežnih priključnica: *aktivne* i *pasivne*. Pasivne mrežne priključnice (eng. *passive ethernet tap*) ne koriste nikakvo dodatno napajanje pa mogu dovesti do male degradacije signala na mrežnom kabelu na kojem se koriste. Aktivne mrežne priključnice (eng. *active ethernet tap*) koriste dodatni izvor napajanja. Priključnica se obično koristi za nadziranje i ispitivanje prometa na najkritičnijim i najzahtjevnijim segmentima mreže iz razloga što kvarovi i gubitak napajanja na uređaju ni na koji način neće utjecati na povezanost i radna svojstva ostatka mreže. Ponegdje se u literaturi izraz *tap* iz engleskog izvornika tumači kao kratica od engleskog izraza *Test Access Port*.

Mrežne priključnice su uređaji s tri ili više izlaza. Na primjer, mrežna priključnica s četiri izlaza uobičajeno izlaze ima označene sa *A*, *B*, *A tap* i *B tap*. Izlazi A i B su podatkovni izlazi i priključuju se izravno na segment mreže koji je potrebno nadzirati, a *tap* izlazi se priključuju na IDS-ove. Tap izlazi služe za zrcaljenje podatkovnih izlaza, tako da tap izlazi A i B zrcale promet koji prolazi podatkovnim izlazima. Svaki tap izlaz daje identičan promet. Mrežna priključnica se u mrežu postavlja na isto mjesto na koje se postavlja i koncentrator, kao što je prethodno opisano i kako prikazuje slika 2.3.

Prednosti korištenja mrežnih priključnica nad koncentratorom i SPAN izlazom preklopnika su mnogobrojne. Priključnica je tolerantna na ispade u napajanju jer su podatkovni izlazi unutar priključnice spojeni žicom bez ikakvih drugih pasivnih ili aktivnih električkih elemenata pa stoga i ne ovise o vanjskom izvoru napajanja. Moguća je jedino mala degradacija signala u slučaju korištenja pasivnih izvedbi ovog uređaja. Priključnica ne utječe na protočnost podataka između usmjernika i preklopnika, ne zahtjeva promjene u konfiguracijama ostalih mrežnih uređaja poput sigurnosnih stijena i usmjernika, te ne zahtjeva promjenu arhitekture mreže u cjelini. Nadalje, uređaj ne odbacuje pakete s lošim CRC poljem pa će sav promet dolaziti nepromijenjen do IDS senzora koji će moći provesti kompletnu analizu dobivenoga prometa. Konačno, priključnica ne dozvoljava izravno povezivanje napadača na IDS jer je njezino djelovanje na mreži potpuno nevidljivo pa pridonosi i cjelokupnoj sigurnosti sustava. Svi prethodno navedeni argumenti su glavni razlog zašto su danas mrežne priključnice najčešći korišteni i najprihvatljiviji način postavljanja (eng. *deploying*) IDS senzora u kompleksnija mrežna okruženja. Dva danas najčešće korištena modela mrežnih priključnica su proizvodi tvrtki *NetOptics* [13] i *Finisar* [14].

Nedostatak korištenja priključnica je već spomenuti problem s gubljenjem signala kod pasivnih izvedbi, te još uvije prilično visoka cijena aktivnih izvedbi. Dodatna je činjenica i da se bez dodatnih modifikacija većina ovakvih uređaja ne može koristiti za nadzor prometa u oba smjera. U slučaju nadzora prometa u oba smjera potrebno je primijeniti udruživanje više kanala u jedan (eng. *channel bonding*) na IDS senzorima koji se priključuju na takve uređaje. Iako se posljednjih godina proizvode ovakvi uređaju i za primjenu u dvosmjernim (eng. *full duplex)* segmentima, zbog cijene se još uvijek vrlo rijetko koriste. Nadalje, IDS koji je priključen na priključnicu mora biti postavljen za rad u prikrivenom načinu rada, kao što je opisano u poglavlju o zaštiti sustava.

2.5. Problemi i nedostaci IDS-ova

Uz brojne prednosti, postoje i određeni nedostaci korištenja IDS-ova. Pregledavanjem sadržaja mrežnih paketa često se dozvoljeni promet krivo karakterizira kao nedozvoljeni, odnosno nedozvoljeni kao dozvoljeni. Do ove pojave dolazi uslijed previše općenitih definicija potpisa pojedinih vrsta napada u slučaju da IDS radi prepoznavanje potpisa. Do grešaka može doći kod prepoznavanja neuobičajenog ponašanja, na primjer u dinamičkim okolinama gdje se vrta prometa mijenja vrlo često. IDS-ovi uglavnom ne mogu zaustaviti ili usporiti aktivne mrežne napade, nego se za tu namjenu koriste sustavi za sprječavanje mrežnih napada. Zbog mogućih dojava o lažnim napadima na sustav, nije niti je poželjno

da IDS-ovi prekidaju uspostavljene veze jer bi to rezultiralo velikim brojem neopravdanih prekida u radu. Posljedica činjenice da IDS-ovi ne mogu prekinuti napade je stvaranje onoliko sigurnosnih upozorenja koliko ima dnevničkih zapisa o samim napadima. Pošto se ponekad i ispravni promet neopravdano prepoznaje kao napad, količina upozorenja za sistemske administratore se time dodatno povećava.

Sljedeći problem je vremenski raskorak između napada i otkrivanja napada. Iz razloga što IDS mora analizirati veliki broj zapisa te moguće potrebe za naknadnom provjerom dojave o napadu od strane sistemskog administratora, IDS-ovi pružaju spor odgovor na napade na računalne i mrežne resurse. Još jedan od nedostataka IDS-ova je činjenica da ne mogu otkriti nove vrste napada, nego mogu ustanoviti samo već postojeće. Kako ovakvi sustavi uglavnom uspoređuju sadržaje dnevničkih zapisa i mrežnih paketa s definiranim potpisima za napade, novi napadi nisu sadržani u bazi potpisa pa ih IDS niti ne prepoznaje. Otkrivanje nepoznatih vrsta napada moguća je samo ukoliko je IDS zasnovan ili kombiniran sa metodom prepoznavanja neuobičajenog ponašanja, međutim i takav pristup ima svojih nedostataka.

Na kraju, u zadnje vrijeme je prisutna sve veća tendencija enkripcije prometa koji prolazi kroz mrežu. Sve više mrežnih usluga razmjenjuje poruke preko sigurnih kanala i virtualnih privatnih mreža (eng. *virtual private network,* VPN). IDS-ovi imaju velikih problema u nadziranju veza zaštićenih IPSec rješenjima ili ostvarenih putem npr. SSH tuneliranja. Nemogućnost prepoznavanja napada u kriptiranim vezama vidljiva je i na primjeru HTTPS-a, odnosno HTTP veza koje se odvijaju u kriptiranom obliku putem SSL protokola. Problem nadziranja HTTPS-a je naročito važan kada se uzme u obzir sve veći broj različitih napada na web aplikacije.

Zbog spomenutih nedostataka jasno je da je nužno razinu sigurnosti pomaknuti s otkrivanja zlonamjernih aktivnosti na njihovo sprječavanje, pa je tako došlo do razvoja sustava za sprečavanje napada sa mreže (eng. *Intrusion Prevention Systems – IPS*). IPS-ovi omogućavaju preventivno djelovanje glede računalnih napada i izbjegavanje nastajanja štete koja bi bila prouzročena tim napadima.

2.6. Zaštita IDS-a

IDS je poput svake ostale komponente na mreži podložan raznim vrstama napada. Uobičajene su dvije mjere zaštite IDS-a, rad na prikrivenom mrežnom sučelju i rad na mrežnom sučelju bez adrese.

Sučelja na kojima radi IDS obično su postavljena i za primanje i za slanje podataka, te rade u takvome načinu rada da primaju i pakete koji nisu namijenjeni direktno njima (eng. *promiscuous mode*). Prikriveno sučelje (eng. *stealth interface*) je mrežno sučelje na kojem IDS senzor može samo primati podatke, ali ih ne može slati nazad. Takvo sučelje se može ostvariti na dva načina. Prvi način je korištenje mrežne priključnice koja sama po sebi ne dozvoljava vraćanje prometa, a drugi je korištenje posebne vrste UTP kabela (eng. *sniffing cable*). Ova vrsta kabela se može napraviti od običnog UTP kabela na nekoliko načina. Najjednostavnije je na jednom kraju kabela kratko spojiti linije 1 i 2, a linije 3 i 6 se spoje s istim tim linijama na drugom kraju kabela. Umjesto da se linije 1 i 2 kratko spoje, između njih se može umetnuti kondenzator koji će služiti kao visoko propusni filtar. Iznos

kondenzatora se može izračunati iz izraza $f = 1 / 2\pi RC$ i iznosi 150 pF za 10 megabitni te 15 pF za 100 megabitni Ethernet. Postoje i naprednije izvedbe prikrivenog sučelja poput takozvanoga UTP Y-kabela koji je detaljnije opisan (zajedno s uputama za implementaciju) na odgovarajućoj web adresi [21].

Druga metoda zaštite IDS-a je rad na sučelju bez dodijeljene IP adrese. Ako IDS senzor nema dodijeljenu IP adresu tada se na njega ne može pristupiti. Postavljanje ovakvog sučelja je vrlo jednostavno i svodi se na podizanje sučelja bez adrese i onemogućavanje protokola za automatsko dodjeljivanje adrese poput DHCP-a. Senzor obično ima dva sučelja od kojih je jedno, ono prema Internetu, bez IP adrese ili ostvareno putem prikrivenog sučelja, dok je drugo prema lokalnoj mreži ili nadzornoj konzoli, sa dodijeljenom IP adresom.

Još jedna popularna metoda zaštite su i transparentni IDS premosnici (eng. *transparent bridges*). Takvi sustavi sadrže dva mrežna sučelja i mogu se postaviti na bilo koje mjesto u lokalnoj mreži što ne zahtjeva dodatne promijene u mrežnoj topologiji ili već postojećim konfiguracijama usmjernika. Transparentni IDS premosnici kombiniraju korištenje Ethernet premosnika s IDS-om i sigurnosnom stijenom što ima niz prednosti nad npr. implementacijom sigurnosne stijene na usmjerniku. Pošto se u takvim konfiguracijama sigurnosna stijena i IDS nalaze na premosniku, a ne na usmjerniku, to ih čini nevidljivima na mrežnom sloju i stoga ih štiti od napada. Svaki paket prolazi kroz premosnik sve dok IDS ne pronađe potpis napada.

2.7. Standardizacija

Jedan od prvih pokušaja da se standardiziraju protokoli i aplikacijska programska sučelja (eng. *Application Programming Interface* – API) koja se koriste u IDS sustavima rezultirao je stvaranjem zajedničkog IDS okvira poznatog kao CIDF (eng. *Common Intrusion Detection Framework*). Glavni cilj CIDF-a je pojednostavniti dijeljenje informacija i resursa između različitih IDS sustava te omogućiti da se njihove pojedine komponente mogu ponovno iskoristiti u drugim sustavima. Sam CIDF definiran je u četiri Internet nacrta (eng. *draft*):

- *"The Common Intrusion Detection Framework Architecture"*
- *"Communication in the Common Intrusion Detection Framework"*
- *"A Common Intrusion Specification Language (CISL)"*
- *"CIDF APIs: Their Care and Feeding*"

Prvi nacrt je najvažniji i opisuje osnovne pojmove u CIDF terminologiji te daje detaljni opis CIDF arhitekture. Ostali dokumenti definiraju protokole, format poruka i programska sučelja putem kojih pojedine komponente iz CIDF arhitekture IDS-a mogu međusobno komunicirati na siguran i učinkovit način te obavljati međusobne provjere autentičnosti i raspoloživosti. Svi navedeni dokumenti dostupni su na web sjedištu CIDF-a [15].

CIDF opisuje IDS kao sustav koji sačinjavaju četiri diskretne komponente. Komponente sustava su sljedeće:

- generatori događaja (eng. *event generators, E – box*)
- analizatori događaja (eng. *event analyzers, A – box*)
- baze podataka o događajima (eng. *event databases, D – box*)
- jedinice za odgovaranje na događaje (eng. *response units, R – box*)

Sve četiri komponente međusobno komuniciraju razmjenom poruka (eng. *message passing*) standardiziranog formata i korištenjem jezika CISL (eng. *Common Intrusion Specification Langague*). S vremenom je CISL koji je prilično kompliciran zamijenjen jednostavnijim oblikom poruka za razmjenu podataka poznatim pod imenom IDMEF (eng. *Intrusion Detection Message Exchange Format*). Uloga generatora događaja je prikupljati podatke s mreže i slati poruke o događajima ostalim komponentama sustava. Analizatori događaja su komponente na koje se uobičajeno misli kada se govori o IDS-ovima. Njihova uloga je primati poruke od ostalih komponenti, analizirati ih te vratiti poruke koje predstavljaju sažetak analize ulaznih poruka. Baza podataka o događajima i jedinice za odgovaranje na događaje imaju sličnu namjenu kao i baza potpisa i izlazni moduli u uobičajenim IDS arhitekturama.

Neke od ideja koje su proizašle iz CIDF projekta potakle su nastanak IETF radne grupe (eng. *Internet Engineering Task Force Working Group*) koja je dobila naziv *Intrusion Detection Working Group*. Glavni cilj ove radne grupe je predstaviti ideje CIDF projekta široj javnosti.

Svi pokušaji standardizacije nastali u okviru IETF-ovih radnih grupa i CIDF projekta datiraju još iz 1998. godine i nisu rezultirali nekim pretjeranim uspjehom, a pojedini dijelovi standarda koriste se svega u nekoliko dostupnih IDS-ova (npr. Prelude), iako se neke ideje iz tih nacrta i danas koriste u promijenjenom obliku. Umjesto primjene ovakvih pravnih i službenih standarda u većini današnjih IDS-ova prisutni su neformalni (*de facto*) standardi proizašli iz nekih od najpopularnijih IDS programa.

3. Sustav za otkrivanje mrežnih napada Snort

3.1. Snort IDS

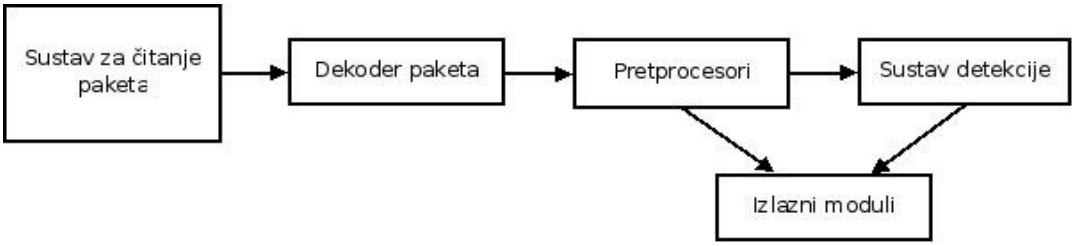
Snort je trenutno najpopularniji i mogućnostima najbogatiji besplatni sustav za prepoznavanje napada sa mreže. Program posjeduje mogućnosti analize prometa u realnom vremenu, kao i bilježenje paketa u IP mrežama. Dostupne su funkcije poput analize protokola na raznim slojevima te pretraživanja i provjere podudaranja sadržaja paketa s unaprijed definiranim potpisima napada. Koristeći ove funkcije mogu se izvoditi operacije prepoznavanja standardnih napada poput skeniranja i probi, prekoračenja međuspremnika (eng. *buffer overflow*), CGI napada, SMB probi, pokušaja prepoznavanja tipa operacijskog sustava (eng. OS *fingerprint attemp*) i niz drugih vrsta napada.

Program je prvi puta objavljen 1999. godine. Originalni autor Snorta je Martin Roesch, a razvoj i održavanje programa preuzela je tvrtka *Sourcefire* (osnivač i voditelj razvojnog tima tvrtke je već navedeni autor). U trenutno dostupnoj verziji Snort je prilagođen za rada na nizu različitih platformi (Linux, Microsoft Windows, SunOS/Solaris, FreeBSD, OpenBSD, NetBSD, IRIX, HP-UX). Iako je program u potpunosti besplatan, tvrtka Sourcefire pruža mogućnost nabave i komercijalnih verzija posebno razvijenih za razne velike okoline (eng. e*nterprise environment*), koje uključuju i tehničku podršku i mogu se nabaviti uz dodatnu specijaliziranu programsku i sklopovsku opremu.

Ukratko opisano, glavni argumenti za korištenje Snorta ispred ostalih besplatnih IDS programa su:

- Snort pruža mogućnost podešavanja svake pojedine komponente sustava
- Snort je slobodan i besplatan
- Snort je de facto standard IDS na LAMP platformi
- Snort radi na gotovo svakom danas poznatijem operacijskom sustavu
- Snort se kontinuirano poboljšava i nadograđuje
- mogućnost pisanja vlastitih pravila

Tehnički gledano, Snort može izvoditi većinu akcija kao i popularni komercijalni IDS sustavi. Centralnu ulogu u Snortu imaju pretprocesori i sustav otkrivanja. Korištenjem raznih pretprocesora Snort može izvoditi defragmentaciju IP paketa, ponovno sastavljanje TCP tokova podataka (eng. *TCP stream reassembly*) te provoditi analizu stanja raznih mrežnih protokola. Shematski prikaz glavnih komponenti Snorta kao i tok izmjene informacija između njih prikazuje slika 3.1.



Slika 3.1: Arhitektura Snorta

Zadaća sustava za čitanje paketa (eng. *packet capture subsystem*) je preuzimanje paketa s mreže i njihovo prosljeđivanje u dekoder paketa. Standardno se u Snortu za ovu namjenu koristi biblioteka *libpcap* (u inačici za operacijski sustav Windows koristi se biblioteka *winpcap*), osim u slučaju korištenja inline načina rada koji će biti opisan kasnije u ovome poglavlju. Dekoder paketa zadužen je za stvaranje unificiranog zapisa paketa neovisno o vrsti mrežnog sučelja s kojeg je paket primljen. Pretprocesori i sustav otkrivanja su centralni elementi Snorta i upravo su oni zaduženi za otkrivanje napada te upozoravanje korisnika. Rezultati otkrivanja napada spremaju se u datoteke dnevnika ili se dojavljuju na neki drugi način korištenjem raznih izlaznih modula.

Za opisivanje potpisa napada Snort koristi vrlo prilagodljiv jezik zasnovan na pravilima, a sam program dizajniran je tako da omogućava laku izmjenu podataka s drugim programima (npr. *SnortSnarf*, *squil*, *OSSIM*, *BASE* itd.). Ovo je vrlo važna činjenica jer je Snort sam po sebi komandno linijski alat i nema ugrađene mogućnosti vizualizacije te statističke analize dobivenih podataka. Program se uz određene prepravke i zakrpe (eng. *patch*) može koristiti zajedno s antivirusnim programima otvorenog koda (npr. *ClamAV*) i to je vrlo česta praksa u postavljanju Snorta u kompleksnije okoline. Tvrtka Sourcefire je krajem 2007. godine i kupila ClamAV projekt. Jednostavnost i velika prilagodljivost jezika za opis pravila omogućuju da svatko napiše vlastita prilagođena pravila za otkrivanje novih vrsta napada. Postoje i posebne kolekcije tj. biblioteke tako napisanih pravila održavanih od strane zajednice otvorenog koda, primjerice *Bleeding Edge Threats,* koje obuhvaćaju pravila za prepoznavanje gotovo svakog danas poznatog načina napada. Mogućnosti jezika za opis pravila te kratki pregled programa koji mogu surađivati sa Snortom opisani su u poglavlju o podešavanju Snorta.

O važnosti samog programa govori i činjenica da je Snort de facto standard na svakoj većoj implementaciji nekog projekta temeljenog na LAMP (Linux, Apache, MySQL, PHP) platformu, te da je nezaobilazni dio skupine najčešće korištenih programa otvorenog koda (eng. *open source programs suite*).

3.2. Načini rada

3.2.1. Sniffer način rada

Najjednostavniji način rada Snorta je *sniffer* ili *packet dump*. U sniffer načinu rada program jednostavno čita sve primljene pakete i prikazuje njihova zaglavlja ili kompletni sadržaj na standardnom izlazu, uobičajeno na ekranu. U nastavku rada podrazumijeva se korištenje Snorta verzije 2.8.

Za pokretanje Snorta u sniffer načinu rada, dovoljno ga je pokrenuti iz komandne linije koristeći sljedeću naredbu:

```
# snort -v
```

Pokrenut na ovaj način, program će ispisivati samo zaglavlja TCP, UDP, IP i ICMP paketa, te osnovne informacije o nekim drugim protokolima. Opcija *-v* (eng. *verbose*) koristi se za naznačivanje da je potrebno raditi ispis na standardni izlaz. Ukoliko je potrebno ispisivati i sadržaj pročitanih paketa, potrebno je dodati opciju *-d*. Dodatni detalji koji uključuju i zaglavlja sa podatkovnog sloja mogu se dobiti korištenjem opcije *-e*. Opcije programa se mogu navoditi pojedinačno, ali i zajedno. Poredak pojedinih opcija kod poziva programa nije bitan. Sljedeći primjeri poziva programa djeluju u potpunosti jednako:

```
# snort -dev
# snort -d -e -v
# snort -v -de
```

Važno je i napomenuti da po definiciji program počinje slušati na mrežnom sučelju *eth0*, pa ukoliko računalo ima više mrežnih sučelja, a potrebno je slušati na nekom drugom sučelju od podrazumijevanog, može se koristiti opcija *-i* iza koje slijedi naziv mrežnog sučelja. Iz prethodno navedenoga ispisa po znaku ljuske (eng. *prompt*) može se i primijetiti da se Snort mora pokrenuti koristeći privilegirani korisnički račun, tj. koristeći *root* račun na Unix/Linux operacijskim sustavima, odnosno administratorski korisnički račun na Microsoft Windows operacijskom sustavu.

Sljedeći primjer pokazuje kompletan primjer korištenja Snorta u sniffer načinu rada kao i primjer ispisa pri snimanju prometa kod pokušaja spajanja protokolom *ssh* (eng. *secure shell*) na udaljeni poslužitelj `faramir.zemris.fer.hr`. Iz ispisa zaglavlja paketa izbačeni su oni dijelovi koji se ne odnose na navedeni pokušaj spajanja.

```
# snort -v -i eth1

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
02/02-17:17:02.437723 192.168.1.65:33052 -> 192.168.1.254:53
UDP TTL:64 TOS:0x0 ID:8124 IpLen:20 DgmLen:67 DF
Len: 39
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

02/02-17:17:02.449481 192.168.1.254:53 -> 192.168.1.65:33052
UDP TTL:64 TOS:0x0 ID:27019 IpLen:20 DgmLen:121
Len: 93
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+ =

02/02-17:17:02.449768 192.168.1.65:44354 -> 161.53.65.246:22
TCP TTL:64 TOS:0x0 ID:28798 IpLen:20 DgmLen:60 DF
******S* Seq: 0x2AE36FC1  Ack: 0x0  Win: 0x16D0  TcpLen: 40
```

```
TCP Options (5) => MSS: 1460 SackOK TS: 6430655 0 NOP WS: 6
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

02/02-17:17:02.460107 161.53.65.246:22 -> 192.168.1.65:44354
TCP TTL:56 TOS:0x0 ID:0 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0x2CDC5ED2  Ack: 0x2AE36FC2  Win: 0x16A0  TcpLen: 40
TCP Options (5) => MSS: 1440 SackOK TS: 256510792 6430655 NOP WS: 6
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

02/02-17:17:02.460148 192.168.1.65:44354 -> 161.53.65.246:22
TCP TTL:64 TOS:0x0 ID:28799 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x2AE36FC2  Ack: 0x2CDC5ED3  Win: 0x5C  TcpLen: 32
TCP Options (3) => NOP NOP TS: 6430657 256510792
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

*** Caught Int-Signal
Run time prior to being shutdown was 23.183057 seconds

===============================================================================
Snort received 52 packets
    Analyzed: 51(98.077%)
    Dropped: 0(0.000%)
    Outstanding: 1(1.923%)
===============================================================================
Breakdown by protocol:
      TCP: 33         (64.706%)
      UDP: 18         (35.294%)
     ICMP: 0          (0.000%)
      ARP: 0          (0.000%)
    EAPOL: 0          (0.000%)
     IPv6: 0          (0.000%)
  ETHLOOP: 0          (0.000%)
      IPX: 0          (0.000%)
     FRAG: 0          (0.000%)
    OTHER: 0          (0.000%)
  DISCARD: 0          (0.000%)
 InvChkSum: 0          (0.000%)

===============================================================================
Action Stats:
ALERTS: 0
LOGGED: 0
PASSED: 0
===============================================================================
Snort exiting
```

U ovom primjeru Snort je pokrenut na bežičnom (eng. *wireless*) sučelju *eth1* računala u lokalnoj mreži. Računalo ima adresu `192.168.1.64` s mrežnom maskom `255.255.255.0`, a adresa podrazumijevanog usmjernika (eng *default gateway*) je `192.168.1.254`. Adresa poslužitelja `faramir.zemris.fer.hr` je `161.53.65.246`, a usluga ssh je pokrenuta na standardnim pristupnim vratima `22/tcp`.

Nakon pokretanja program prvo ispisuje informacije o verziji, licenci, postavkama pokretanja te uključenim dodacima (eng. *plugins*). Radi kratkoće prikaza ti podaci su isključeni iz prikazanog ispisa. Zatim slijede informacije o snimljenim paketima. Podaci o pojedinim paketima su međusobno odvojeni jednostavnom ASCII grafikom. Prva dva prikazana paketa predstavljaju DNS promet, tj. DNS upit za adresom poslužitelja na koji se spaja, te odgovor s adresom poslužitelja. Prva linija ispisa sadržava vrijeme kada je paket

primljen, te izvorišnu i odredišnu IP adresu odnosno broj pristupnih vrata. U sljedećoj liniji nalaze se podaci o raznim poljima unutar IP paketa, kao i o samom paketu (*time to live, type of service, don't fragment*, duljina IP zaglavlja, datagrama itd.), te vrsta protokola koji se koristi na prijenosnom sloju. U ovom slučaju jasno je vidljivo da se DNS upiti prenose UDP protokolom (iznimka su prijenosi zone te odgovori veći od 512 okteta). Posljednja linija prikazuje ukupnu veličinu paketa u oktetima. Važno je i primijetiti da se ispis za prva dva slučaju kada se na prijenosnom sloju koristi UDP protokol razlikuje od ispisa za ostale pakete gdje se na prijenosnom sloju koristi TCP protokol.

Sljedeća tri paketa u prikazu predstavljaju pakete u protokolu sinkronizacije u tri koraka kod uspostave TCP veze (eng. *three way handshake*). Prve dvije linije u ispisu su identične slučaju za DNS upite, osim što se umjesto UDP protokola koristi TCP. Treća i četvrta linija su različite i u ovom slučaju prikazuju informacije o TCP protokolu s prijenosnog sloja (npr. *sequence* i *acknowledge* brojeve, oznake prozora itd.). Prvo polje u trećoj liniji je posebno važno kod dijagnostike jer pokazuje stanje zastavica u TCP zaglavlju. Ukoliko zastavica nije uključena, tj. u zaglavlju paketa ima vrijednost nule, u ispisu će na njenom mjestu stajati znak zvjezdice. U protivnome će kao oznake zastavica umjesto zvjezdice stajati sljedeće oznake: C (CWR), E (ECE), U (URG), A (ACK), P (PSH), R (RST), S (SYN), i F (FIN). Na prikazanom konkretnom primjeru mogu se primijetiti tri paketa s različitim upaljenim zastavicama. Pojedini paketi u tom prikazu odgovaraju već spomenutom protokolu sinkronizacije u tri koraka kod uspostave TCP veze:

1. Klijent šalje poslužitelju paket s upaljenom SYN zastavicom
2. Poslužitelj odgovara paketom s upaljenim zastavicama SYN i ACK. U slučaju da na poslužitelju nema pokrenute usluge na tim pristupnim vratima vraća se paket s upaljenim zastavicama RST i ACK.
3. Klijent potvrđuje uspostavu veze paketom upaljenom zastavicom ACK

Nakon toga, rad programa je zaustavljen pritiskom na tipke CTRL-C, tj. slanjem signala SIGINT Snort procesu. Primanjem ovog signala program prazni sve međuspremnike, zapisuje potrebne informacije na disk, te zaustavlja s radom ispisujući razne statistike sakupljene tijekom rada programa. Prve grupe govore o vremenu izvođenja programa, te o u ukupnom broju primljenih paketa i o broju i postotku obrađenih (eng. *analyzed*), ispuštenih (eng. *dropped*) te još neobrađenih paketa (eng. *outstanding*). Iz ispisa je vidljivo da je program prekinut upravo u trenutku kada je obrađivao paket, pa otuda i jedan paket u statistici za neobrađene pakete. Grupa nazvana *breakdown by protocol* daje statistički uvid u količinu prometa po pojedinoj vrsti protokola. U ovom slučaju sav promet otpada na TCP i UDP promet. Posljednja grupa u statistici govori o poduzetim akcijama od strane sustava, međutim u ovom načinu rada te statistike će uvijek biti na nuli pošto se funkcionalnost svodi samo na snimanje prometa. Detaljni opis pojedinih akcija koje su navedene u ovoj statistici slijedi u narednim poglavljima.

3.2.2. Packet logger način rada

Način rada *packet logger* je u svojoj osnovi isti kao i sniffer način rada, osim što se umjesto na standardni izlaz informacije zapisuju u datoteku dnevnika ili na neki drugi izlazni medij, ovisno o odabranim izlaznim modulima. Za pokretanje Snorta u ovome načinu rada, dovoljno je korištenjem opcije *-l* zadati apsolutnu ili relativnu stazu do direktorija gdje će se zapisivati datoteke dnevnika:

```
# snort -dev -l ./logs
```

Direktorij koji se koristi za zapisivanje datoteka dnevnika mora biti kreiran prije pokretanja programa, u protivnome će biti prijavljena pogreška da direktorij ne postoji. Podaci o svakome pročitanom paketu smještaju se unutar zadanog direktorija. Hijerarhija i imena dodatnih direktorija u zadanom direktoriju određuju se ovisno o IP adresama pronađenim u pročitanim paketima. Ako se koristi samo opcija *-l*, ponekad će se za ime direktorija koristiti adresa računala na kojem je program pokrenut, a nekad će se koristiti adresa udaljenog računala. Ukoliko je potrebno sve adrese gledati relativno prema adresi računala na kojemu se program izvodi, potrebno je koristiti opciju *-h* (*home network*) s argumentom koji označava adresu mreže s koje se pokreće program. Jednaki efekt ima definiranje varijable `$HOME_NET` unutar konfiguracijske datoteke `snort.conf` kao što je prikazano kasnije u nastavku poglavlja.

```
# snort -dev -l ./logs -h 192.168.1.0/24
```

Koristeći ove opcije, Snort će bilježiti sva zaglavlja s prijenosnog, mrežnog i podatkovnog sloja, te ih bilježiti označene relativno prema adresama mreže `192.168.1.0/24`. U direktoriju sa zapisima nalaziti će se dodatni direktoriji s adresom udaljenih računala u svojem imenu. U slučaju da su i odredište i polazište paketa na istoj mreži, ime direktorija se određuje prema onom paketu koji ima veći broj pristupnih vrata (obično pošiljatelj).

Podrazumijevani način zapisa datoteka dnevnika je u ASCII obliku, međutim ukoliko se očekuje velika količina prometa, preporuča se korištenje binarnog formata za zapis koji je puno brži i kompaktniji te može uštediti znatne količine prostora. Unaprijed pripremljeni Snort paketi u različitim distribucijama mogu koristiti i binarni način zapisa kao podrazumijevani, ukoliko je autor paketa naveo takve postavke. Kod korištenja binarnog načina zapisa datoteka dnevnika nije potrebno koristiti opciju *-h* za navođenje adrese lokalne mreže jer se svi podaci zapisuju u istu datoteku. Format zapisa u ovom načinu rada istovjetan je zapisu kakvog ga daje komando linijski alat `tcpdump`. Za zapisivanje paketa u binarnog obliku, program je potrebno pokrenuti koristeći opciju *-b*:

```
# snort -b -l ./logs
```

Koristeći opciju *-b*, nije potrebno koristiti ostale opcije za prikaz više detalja o paketu poput opcija *-d* i *-e*, jer binarni oblik u sebi već uključuje ne samo sva zaglavlja nego i kompletan sadržaj pročitanog paketa. Ovakav zapis se kasnije može pročitati s bilo kojim programom koji podržava tcpdump zapis, primjerice sa samim programom `tcpdump` ili programom Ethereal/Wireshark. Binarni zapis se također može ponovno iskoristiti i od strane samog programa Snort, koristeći opciju *-r* za ponavljanje (eng. *replay*). Na ovaj način može se proizvesti zapis u ASCII obliku kakav je bio prikazan u prethodnog poglavlju koristeći prethodno snimljeni tcpdump zapis:

```
# snort -dv -r packet.log
```

Ponavljanje paketa može se koristiti i za ispitivanje novih pravila. Na primjer, na jednom računalu se može izvesti čitav niz napada, te zatim datoteku sa snimljenim prometom prenijeti na drugo računalo na kojem želimo ispitivati mogućnosti otkrivanja. Jednako tako, moguće je rekonstruirati sav promet koji je uzrokovao napad na nekom računalu. Ovo je naročito korisno kod forenzičke analize kompromitiranih poslužitelja, pa je moguće ponoviti sve događaje koji su se događali za vrijeme napada u kontroliranim laboratorijskim uvjetima.

Dodatna mogućnost kod ponavljanja zapisa je i korištenje BPF notacije (eng. *Berkley packet filter*). BPF notacija omogućava filtriranje prikazanih paketa korištenjem raznih izraza. BPF izrazi se sastoje od primitiva poput *src host* (izvorišna adresa), *dst host* (odredišna adresa), *port* (pristupna vrata), *ip proto* (vrsta protokola) itd. BPF notaciju mogu koristiti i slični programi poput `tcpdump` i Wireshark, te svi programi koji implementiraju biblioteku *libpcap*. Primjerice, želimo li iz prethodno snimljene datoteke prikazati samo pakete koji su koristili ICMP protokol, dostupna nam je sljedeća notacija:

```
# snort -dv -r packet.log icmp
```

Mogući su i znatno složeniji izrazi za odabir paketa koji će biti prikazani, a jednostavniji izrazi se mogu kombinirati korištenjem logičkih operatora *not*, *and* i *or*. Ukoliko se neki logički operator odnosi na grupu izraza, tada se grupa tih izraza može navesti unutar zagrada. U slučaju korištenja specijalnih znakova poput zagrada, BPF notacija se mora navoditi unutar dvostrukih navodnika, ili se svaki specijalni znak mora koristiti s takozvanim *escape* znakom. U protivnome se specijalni znakovi interpretiraju u onom kontekstu koje im dodjeljuje ljuska u kojoj se naredba pokreće, a ne u kontekstu Snorta odnosno BPF zapisa. Primjer svega navedenoga uz složenije BPF zapise prikazuje naredba koja će ispisati TCP pakete s odredišnim pristupnim vratima 80, ali koji ne dolaze sa računala `faramir` i `graph`:

```
# snort -dv -r packet.log tcp port 80 and not \(faramir or graph\)
```

U slučaju dužih BPF zapisa za odabira paketa, nepraktično ih je navoditi u komandnoj liniji, pa postoji mogućnost njihovog navođenja unutar tekstualne datoteke. Sadržaj takve datoteke se može proslijediti programu na obradu korištenjem *-F* opcije. BPF zapis je izuzetno kompleksan te pruža fleksibilan i moćan način odabira paketa za prikaz. Daljnje izlaganje o njegovim mogućnostima izlazi izvan opsega ovog rada, pa se neće detaljnije proučavati. Detaljnije informacije o BPF notaciji mogu se pronaći u dokumentacijskoj stranici (eng. *UNIX man page*) alata `tcpdump`, te na web stranicama navedenim u popisu literature na kraju rada.

3.2.3. NIDS način rada

Primarna namjena programa Snort je rad u načinu rada za otkrivanje napada sa mreže, tj. kao mrežni IDS. Zbog toga je IDS najčešće korišteni način rada. Da bi Snort radio u IDS načinu rada, potrebno je pri pokretanju zadati stazu do datoteke s opisom pravila za prepoznavanje napada. Uz pravila prepoznavanja, datoteka sadrži niz drugih uputa vezanih uz pokretanje i rad Snorta, uputa za postavljanje pretprocesora i izlaznih modula,

ograničavanje broja upozorenja itd. Uobičajeni naziv ove datoteke je `snort.conf`, te se obično nalazi u direktoriju */etc/snort* ili */etc/snort/conf*. Pravila navedena o ovoj datoteci primjenjuju se na svaki pročitani mrežni paket i na osnovu njih se odlučuje o mogućim daljnjim akcijama. Ukoliko nije navedeno drukčije, podrazumijeva se zapisivanje svih informacija i rezultata otkrivanja u datoteke dnevnika u direktoriju `/var/log/snort`. Zbog brzine izvođenja, u ovom načinu rada se ne preporuča istodobni ispis paketa na standardni izlaz pomoću *-v* opcije. Isto tako, rijetko kad se koristi i opcija -e za prikaz zaglavlja s podatkovnog sloja. Najčešći način pokretanja u IDS načinu je sljedeći:

```
# snort -d -h 192.168.1.0/24 -c /etc/snort/snort.conf
```

Pokrenut na ovaj način, Snort će čitati pakete s podrazumijevanog mrežnog sučelja, ispitivati ih da li zadovoljavaju neka od navedenih pravila unutar konfiguracijske datoteke *snort.conf*, te ukoliko nađe neka pravila koja su zadovoljena zapisati zaglavlja i sadržaj paketa u datoteku dnevnika unutar */var/log/snort* direktorija. Format datoteka dnevnika je tekstualan (tj. ASCII zapis), a hijerarhija direktorija je istovjetna onoj opisanoj za packet logger načina rada. Detaljan opis opcija za postavljanje raznih pravila nalazi se u poglavlju o postavljanju Snorta.

Uz prethodni način pokretanja s osnovnim opcijama, postoji još niz drugih opcija za preciznije postavljanje funkcija u IDS načinu rada. Na primjer, podrazumijevani način zapisivanja datoteke dnevnika je korištenje ASCII zapisa s prikazom zaglavlja i sadržaja paketa. Dodatni načini zapisivanja mogu se odabrati pomoću *-A* opcije s nazivom načina zapisivanja kao opcijom. Načini zapisivanja se međusobno razlikuju po brzini rada te po količini detalja koji će se zapisivati. Pojedini načini zapisivanja i njihove karakteristike prikazani su u tablici 3.1.

Opcija	Opis
-A fast	Brzi način zapisivanja, zapisuje se samo vremenska oznaka, izvorišna i odredišna adresa paketa i pristupna vrata, te poruka upozorenja
-A full	Podrazumijevani način rada, zapisuje vremensku oznaku, poruku upozorenja, zaglavlja i sadržaj paketa
-A unsock	Slanje poruka upozorenja putem UNIX priključnica (socketa). Pogodno za povezivanje izlaza Snorta s drugim programima.
-A none	Isključeno zapisivanje
-A console	Poput fast načina rada, ali poruke se ispisuju na standardni izlaz umjesto u datoteku dnevnika
-A cmg	CMG zapis (nazvan po jednom od programera aplikacije). Prikazuje poruku upozorenja, zaglavlje i sadržaj paketa na standardnom izlazu. Koristi se uglavnom kod ispravljanja greška pri razvoju novih modula ili pisanja novih pravila.

Tablica 3.1: Načini zapisivanja datoteka dnevnika

Kao i kod packet logger načina rada, moguće je odabrati i binarni način zapisivanja korištenjem opcije *-b*. Zapisivanje na disk može se potpuno isključiti korištenjem opcije *-N*. Važno je i napomenuti da se većina navedenih opcija može navesti i u konfiguracijskoj

datoteci `snort.conf`, a u slučaju da se u datoteci nalaze opcije međusobno isključive s onima navedenim preko komandne linije, prioritet imaju opcije navedene u komandnoj liniji. U slučaju da je brzina izvođenja kritični parametar kod izvođenja programa, primjerice kod nadzora gigabitnih mreža, najbolje je koristiti kombinaciju zapisivanja u binarnom obliku zajedno s *fast* načinom zapisivanja datoteke dnevnika:

```
# snort -b -A fast -c /etc/snort/snort.conf
```

Datoteke dobivene na ovakav način najčešće se analiziraju kasnije pomoću posebnih alata za tu namjenu, npr. programom *barnyard.*

Uz navedene načine zapisa u obične tekstualne ili binarne datoteke, moguće je koristiti i zapisivanje upozorenja putem *syslog* infrastrukture. Korištenje syslog infrastrukture je vrlo često na Unix i Linux operacijskim sustavima jer pruža jedinstveni način zapisivanja poruka od strane velikog broja raznih programa koji ga podržavaju i koriste. Syslog može zapisivati primljene zapise u razne datoteke na veliki broj načina, ovisno o odabranim postavkama u konfiguracijskoj datoteci `syslog.conf`. Više informacija o syslogu i mehanizmima upozoravanja mogu se pronaći u man stranicama `syslogd(8)` i `syslog.conf(5)`. Snort uobičajeno koristi dva syslog mehanizma upozoravanja: `LOG_AUTHPRIV` i `LOG_ALERT`, ali moguć je odabir drugih putem određenih izlaznih modula. Za korištenje sysloga, kod pokretanja Snorta potrebno je koristiti opciju *-s*.

Kod IDS načina rada, standardni dio zapisa u datoteci dnevnika je i poruka upozorenja (eng. *alert*). Primjer takvog zapisa je sljedeći:

```
[**] [116:56:1] (snort_decoder) T/TCP Detected [**]
```

Poruke upozorenja nalaze se unutar skupina znakova "`[**]`". Tri broja unutar uglatih zagrada međusobno odvojeni dvotočkom označavaju identifikacijski broj generatora (eng. *generator identificator*), identifikacijski broj potpisa (eng. *signature identificator*) te verziju potpisa (eng. *signature revision*). Identifikacijski broj generatora označava koja je komponenta sustava stvorila poruku upozorenja. Popis identifikacijskih brojeva pojedinih komponenti dostupan je u datoteci `etc/generators` unutar direktorija s izvornim kodom Snorta. Sljedeći broj je identifikacijski broj pravila pomoću kojega je moguće točno odrediti koje pravilo je paket zadovoljio i zbog kojeg je stvoreno upozorenje. Svako pravilo ima jedinstveni identifikacijski broj i u pravilu je grupa brojeva vezana uz pojedini pretprocesor. Njihov popis se također može naći unutar direktorija s programskim kodom, u datoteci `etc/gen-msg.map`. Treći broj u grupi služi za označavanje pojedinih verzija pravila. Nakon grupe identifikacijskih brojeva slijedi kratki opis pravila koje se podudara s pročitanim mrežnim paketom.

Svaki puta kada Snort utvrdi da se sadržaj ili zaglavlje nekog pročitanog paketa podudara s nekim od pravila u konfiguracijskoj datoteci, sustav stvara *događaj* (eng. *event)*. Uz svaki događaj je vezana određena akcija. Akcija koja će se poduzeti za određeni događaj se određuje na osnovi definicije potpisa. Primjerice, sljedeće pravilo ima definiranu akciju *alert*.

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"TELNET root
login"; flow:from_server,established; content:"login|3A| root";
classtype:suspicious-login; sid:719; rev:7;)
```

Pravilo definira koji uvjeti na pročitanom paketu moraju biti ispunjeni da bi se stvorio događaj. Uvjeti mogu biti polazna i odredišna mreža ili adresa, pristupna vrata, sadržaj paketa itd. Prva ključna riječ u zaglavlju pravila (u ovom slučaju *alert*) određuje akciju. Detaljni opis ključnih riječi i njihovih opcija u definiciji pravila biti će objašnjen u poglavlju o pisanju vlastitih pravila. U definiciji pravila moguće je koristiti sljedeće akcije:

Akcija	Opis
alert	Prvo se stvara upozorenje navedenom metodom, zatim se zapisuje informaciju u datoteku dnevnika
log	Informacija se zapisuje u datoteku dnevnika u tekstualnom ili binarnom obliku
pass	Paket se ignorira, odnosno ne stvara se nikakav zapis u datoteci dnevnika niti se poduzimaju ikakve druge akcije
activate	Djeluje kao alert pravilo, a zatim se aktivira neko drugo dinamičko pravilo
dynamic	Ostaje neaktivno dok ga ne aktivira neko *activate pravilo*, a tada se ponaša kao *log* akcija
drop	Koristi se program iptables za ispuštanje paketa
reject	Koristi se program iptables za odbijanje paketa, poput *drop* akcije, ali se u ovom slučaju pošiljatelju šalje odgovor.
sdrop	Koristi se program iptables za ispuštanje paketa, ali ne zapisuje nikakve informacije u datoteku dnevnika

Tablica 3.2: Akcije Snort pravila

Također je moguće definirati i vlastite akcije. Primjerice, sljedeće pravilo definira akciju koja će napraviti zapis koristeći syslog infrastrukturu, te zatim napraviti zapis u MySQL relacijskoj bazi podataka:

```
ruletype log_syslog_mysql {
  type alert
  output alert_syslog: LOG_AUTH LOG_ALERT
  output dataabase: log, mysql, user=snort dbname=snort
}
```

Nakon definicije tipa akcije (alert), za zapisivanje podataka preko sysloga odnosno u MySQL bazu podataka, koriste se izlazni moduli pomoću ključne riječi *output*, zajedno s podacim potrebnim za spajanja na bazu poput korisničkog imena i lozinke, adrese poslužitelja i imena baze.

Standardni redoslijed kojim Snort obrađuje pravila je sljedeći:

1. alert
2. pass
3. log

Redoslijed se, ukoliko je potrebno, može promijeniti korištenjem *-o* opcije koja će promijeniti redoslijed obrade pravila u pass – alert – log.

Ova opcija će u budućim verzijama programa biti izbačena jer je danas uobičajeno birati poredak obrade pravila s preciznijim opcijama poput *-alert-before-pass*, *-treat-drop-as-*

alert itd. Opcija *-treat-drop-as-alert* će uzrokovati da se akcije drop, sdrop i reject zapisuju u datoteku dnevnika kao i pravila s akcijom alert. Korištenjem dodatne opcije *-proces-all-events* obrađivati će se sva pravila bez obzira na njihov redoslijed, izuzevši pravila s *pass* akcijom. Podrazumijevano ponašanje je da se prekida pretraživanje daljnjih pravila jednom kada se naiđe na prvo pravilo koje zadovoljava akciju navedenu u odabranom poretku (npr. alert – pass – log ili pass – alert – log).

3.2.4. Način rada korištenjem iptables-a

Počevši od verzije 2.3.0 RC1, uz standardne mogućnosti za otkrivanje napada, na Linux operacijskom sustavu Snort ima ugrađene i osnovne mogućnosti sustava za sprječavanje napada sa mreže (eng. *intrusion prevention system*). Mogućnosti sustava za sprječavanje napada sa mreže (u daljnjem tekstu koristi se uobičajena kratica IPS) dostupne su korištenjem posebnog načina rada - *Snort Inline*. Za razliku od svih prethodno navedenih načina rada koji za primanje paketa koriste standardni mehanizam putem biblioteke *libpcap*, u inline načinu rada za primanje paketa se koristi netfilter programski okvir (eng. *framework*) i modul jezgre, odnosno najpoznatiji program izgrađen na tom okviru – *iptables*. Netfilter i iptables kao i primjeri njihovog korištenja su detaljnije opisani u poglavlju o dodatnim programima koji se koriste iz Snort.

Koristeći iptables, u inline načinu rada Snort može propuštati ili blokirati određene pakete primjenom raznih iptables pravila. Kada Snort radi u inline načinu rada, dostupne su sljedeće akcije koje se mogu navesti u zaglavlju pravila:

1. *drop*
2. *sdrop*
3. *reject*

Akcija drop prevodi se direktno u iptables opciju *-j DROP*, kao i akcija sdrop, s iznimkom što se drop zapisuje u datoteku dnevnika, a sdrop se ne zapisuje. Akcija reject se prevodi direktno u iptables opciju *-j REJECT*. Za razliku od akcije drop, reject će ispustiti paket, ali će pošiljatelju vratiti TCP paket s upaljenom zastavicom RST (u slučaju TCP protokola) ili će vratiti ICMP poruku *port unreachable* (u slučaju UDP protokola). Kao i kod prethodnih načina rada, i u inline načinu rada postoji unaprijed definirani redoslijed izvođenja akcija:

1. *activation*
2. *dynamic*
3. *drop*
4. *sdrop*
5. *reject*
6. *alert*
7. *pass*
8. *log*

Moguće je koristi i već spomenutu opciju *-o* za izmjenu redoslijeda izvođenja akcija. U tom slučaju akcija *pass* imati će veći prioritet od akcija *drop, sdrop* i *reject*, tj. akcija *pass* će se nalaziti na trećem mjestu u navedenoj listi redoslijeda obrade akcija.

Za pokretanje Snorta u inline načinu rada potrebno je napraviti posebne pripremne akcije prije pokretanja samog programa. Nakon što je Snort postavljen i preveden s *-use-inline* opcijom, potrebno je napraviti nekoliko koraka. Prvi korak je učitavanje *ip_queue* modula Linux jezgre (eng. *kernela*) korištenjem naredbi `modprobe` ili `insmod`. Zatim je potrebno preusmjeriti promet koji se mora analizirati u posebni iptables red – `QUEUE`. Snort inline će analizirati samo one pakete koji dođu u ovaj red. Primjerice. ukoliko je potrebno nadzirati samo odlazni promet na pristupna vrata `80/tcp` (web promet), potrebno je dodati posebno iptables pravilo. Pravilo je prikazano zajedno s naredbom za učitavanje potrebnog modula jezgre:

```
# modprobe ip_queue
# iptables -A OUTPUT -p tcp --dport 80 -j QUEUE
```

Ovime se sav odlazni promet prvo stavlja u red `QUEUE`. Ono što se zapravo događa u pozadini je prebacivanje paketa iz područja privilegiranog načina rada jezgre (eng. *kernel space*) u korisnički način rada (eng. *user space*). Prebacivanjem prometa u navedeni iptables red, paketi se mogu analizirati od strane Snorta pokrenutog u inline načinu rada:

```
# snort_inline -QD -c drop.conf -l /var/log/snort_inline
```

Najčešće korištene opcija kod pokretanja inline načina rada su:

- *-Q*, primanje paketa od strane iptables-a
- *-D*, pokretanje programa u *daemon* načinu (proces nema kontrolirajući terminal, izvodi se u pozadini, izlaz je preusmjeren u posebne datoteke itd.)
- *-c*, staza do datoteke s postavkama
- *-l*, odabir direktorija za stvaranje dnevničkih zapisa

Kod pokretanja programa u daemon načinu rada identifikacijski broj procesa (PID) se zapisuje u datoteku `/var/run/snort_inline.pid` ukoliko nije drukčije eksplicitno navedeno korištenjem opcije *-pid-path*. Postojanje ove datoteke omogućava da je u jednom trenutku pokrenuta samo jedna instanca Snort programa u daemon načinu rada. Ovakvo podrazumijevano ponašanje je moguće zaobići korištenjem opcije *-nolock-pidfile*. Važno je napomenuti da se opcija -D može koristiti i u bilo kojem drugom načinu rada programa. Ukoliko se želi prisiliti program na ponovno čitanje konfiguracijskih datoteka slanjem signala SIGHUP, kod pokretanja se mora koristiti puna staza do Snort izvršne datoteke, tj. program se mora pokrenuti na sljedeći način:

```
# /usr/local/bin/snort -d -h -s -D 192.168.1.0/24 -c snort.conf
```

Preporuka za rad sa Snortom u inline načinu je najčešće korištenje već gotovog proizvoda *Honeynet Snort Inline Toolikt* koji se sastoji od statički prevedenog Snorta sa inline opcijama, te se nalazi na već pripremljenom *live* CD-u s instaliranim Linux operacijskim sustavom. Korištenje Honeynet CD-a je vrlo često i popularno. S proizvodom dolazi i skup već definiranih pravila za ispuštanje paketa, odnosno *drop.conf* konfiguracijskih datoteka za Snort, kao i već složenim raznim skriptama za održavanje. Primjerice na CD-u se nalaze

već gotove skripte za rotiranje dnevničkih zapisa (eng. *log rotation*), automatsku konfiguraciju i stvaranje konfiguracijskih datoteka itd. Na istom CD-u je dostupan i niz drugih dodatnih programa za analizu i grafički prikaz sakupljenih podataka korištenjem standardne LAMP platforme, poput onih opisanih u poglavlju o dodatnim programima. Snort inline kao i Honeynet Snort Inline Toolkit nije primarna tema ovoga rada pa neće biti detaljnije analiziran. Dodatne informacije su također dostupne u Snort dokumentaciji kao i u dokumentaciji Honeynet Snort Inline Toolkit projekta.

3.3. Postavljanje

Većina parametara programa Snort navodi se unutar konfiguracijske datoteke `snort.conf`, koja se najčešće nalazi u direktoriju */etc/snort*. Pošto bi konfiguracijska datoteka bila prilično velika kada bi se se postavke, uključujući i pravila odnosno potpise napada, navodile u jednoj datoteci, najčešće se potpisi izdvajaju u posebne datoteke. U posebne datoteke se često odvajaju i grupiraju međusobno povezane postavke. Datoteke izdvojene na ovakav način mogu se naknadno uključiti u glavnu konfiguracijsku datoteku korištenjem ključne riječi `include`. Na primjer, u datoteci `snort.conf` mogu postojati linije:

```
include classification.config
include rules/exploit.rules
```

Ključna riječ `include` funkcionira slično naredbi `#include` pretprocesora programskog jezika C, tj. red s ključnom riječi `include` i stazom do datoteke će se zamijeniti sadržajem te datoteke. Pri uključivanju datoteke potrebno je obratiti pažnju na činjenicu da će sve vrijednosti varijabli koje su definirane prije uključivanja biti zamijenjene onima unutar uključene datoteke ukoliko se pojave ista imena. Na kraju Snort naredbi u konfiguracijskoj datoteci ne navodi se točka zarez.

Vrlo često korišteni element u konfiguracijskim datotekama su i komentari. Namjena komentara je povećati čitljivost datoteke i opisati manje razumljive opcije, kao i dokumentirati moguće izmjene u samoj datoteci. Komentari se navode slično kao i u skriptama ljuske, korištenjem "#". Komentari se mogu nalaziti na bilo kojoj poziciji u bilo kojoj liniji, primjerice:

```
# ovo je komentar
include rules.conf   # komentar
```

Sljedeća klasa elemenata konfiguracijskih datoteka su varijable. Varijable je moguće definirati korištenje tri ključne riječi:

- *var*
- *ipvar*
- *portvar*

Ključna riječ `var` koristi se za definiranje generičkih varijabli poput imena direktorija, datoteka, IP adresa i drugih često korištenih objekata. Ključna riječ `ipvar` koristi se za definiranje IP adresa, uz napomenu da se na ovaj način adrese mogu definirati samo ako je u programu uključena IPv6 potpora, dok je u protivnome (bez IPv6 podrške) za adrese

potrebno koristiti ključnu riječ `var` kao što je prethodno opisano. Konačno, `portvar` se koristi za definiranje raspona ili pojedinačnih pristupnih vrata. Kod svakog daljnjeg pojavljivanje varijable, ispred njezinoga imena mora se navesti dolar ("$"), a kod parsiranja datoteke Snort će zamijeniti varijablu njenom vrijednošću. Ukoliko se ista varijabla definira više puta, relevantna je samo zadnja definicija.

Navedene ključne riječi koriste se vrlo često i mogu se pronaći u svakoj imalo kompleksnijoj `snort.conf` datoteci. Na primjer, prvo se definiraju razni direktoriji, kao i IP adrese:

```
var HOME_NET 192.168.1.0/24
var EXTERNAL_NET !$HOME_NET
var RULE_PATH /etc/snort/rules
```

U navedenom ispisu može se vidjeti i primjer korištenja operatora negacije (!) koji se uobičajeno koristi upravo za definiranje adresa ili pristupnih vrata izvan nekog zadanog opsega. U ovom slučaju definira se da je vanjska mreža sve što nije na privatnoj mreži navedenoj u varijabli `HOME_NET`.

Kod navođenja IP adresa korištenjem ključne riječi `ipvar`, uz definiranje pojedinačnih adresa ili mreža korištenjem CIDR zapisa, moguće je korištenjem posebnog zapisa definirati i grupe od više adresa ili mreža. Pojedinačne vrijednosti se navode na već opisani način, a grupe adresa mogu se navesti unutar uglatih zagrada, međusobno odvojene zarezom. Dodatno, moguće je koristiti i već prije spomenuti operator negacije. Primjerice, privatna mreža se može definirati na ovakav način:

```
ipvar PRIVATE_NET [192.168.1.0/24, 10.1.1.0/24]
```

Negaciju je moguće koristiti i sa grupama adresa:

```
ipvar MY_NET [1.1.1.1, 2.2.2.2/23, ![2.2.2.2, 2.2.2.3]]
```

Dodatna ključna riječ dostupna kod definiranja IP adresa je `any` koja podrazumijeva kompletni IP prostor, odnosno mrežu 0.0.0.0/0. Ključnu riječ `any` nije moguće koristiti zajedno s operatorom negacije. Na primjer, moguće je napisati:

```
var HOME_NET any
```

Kod definiranja pristupnih vrata ključnom riječju `portvar`, uz navedeni način zapisivanja više vrijednosti korištenjem uglatih zagrada moguće je koristiti i zapis za definiranje raspona pristupnih vrata. U tu svrhu koristi se dvotočka:

```
portvar MY_PORTS [22, 80, 1024:1050]
```

Kao i kod navođenja IP adresa, moguće je koristiti ključnu riječ `any` za specificiranje svih pristupnih vrata. Također nije moguće koristiti `any` u kombinaciji s operatorom negacije. U ovom kontekstu `any` će označavati pristupna vrata od 1 do 65535. Zato su sljedeće dvije definicije istovjetne:

```
portvar PORTS any
portvar PORTS [1:65535]
```

Uz varijable se može koristiti i specijalan skup modifikatora (eng. *variable modifiers*) koji mijenjaju značenje varijabli ovisno o kontekstu u kojem se koriste. Dostupni su sljedeći modifikatori varijabli:

Modifikator	Opis
var	Definicija meta-varijable
$(var) ili $(var)	Zamjena sa sadržajem varijable
$(var:-default)	Zamjena sa sadržajem varijable, ukoliko varijabla nije definirana onda se koristi podrazumijevana vrijednost zadana modifikatorom
$(var:?message)	Zamjena sa sadržajem varijable, ukoliko varijabla nije definirana, ispisuje se poruke pogreške zadana modifikatorom i program zaustavlja s radom

Tablica 3.3: Modifikatori Snort varijabli

Posljednja dva modifikatora se koriste u definiranju pravila za provjeru dostupnost raznih važnih varijabli, npr. varijable `HOME_NET`.

Sljedeća često korištena ključna riječ u konfiguracijskoj datoteci je `config`. Ova ključna riječ koristi se za postavljanje vrijednosti globalnih varijabli vezanih uz razne Snort postavke. Globalne varijable često su povezane s opcijama koje se zadaju iz komandne linije. U trenutnoj verziji (2.8) dostupno je preko pedeset takvih globalnih varijabli, a općeniti format korištenja i neki od primjera su prikazani na sljedećem ispisu:

```
# config <opcija> [:vrijednost]

config utc         # koristi se univerzalno umjesto lokalnog vremena
config verbose     # ispis poruka na standardni izlaz (opcija -v)
config set_gid: 30 # postavi GID procesa
```

Popis svih podesivih globalnih opcija, njihov opis, namjena i moguće vrijednosti te primjeri korištenja nalaze se u Snort dokumentaciji.

Kod početnog postavljanja Snorta treba obratiti i pozornost na količinu podataka koji će se zapisivati u datoteke dnevnika. U određenim situacijama može se dogoditi da će se zapisivati iznimno velike količine podataka za ispravan promet, što će otežati pregled tih istih datoteka. U takvim slučajevima vrlo je teško pronaći informacije o pravim napadima. Vrlo često navođen ogledni primjer ovog problema su upozorenja vezana uz UPnP poruke koje stvaraju razni mrežni uređaji. Postoje dva moguća rješenja ovog problema:

- ograničavanje događaja (eng. *event thresholding*)
- blokiranje događaja (eng. *event suppression*)

Kod ograničavanja događaja programu se daju posebne upute za pravila koja stvaraju puno događaja. Uz smanjivanje broja zapisa u datoteci dnevnika, također se utječe i na brzinu rada samog sustava. Ideja pristupa je ograničiti broj zapisa u jedinici vremena na neki maksimalni broj. Postoje tri načina ograničavanja:

- ispis upozorenja samo za prvih *n* stvorenih događaja
- ispis upozorenja za svako *n*-to pojavljivanje nekog događaja
- kombinacija oba načina

Opcije za ograničavanje događaja mogu se navesti unutar samog pravila ili se mogu navesti kao zasebna opcija unutar konfiguracijske datoteke. Ograničenja se najčešće navode u zasebnoj datoteci, obično u `/etc/snort/threshold.conf`. U slučaju da se ne navodi kod definicije pravila, potrebno je navesti i identifikacijski broj generatora te identifikacijski broj potpisa na koje se ograničenje odnosi.

Za razliku od prethodnog pristupa, kod blokiranja događaja se sprječava pojavljivanje nekog događaja bez da se ukloni njegovo pravilo iz baze potpisa. Uz blokiranje određenih događaja, moguće je realizirati i blokiranje događaja po određenim IP adresama ili mrežnim rasponima tj. mrežama. Pravila za blokiranje događaja obrađuju se prije pravila za ograničavanje događaja. Kao i kod pravila za ograničavanje, potrebno je navesti identifikacijski broj generatora te identifikacijski broj potpisa za događaj koji se želi zaustaviti. Tako je moguće kompletno zabraniti stvaranje nekog događaja, ili je moguće napraviti selektivno stvaranje događaja ovisno o adresi s koje je paket stigao.

3.4. Pretprocesori

Pojam pretprocesora koji je definiran u teoretskom izlaganju o arhitekturi sustava za otkrivanje napada sa mreže ponešto se razlikuje od koncepcije pretprocesora koji se koriste u programu Snort. Počevši od verzije 1.5, Snort je dobio mogućnost proširivanja funkcionalnosti rada umetanjem raznih modula – *pretprocesora*. Akcije pretprocesora izvode se na paketu prije nego što dođe u sustav otkrivanja (eng. *detection engine*), odmah nakon što se paket dekodira u Snort dekoderima. Pretprocesori mogu obavljati jednostavne akcije poput zamjene određenih nizova znakova u paketu, ali mogu obavljati i kompleksnije zadaće pa čak i preuzeti neke uloge sustava otkrivanja. Pretprocesori se izvode neovisno o Snort pravilima, a sažeto se njihova funkcija može svesti na sljedeći skup akcija:

- stvaranje upozorenja
- klasifikacija paketa
- ispuštanje paketa prije nego što dođu do sustava otkrivanja

Za učitavanje (eng. *loading*) i postavljanje pretprocesora, u `snort.conf` konfiguracijskoj datoteci koristi se ključna riječ `preprocessor`. Sintaksa korištenja ključne riječi je identična onoj od ključne riječi `configure`, tj. sastoji se od same ključne riječi, imena pretprocesora i opcija odvojenih od naziva pretprocesora pomoću dvotočke. Tako npr. sljedeći primjer učitava pretprocesor `minfrag` i prosljeđuje mu vrijednost 128:

```
preprocessor minfrag: 128
```

Često korišteni Snort pretprocesori prikazani su u sljedećoj tablici.

Pretprocesor	Opis
frag3	Defragmentacija paketa
stream4	TCP sastavljanje asocijacija
flow	Praćenje tijeka konekcija
stream5	Objedinjuje funkcionalnosti flow i stream4 pretprocesora
sfportscan	Prepoznavanje skeniranja pristupnih vrata
rpc_decode	Dekodiranje RPC protokola
perfmon	Služi za mjerenje učinkovitosti drugih komponenti Snorta
http_inspect	Dekodiranje HTTP protokola
smtp	Dekodiranje SMTP protokola
ftp_telnet	Dekodiranje FTP i telnet protokola
ssh	Dekodiranje SSH protokola
dcerpc	Dekodiranje SMB i DCE/RPC protokola
dns	Dekodiranje DNS protokola

Tablica 3.4: Snort pretprocesori

Frag3 pretprocesor je namijenjen defragmentaciji IP paketa. Nastao je kao nadogradnja ranije verzije frag2 s ciljevima da ga poboljša u smislu brzine izvođenja i podesivosti, te da omogući modeliranje takozvanoga *target based host* modela (detalji o ovom modelu mogu se pronaći u dokumentaciji pretprocesora). Unutrašnja implementacija ovog dodatka optimizirana je za što brže izvođenje, pa se koriste strukture podataka poput samopodešavajućih stabala (eng. *splay trees, self balancing trees*) te povezanih lista (eng. *linked lists*). Frag3 može prepoznati ukupno osam različitih neuobičajenih situacija unutar pročitanih paketa, većinom povezanih uz napade fragmentacijom paketa. Ime pretprocesora koje se koristi u konfiguracijskim datotekama je frag3_global, a dostupne su tri opcije pomoću kojih se može upravljati radom pretprocesora. To su:

- *max_frags*: označava maksimalni broj dijelova paketa koji se trenutno mogu obrađivati, podrazumijevana vrijednost je 8192
- *memcap*: označava maksimalnu količinu memorije koja se može koristiti od strane ovog pretprocesora, podrazumijevana vrijednost je 4 MB
- *prealloc_flags*: upravlja načinom rukovanja memorijom

Sljedeći pretprocesor je *stream4*. Namjena ovog pretprocesora je omogućiti sastavljanje TCP veza (asocijacija) te omogućiti praćenje stanja veze (eng. *stateful analysis*). Koristeći stream4, moguće je ignorirati takozvane *stateless* napade poput napada uskraćivanjem usluge i sličnih. Nadalje, moguće je pratiti veliki broj paralelnih TCP asocijacija. Podrazumijevana vrijednost je 8192 asocijacije, ali definiranjem odgovarajućih parametara moguće je pratiti i po 100000 istovremenih veza. Također je moguće koristiti pretprocesor za praćenje protokola koji ne uspostavljaju vezu, npr. UDP, korištenjem *-enable-stream4-udp* opcije kod prevođenja izvornog koda pretprocesora. Pretprocesor se sastoji od dva podesiva modula:

- *stream4*
- *stream4_reassemble*

Pretprocesor *flow* namijenjen je kao buduća zamjena za sve ostale pretprocesore koji se koriste za praćenje stanja veze. Trenutno su implementirane samo mogućnosti za prepoznavanje skeniranja pristupnih vrata, ali u budućim verzijama Snorta planira se proširenje s mogućnostima koje će zamijeniti stariji *conversation* pretprocesor. U *flow* pretprocesoru, IP tok podataka (eng. *flow*) je jedinstven ako se razlikuje od ostalih tokova podataka po protokolu prijenosnog sloja (*ip_proto*), te odredišnoj i izvorišnoj IP adresi (*dip*, *sip*), odnosno pristupnim vratima (*dport*, *sport*). Podrazumijeva se da je vrijednost opcija *dport* i *sport* jednaka nuli ukoliko se ne radi o protokolima TCP i UDP.

Stream5 pretprocesor je namijenjen kao zamjena za pretprocesore *stream4*. Njegovim korištenjem moguće je pratiti TCP i UDP protokole, kao i izvoditi praćenje TCP asocijacija. *Stream5* se ne može koristiti ukoliko je u konfiguracijskoj datoteci navedeno da se koristi pretprocesor *stream4*. Pretprocesor posjeduje sljedeće mogućnosti:

- Praćenje prijenosnih protokola: Uz klasične TCP veze mogu se pratiti i UDP veze koje se prepoznaju kao niz UDP paketa između dvije krajnje točke istih IP adresa i pristupnih vrata. Na isti način može se pratiti i protokol ICMP.
- Praćenje ciljeva: Prisutne su metode za praćenje TCP vremenskih oznaka, praćenje korištenja TCP zastavica i brojeva sekvenci vezanih uz isto odredište (cilj). Cilj je apstrakcija napadnutog računala koja pomaže pretprocesoru kod obavljanja njegovih operacija.
- Praćenje tokova: Podržan je takozvani Stream API (eng. *Application Programming Interface*) koji dozvoljava ostalim pretprocesorima da prepoznaju TCP veze, prate TCP tokove i ignoriraju tokove koji ne mogu utjecati na stvaranje sigurnosnih upozorenja (npr. transfer velikih datoteka).
- Otkrivanje neuobičajenih situacija: Praćenje neuobičajenog ponašanja u TCP protokolu, npr. pojavljivanje paketa s puno podataka u tijelu paketa, a koji ima upaljenu SYN zastavicu, primanje paketa izvan TCP prozora itd. Moguće je praćenje različitih ciljeva, npr. za određene operacijske sustave čiji mrežni stog dozvoljava upaljenu SYN zastavicu u paketu s podacima u tijelu paketa neće se stvoriti nikakvo upozorenje.

Sljedeći često korišteni pretprocesor je *sfPortscan*, koji omogućuje otkrivanje najčešće vrste napada - skeniranja pristupnih vrata. Pretprocesor je razvijen od strane tvrtke Sourcefire (otuda i slova *sf* u imenu). Skeniranje pristupnih vrata je najčešće samo uvod u kasnije ozbiljnije metode napada, a svrha metode je otkriti koje su sve usluge pokrenute na napadnutom računalu, odnosno prepoznati pokrenute ranjive usluge. Pravovremeno otkriveni pokušaj skeniranja pristupnih vrata može upozoriti na kasnije ozbiljnije akcije. Danas najpopularniji program za skeniranje je *nmap*. Korištenjem ovog programa, napadač može vrlo jednostavno izvesti čitav niz različitih vrsta skeniranja, od kojih su najpoznatije:

Vrsta skeniranja	Opis
TCP connect()	Uspostavlja se cijela TCP veza na zadani port, tj. izvršava se kompletna sinkronizacija u tri koraka
SYN	Napadač šalje SYN paket te ukoliko dobije SYN/ACK zaključuje da su pristupna vrata otvorena. U slučaju da su pristupna vrata zatvorena, napadač dobiva RST/ACK.
FIN	Napadač šalje FIN paket te ukoliko dobije RST/ACK zaključuje da su pristupna vrata zatvorena, u suprotnome su otvorena. Ovo ponašanje propisano je odgovarajućim RFC-om, ali mrežni stogovi pojedinih operacijskih sustava se ponašaju drukčije pa se ova vrsta skeniranja često koristi i kod identifikacije operacijskog sustava.
UDP	Na zadana pristupna vrata se šalje prazan UDP paket, tj. samo UDP zaglavlje. Ukoliko se kao odgovor dobije poruka ICMP port unreachable, pristupna vrata su zatvorena, inače su otvorena (pojednostavljeni prikaz, moguće je i primanje ostalih tipova ICMP poruka što označava da je promet prema pristupnim vratima filtriran).
XMAS tree	Kao i FIN tehnika skeniranja, ali su dodatno upaljene zastavice PSH i URG, s ciljem zbunjivanja zaštitnih stijena i usmjernika.

Tablica 3.5: Načini skeniranja pristupnih vrata

Pretprocesor *sfPortscan* može otkriti sljedeće klase skeniranja:

- klasična skeniranja
- *decoy* skeniranja
- raspodijeljena skeniranja

Klasične metode skeniranja su tehnike navedene u prethodnoj tablici. *Decoy* tehnika je tehnika u kojoj će paketi napadača biti pomiješani s paketima koje je također poslao napadač, ali koji u IP zaglavlju imaju upisanu neku drugu IP adresu pošiljatelja. Raspodijeljena skeniranja su skeniranja kod kojih napadač za napad koristi više računala istovremeno.

Pretprocesori poput SSH, FTP/Telnet, DNS, SMTP, HTTPInspect i DCE/RPC namijenjeni su analizi pojedinih vrsta protokola. Unutar protokola se traže neuobičajeni uzorci, tj. nepravilno korištenje koje nije propisano u definiciji protokola ili u odgovarajućem RFC-u. Tako se npr. kod protokola SMTP može pratiti pokušaje zlouporabe naredbi poput VRFY, EXPN i RCPT, ili se mogu tražiti pokušaji pristupa izvršnim datotekama unutar HTTP zahtjeva, poput pristupa sistemskoj ljusci.

3.5. Izlazni moduli

Izlazni moduli su se prvi puta pojavili u Snortu verzije 1.6. Namjena izlaznih modula je primati poruke koje stvaraju sustav otkrivanja i pretprocesori, te ih transformirati u neki pogodniji oblik za prikaz ili spremanje i slanje u neki drugi sustav ili program. Snort omogućava korištenje više izlaznih modula u isto vrijeme, a podrazumijevani način zapisivanja datoteke dnevnika u `/var/log/snort` je također standardni izlazni modul. Način postavljanja izlaznih modula vrlo je sličan postavljanju pretprocesora. U uvodnom izlaganju o Snortu već su spomenuti i neki drugi izlazni moduli, poput modula *syslog*. U trenutnoj verziji Snorta, dostupni su sljedeći dodatni izlazni moduli:

- unixsock
- tcpdump
- database
- csv
- unified i unified 2
- prelude
- aruba action

Izlazni modul *unixsock* kod pokretanja postavlja UNIX priključnicu (eng. *socket*) te na nju šalje upozorenja. Vanjski programi mogu slušati na toj priključnici i tim putem primati podatke od Snorta u realnom vremenu. Ovaj modul je trenutno u eksperimentalnoj fazi i ne preporuča se njegovo korištenje u osjetljivim okolinama.

Modul *tcpdump* zapisuje pakete u formatu kakav inače zapisuje već prije spomenuti program tcpdump. Ovakav zapis se najčešće koristi za naknadnu analizu prometa i glavna mu je prednost što je danas dostupan veliki broj dodatnih alata koji čitaju upravo ovaj format. Kao parametar, ovaj modul prima ime datoteke u koju će se spremati prikupljeni podaci.

Modul *database* koristi se za zapisivanje poruka upozorenja i informacija o paketima u relacijske baze podataka. Trenutačno su podržani gotovo svi poznatiji sustavi za upravljanje relacijskim bazama podataka, poput MySQL, PostgreSQL, Oracle itd. Kao parametre ovaj modul prima niz podataka potrebnih za spajanje na bazu podataka, primjerice ime ili adresu poslužitelja na kojem se nalazi baza podataka, pristupna vrata, korisničko ime i lozinku, ime baze podataka itd. Na ovom mjestu valja spomenuti i izlazni modul *prelude* koji se koristi za zapisivanje u bazu podataka koju koristi još jedan IDS sustav otvorenog koda, Prelude. Prelude u bazi podataka koristi različite strukture shema i tablica od Snorta, pa je zapise potrebno dodatno prilagoditi korištenjem izlaznog modula.

CSV (eng. *comma separated value*) izlazni modul koristi se za spremanje podataka u formatu pogodnom za pregledavanje raznim alatima poput tabličnih kalkulatora ili UNIX filtera (awk, cut, grep, sort, uniq, itd). Podaci se zapisuju u obliku običnih tekstualnih datoteka gdje su pojedini podaci odnosno polja međusobno odvojena zarezom. Kao argumente, ovaj modul prima ime datoteke u koju će se spremati podaci kao i dodatnu opciju za formatiranje ispisa (npr. mogu se ispisivati samo određena polja umjesto svih).

Modul *unified* je namijenjen izrazito brzom zapisivanju dnevničkih zapisa i preporuča se njegovo korištenje u okolinama s vrlo velikim količinama mrežnog prometa. Kao rezultat korištenja zapisuju se dvije datoteke: *alert* i *log*. Datoteka s upozorenjima sadrži podatke o događajima na visokoj razini, npr. vrstu protokola, IP adrese, pristupna vrata itd. Log datoteka sadrži detaljne informacije o paketu, poput njegovog punog sadržaja, opisa aktiviranog pravila itd. Noviji je modul *unified2* namijenjen kao zamjena za unified modul te pruža dodatne opcije kod postavljanja.

3.6. Prilagođena pravila

Za opis pravila Snort koristi jednostavan i prilagodljiv jezik. Pravila se sastoje od dva dijela, zaglavlja pravila te postavki pravila. Zaglavlje pravila sadržava akciju, protokol, izvorišnu i odredišnu IP adresu i mrežnu masku, te izvorišna i odredišna pristupna vrata. Postavke pravila sadržavaju poruku upozorenja i opis dijelova paketa koji se trebaju ispitivati da bi se ustanovilo da li se pravilo može aktivirati za neki specifični paket. Sljedeći primjer pokazuje jednostavno Snort pravilo:

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

Tekst do prve otvorene zagrade je zaglavlje pravila, a tekst unutar zagrada su postavke pravila. Postavke pravila sadržavaju ključne riječi koje su odvojene od vrijednosti pomoću dvotočke. Važno je i napomenuti da je za svako pravilo njegovo zaglavlje obavezan dio, dok su postavke pravila neobavezan dio, međutim uobičajeno je da svako pravilo sadrži i dio s postavkama da bi se mogao smanjiti broj paketa na koje se to pravilo odnosi. Da bi se pravilo aktiviralo, moraju biti zadovoljeni svi uvjeti iz zaglavlja pravila, kao da su povezani logičkim operatorom AND.

Akcije koje se mogu navesti u zaglavlju pravila već su opisane u poglavlju o radu u inline načinu rada. Zatim slijedi oznaka protokola, među kojima su trenutno podržani TCP, UDP, IP i ICMP. U budućnosti se planira proširenje na dodatne protokole među kojima su ARP, IGRP, GRE, OSPF, RIP, IPX itd. Sljedeća polja u zaglavlju su četiri oznake od kojih su prve dvije odvojene od druge dvije nizom znakova "->". Navedena polja predstavljaju redom izvorišnu adresu (odnosno mrežu) i pristupna vrata, te odredišnu adresu (odnosno mrežu ili pristupna vrata). Za označavanje IP adresa, mreža i pristupnih vrata koristi se zapis predstavljen u poglavlju o postavljanju Snorta, a kao dodatno pravilo navodi se da nije moguće koristiti simbolička imena, odnosno program nema mogućnosti razrješavanja (eng. *resolving*) imena. Primjer kompletnog zaglavlja pravila prikazan je na sljedećem ispisu:

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 25
```

Zaglavlje pravila zadanog na ovaj način će aktivirati svaki paket koji dolazi s bilo kojih pristupnih vrata i bilo koje adrese izvan lokalne mreže na lokalnu mrežu i pristupna vrata 25/tcp (SMTP protokol). Umjesto operatora "->" koji označava jednosmjernu komunikaciju, može se koristiti i operator "<>" koji označava dvosmjernu komunikaciju. Operator "<-" ne postoji zbog čitljivosti i konzistentnosti pravila, nego je u tome slučaju potrebno okrenuti izvorišnu i odredišnu adresu te koristiti operator "->".

Posebna klasa pravila su dinamička i aktivirajuća pravila. Ovakva pravila uvijek dolaze u paru, odnosno uz svako dinamičko pravilo dolazi i jedno aktivirajuće pravilo. Nijedno od ove dvije vrte pravila ne mogu postojati sama za sebe. Ukoliko se zadovolje uvjeti iz aktivirajućeg pravila, počet će se obrađivati i pripadajuće dinamičko pravilo. Aktivirajuće pravilo ponaša se kao i obično pravilo s akcijom *alert*, osim što ima dodatnu obaveznu opciju koja označava dinamičko pravilo koje treba obraditi ukoliko su zadovoljeni uvjeti iz zaglavlja pravila. Dinamičko pravilo se ponaša kao i pravilo s akcijom *log* osim što se obrađuje samo u slučaju kada ga aktivira aktivirajuće pravilo. Za definiranje ovih vrsta pravila koriste se ključne riječi *activate* odnosno *dynamic*. Primjer korištenja para dinamičkog i aktivirajućeg pravila pokazuje sljedeći ispis:

```
activate tcp !$HOME_NET any -> $HOME_NET 143 (flags: PA; \
         content: "|E8C0FFFFFF|/bin"; activate: 1; \
         msg: "IMAP buffer overflow!")

dynamic tcp !$HOME_NET any -> $HOME_NET 143 \
         (activated_by: 1; count: 50;)
```

Navedeno pravilo otkriva napade prekoračenjem međuspremnika na IMAP protokol i zatim sprema podatke za idućih 50 paketa koji dolaze na IMAP pristupna vrata (143/tcp). Ukoliko je napad uspio vrlo je vjerojatno da će idućih 50 paketa sadržavati korisne informacije o daljnjim postupcima napadača, pa je korisno sakupiti i takve podatke za kasniju analizu.

Postavke pravila mogu se podijeliti na četiri glavne kategorije:

- general (informacije o paketu nevezane uz njegovo zaglavlje ili sadržaj)
- payload (za pregled podatkovnog dijela paketa)
- non-payload (za pregled zaglavlja paketa)
- post-detection (okidači koji se pokreću nakon što je pravilo pokrenuto)

Postavke se međusobno odvajaju korištenjem točke zareza (";"). Najjednostavnija i najčešće korištena opcija koja se može navesti u postavkama pravila je *msg*, a koristi se za navođenje poruke odnosno upozorenja koje će se ispisati u slučaju da je pravilo pokrenuto. Iza ključne riječi slijedi tekst poruke unutar dvostrukih navodnika. Sličnu namjenu i način korištenja ima i ključna riječ *reference*, samo što se ona koristi za navođenje URL-a s detaljnijim informacijama o vrsti napada koji je otkriven. Kod korištenja ove ključne riječi potrebno je navesti jednu od unaprijed definiranih oznaka stranica koje čuvaju podatke o vrstama napada, te identifikacijski broj vrste napada na koji se referenciramo. Na primjer, za referenciranje na napad IDS411 opisan na *arachnids* Internet sjedištu koristi se opcija "`reference:arachnids,IDS411`". Tekst naveden iza oznake stranice slijedi dio koji će se nadodati na URL definiran oznakom stranice. Sljedeća tablica pokazuje dostupne oznake te URL koji se stvara njihovim korištenjem.

Oznaka	URL
bugtraq	http://www.securityfocus.com/bid/
cve	http://cve.mitre.org/cgi-bin/cvename.cgi?name=
nessus	http://cgi.nessus.org/plugins/dump.php3?id=
arachnids	http://www.whitehats.com/info/IDS
mcaffe	http://vil.nai.com/vil/dispVirus.asp?virus_k=
url	http://

Tablica 3.6: Unaprijed definirane oznake na web sjedišta

Ključna riječ *gid* (eng. *generator id*) koristi se za označavanje komponente Snorta koja je stvorila određeni događaj. Na primjer, GID 1 označava pravila, dok su GID brojevi iznad 100 rezervirani za razne pretprocesore i dekodere paketa. Radi izbjegavanja konflikata kod definiranja identifikacijskih brojeva, za vlastita pravila preporuča se korištenje identifikacijskih brojeva iznad 1000000 ili se uopće ne preporuča njihovo korištenje jer navođenje ove ključne riječi nije obavezno. Slično ključnoj riječi *gid* koristi se i ključna riječ *sid*, koja se koristi za definiranje jedinstvenog identifikacijskog broja svakog pravila. Moguće je koristiti i ključnu riječ *rev* za definiranje verzije (revizije) određenog pravila. Kao i kod identifikacijskih brojeva generatora, brojevi manji od 100 rezervirani su za buduću upotrebu, brojevi između 100 i 1000000 rezervirani su za standardna Snort pravila, a brojevi veći od 1000000 dostupni su za korisnički definirana pravila. Korištenje ključnih riječi *gid*, *sid* i *rev* prikazuje sljedeće pravilo:

```
alert tcp any any -> any 80 (content:"BOB"; gid: 100900; sid:1000983;
rev 1;)
```

Konačno, napadi se mogu svrstati u šire klase, a pojedine klase se označavaju ključnom riječju *classtype*. Na primjer, razne vrste skeniranja mogu se svrstati u zajedničku klasu "*port scans*". Svakoj klasi je predodređen i određeni prioritet koji govori o važnosti otkrivenog napada. Prioriteti su visok, srednji, nizak i vrlo nizak. Primjeri nekih klasa napada kao i pridruženi prioriteti prikazani su u sljedećoj tablici.

Klasa napada	Opis	Prioritet
Attempted-admin	Pokušaj pristupa administratorskim ovlastima	Visok
Policy-violation	Potencijalno kršenje zadane sigurnosne politike	Visok
Kickass-porn	Otkriveni pornografski sadržaji	Visok
Shellcode-detect	Otkriveni izvršni kod unutar paketa	Visok
Trojan-activity	Otkriveni zlođudni program unutar paketa	Visok
Attempted-dos	Pokušaj napada uskraćivanjem usluge	Srednji
Web-application	Pristup potencijalno ranjivoj mrežnoj aplikaciji	Srednji
Network-scan	Otkriveni napad skeniranjem pristupnih vrata	Nizak
Unknown	Nepoznati promet	Nizak
Tcp-connection	Otkrivena je nova uspostavljena TCP veza	Vrlo nizak

Tablica 3.7: Snort klase napada

Uz unaprijed definirane klase napada od kojih su neke prikazane u tablici, moguće je definirati i vlastite klase napada. Za definiranje vlastitih klasa napada koristi se izraz *config classification.* Uz ime klase potrebno je navesti jedan od već spomenutih prioriteta. Popis svih već definiranih klasa napada može se pronaći u datoteci *classification.config* unutar direktorija sa ostalim Snort konfiguracijskim datotekama, obično u direktoriju /etc/snort.

Posljednja u nizu opcija za navođenje općih informacija o vrsti napada (*general*) je ključna riječ *metadata,* a namjena joj je navođenje ostalih informacija koje nije moguće opisati korištenjem jedne od prethodno opisanih ključnih riječi. Format vrijednosti koje se navode korištenjem ključne riječi *metadata* moraju poštovati pravilo ključ – vrijednost, odnosno podatak se sastoji od identifikacijske oznake i pridružene vrijednosti.

Klasa opcija *payload* sadrži niz opcija vezanih uz definiranje sadržaja paketa. Najčešće korištena ključna riječ je *content* iza koje slijedi niz znakova koji treba prepoznati. Niz znakova mora biti okružen dvostrukim navodnicima, a ispred njih je moguće navesti i uskličnik koji označava negaciju, tj. pravilo će biti zadovoljeno ako se zadani niz ne pojavi unutar promatranog paketa. Neki specijalni znakovi se unutar niza za prepoznavanje moraju pisati zajedno sa takozvanim *escape* znakom, tj. sa "\". Znakovi koji se moraju tako pisati su dvotočka, točka zarez, backslash i dvostruki navodnik, tj. svi znakovi koji inače imaju neko specijalno značenje. Općenita sintaksa korištenja ključne riječi *content* je dakle:

```
content: [!] "<content string>";
```

Primjerice, sav promet koji dolazi sa interneta na internu mrežu, a koristi HTTP i HTTPS protokole na standardnim pristupnim vratima, može se opisati sljedećim pravilom:

```
alert tcp !$HOME_NET any -> $HOME_NET [80,443] (content:"GET";)
```

Uz ključnu riječ moguće je koristiti i niz modifikatora koji mijenjanju neke od parametara prepoznavanja. Neki od modifikatora su *nocase* (zanemaruje se razlika između velikih i malih slova), *depth* (označava dubinu odnosno broj prvih *n* okteta koje treba pretraživati za zadanim nizom znakova), *offset (*definira odmak od početka paketa i mjesto na kojem treba početi tražiti za zadani niz znakova) itd. Modifikatori se navode iza ključne riječi *content.*

Klasa opcija *non-payload* omogućava ispitivanje raznih vrijednosti unutar paketa koje nisu vezane uz njegov sadržaj. Popis nekih češće korištenih opcija kao i njihova namjena prikazana je u sljedećoj tablici.

Opcija	Opis
ttl	Provjerava time to live vrijednost unutar IP paketa
tos	Provjerava type of service vrijednost unutar IP paketa
flags	Provjerava stanje zastavica unutar TCP zaglavlja (F, S, R, P, A, U)
itype	Provjerava tip poruke protokola ICMP
ack	Provjerava acknowledge brojeve unutar TCP zaglavlja
dsize	Provjerava veličinu podatkovnog dijela IP paketa
window	Provjerava veličinu prozora u TCP protokolu

Tablica 3.8: Non-payload klasa opcija

Na primjer, sve TCP pakete koji iniciraju vezu s vanjske na unutrašnju mrežu može se opisati sljedećom sintaksom:

```
alert tcp !$HOME_NET any -> $HOME_NET any (flags:S)
```

Posljednja klasa opcija je *post-detection* klasa. Korištenjem ključnih riječi iz ove klase opcija moguće je navesti razne akcije koje će se izvršiti nakon aktivacije pravila. Na primjer, korištenjem ključne riječi *logto* možemo specificirati posebnu datoteku dnevnika za određenu klasu paketa. Primjerice, u posebnu datoteku možemo zapisivati samo UDP pakete s određene mreže. Korisna je i ključna riječ *session* koja se koristi za izvlačenje nizova znakova koji se mogu ispisati (eng. *printable characters*) iz sadržaja paketa. Ovo može biti korisno za izvlačenje korisničkih imena i lozinki kod korištenja protokola koji ih šalju u obliku čistog teksta (na primjer FTP, Telnet, rlogin, web aplikacije i drugi). Sljedeći primjer pokazuje pravilo koje će ispisivati sve takve nizove znakova u paketima koji se koriste za razmjenu informacija telnet protokolom:

```
log tcp any any -> any 23 (session:printable;)
```

Važno je i napomenuti da ključna riječ *session* može znatno usporiti rad Snorta, tako da se njezino korištenje ne preporuča u okruženjima gdje je potrebna velika brzina izvršavanja tj. u okruženjima s vrlo velikom količinom prometa ili velikom bazom pravila.

Korištenjem ključne riječi *resp* moguće je zatvoriti TCP vezu ukoliko je aktivirano pravilo koje ju sadrži. Inicijatoru TCP veze mogu se poslati razni odgovori, ovisno o tipu odgovora koji smo naveli uz ključnu riječ.

Tip odgovora	Opis
rst_snd	Šalje paket s upaljenom RST zastavicom na pošiljateljevu adresu
rst_rcv	Šalje paket s upaljenom RST zastavicom na primateljevu adresu
rst_all	Šalje paket s upaljenom RST zastavicom na obje adrese
icmp_net	Šalje pošiljatelju poruku ICMP Network Unreachable
icmp_host	Šalje pošiljatelju poruku ICMP Host Unreachable
icmp_port	Šalje pošiljatelju poruku ICMP Port Unreachable
Icmp_all	Šalje pošiljatelju sve prethodno navedene ICMP poruke

Tablica 3.9: Opcije za zatvaranje TCP veze

Korištenje ove ključne riječi nije omogućeno ako se kod prevođenja Snorta koristi podrazumijevani skup zastavica za prevođenje, nego se mora posebno omogućiti s opcijom `--enable-flexresp`. Korištenjem ključne riječi *resp* također se bitno utječe na brzinu rada kompletnog programa. Također se pojavljuje i opasnost da se uz potencijalne napade prekine i ispravan mrežni promet.

3.7. Dodatni alati

3.7.1. iptables

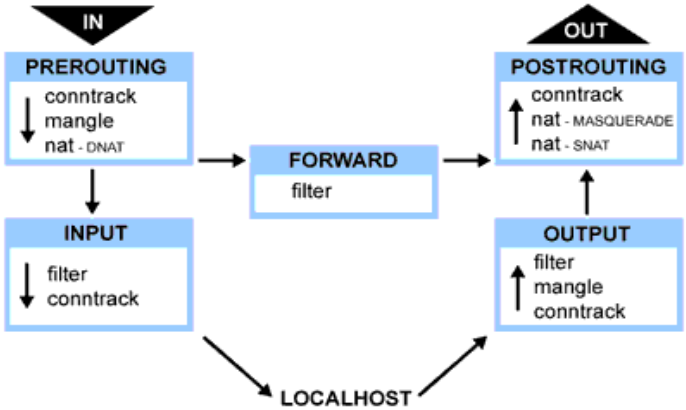
Iako je netfilter/iptables potpuno neovisni projekt od Snorta, vrlo se često koriste zajedno i međusobno nadopunjuju svoje funkcije (npr. rad Snorta u inline načinu). Pošto sigurnosna zaštitna stijena nije primarna tema ovog rada, te pošto je tema netfiltera preopširna da se ovdje u cijelosti opiše, razmotreni su samo osnovna načela netfiltera odnosno iptablesa.

Netfilter je programski okvir (eng. *framework*) u obliku dodataka na Linux jezgru koji omogućuje čitanje mrežnih paketa kao i izvođenje raznih akcija na njima. Najpoznatija komponenta izgrađena povrh tog programskog okvira je program iptables, koji služi kao sigurnosna stijena te može izvršavati dodatne operacije poput translacije adresa (eng. *network address translation*, NAT), praćenja stanja veze (eng. *stateful connection tracking*) te slanja paketa u korisnički prostor (eng. *user space enqueuing*). Iptables je prvenstveno nastao kao nadogradnja starijeg sustava *ipchains*, da bi ga vremenom potpuno zamijenio. Kod korištenja imena iptables potrebno je razlikovati sigurnosnu stijenu izgrađenu povrh netfilter infrastrukture od korisničkog programa istog imena koji se koristi za upravljanje tom istom sigurnosnom stijenom.

Iptables omogućava mrežnim i sistemskim administratorima izradu pravila za rukovanje određenim vrstama mrežnih paketa. Pravila se grupiraju u grupe koje se zovu *lanci* (eng. c*hains*), a lanci se dalje grupiraju u složenije strukture koje se zovu tablice. Svaka od tablica je zadužena za drukčiji način upravljanja mrežnim prometom. Iptables pravila se sastoje od dva dijela:

- dio podudaranja (eng. *packet match*)
- dio akcije (eng. *rule action*)

Dio podudaranja označava na koju vrstu paketa se odnosi navedeno pravilo, dok dio akcije (koji je neobavezan) označava što sa takvim paketima treba raditi. Svaki paket koji dolazi ili odlazi s računala mora proći kroz bar jedan lanac. Paket se tada sekvencijalno provjerava sa pravilima unutar pojedinog lanca, Ukoliko paket zadovoljava dio podudaranja nekog od pravila, akcija tog pravila određuje što će se dalje raditi s tim paketom. Ako paket dođe do kraja lanca bez da je zadovoljio bilo koje od pravila unutar tog lanca, na njega se primjenjuje takozvana *politika lanca* (eng. *chain policy*). Politiku lanca moguće je navesti samo za unaprijed definirane lance. Za podrazumijevanu *filter* tablicu unaprijed su definirani lanci INPUT, OUTPUT i FORWARD. Korisnik ima mogućnost kreirati i vlastite lance. U slučaju da paket dođe do kraja lanca koji nije unaprijed definiran, nastavit će s obradom pravila unutar lanca iz kojeg je došao u taj lanac. Međusobna ovisnost tablica, lanaca i toka paketa prikazana je na slici 3.2.



Slika 3.2: Iptables sustav

Praćenje veza (eng. *connection tracking*) je jedna od važnijih komponenti izgrađena povrh netfilter programskog okvira. Koristeći praćenje veza, jezgra operacijskog sustava može pratiti logičke mrežne veze i klasificirati pročitane paket u jednu od klasa povezanih s tom vezom. Informacije o praćenju veza mogu se koristiti ili u svrhu izrade filtra paketa sa stanjem (odnosno sigurnosne stijene), ili za prevođenje adresa (NAT). Paketi mogu biti klasificirani u jedno od sljedećih četiri stanja:

- *new,* paket pokušava kreirati novu vezu
- *established,* paket je dio već uspostavljene veze
- *related,* paket otvara novu vezu, ali je povezan s nekom od već postojećih veza (npr. upravljačka i podatkovna veza kod protokola FTP)
- *invalid,* paket nije dio već uspostavljene veze niti je moguće kreirati novu vezu

Stanje veze kod netfiltera odnosno njegovog dijela za praćenje veza je potpuno neovisno od TCP stanja veze. Npr. ukoliko na primljeni SYN paket računalo odgovori sa SYN/ACK, TCP veza još nije u potpunosti uspostavljena, ali u connection tracking komponenti ipak će biti označena kao uspostavljena, odnosno established. Također, moguće je i pratiti stanje za protokole koji po svojoj prirodi nemaju stanja, primjerice UDP protokol. Stanje veze se često koristi za definiranje pravila, a najčešća je praksa da se prvo za dolazni promet dopuste samo established i related stanja, dok se new stanje dopušta samo za posebno propuštene grupe prilaznih vrata.

Iptables je i korisnička aplikacija koja se koristi za definiranje pravila, odnosno upravljanje tablicama, lancima i pravilima. Primjer korištenja iptablesa može se demonstrirati jednostavnim primjerom:

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT
  --to-ports 8080
```

Navedeno pravilo će sav dolazni promet na pristupna vrata 80 (http) preusmjeriti na pristupna vrata 8080. Opcijom *-t* navodi se tablica koju uređujemo, u ovom slučaju uređujemo tablicu za NAT, a ukoliko se ova opcija izostavi onda se podrazumijeva korištenje tablice *filter*. Opcijom *-p* odabire se vrsta protokola na koje se pravilo odnosi, a opcijom *--dport* označavamo da se pravilo odnosi na pakete koji za odredišna pristupna vrata imaju naznačena 80/tcp. Opcijom *-j* odabiremo akciju koja će se izvršiti ukoliko

paket zadovoljava sve prethodno navedene uvjete. Moguće je koristiti velik broj dostupnih akcija poput DROP (ispuštanje paketa), REJECT (odbijanje paketa), REDIRECT (preusmjeravanje odredišnih pristupnih vrata paketa na lokalnom računalu) itd. Ovisno o odabranoj akciji, moguće je odabrati čitav niz dodatnih opcija vezanih specifično uz tu akciju.

3.7.2. Ostali alati

Kao što je već prije navedeno, dobra osobina Snorta je njegova mogućnost da preko izlaznih modula surađuje sa čitavim nizom drugih alata. Dodatni alati mogu imati razne namjene, poput održavanja baze potpisa ažurnom, alata za statističku analizu prikupljenih podataka, grafički prikaz podataka itd. Radi kratkoće ovdje su opisani samo najčešće korišteni dodatni alati.

Jedan od najčešćih problema i zadataka kod održavanja Snort okruženja je ažuriranje baze potpisa napada (baze u širem smislu te riječi, pošto se popisi napada čuvaju u običnim tekstualnim datotekama). Najčešće korišteni alat u ovu svrhu je *oinkmaster*. Oinkmaster je skripta napisana u programskom jeziku Perl i koristi se za niz operacija vezanih uz održavanje Snort pravila. Moguće je preuzimati službeno podržana pravila od strane Sourcefirea, ili koristiti neki od neslužbenih izvora poput BleedingSnort. Dodatne mogućnosti programa uključuju automatsko stvaranje identifikacijskog broja pravila (SID) za pravila koja ih nemaju, uklanjanje dvostrukih pravila, grupno (eng. b*ulk*) mijenjanje pravila korištenjem regularnih izraza, omogućavanje i onemogućavanje određenih skupova pravila, stvaranje sigurnosnih preslika itd. U većini distribucija standardna instalacija Snorta obuhvaća i oinkmaster. Korištenje programa je vrlo jednostavno i većina parametara se ne mora zadavati u komandnoj liniji za vrijeme pokretanja nego se može navesti u konfiguracijskoj datoteci *oinkmaster.conf*. Preuzimanje i osvježavanje pravila tada se može obaviti jednostavnom naredbom:

```
# oinkmaster -o /etc/snort/rules
```

Prva operacija koja se izvodi je preuzimanje novih pravila korištenjem programa *wget*. Pravila su dostupna na raznim repozitorijima u obliku *tarball* (*.tar.gz*) arhiva. Nakon što se otpakira arhiva s novim pravilima, skripta provjerava jednoznačnost pravila te ih pretražuje za postojanje dvostrukih zapisa sa starim pravilima u navedenom direktoriju. Tijekom procesa stvaraju se i sigurnosne preslike starih pravila. Nakon izvođenja skripte ispisuju se informacije o izvršenim akcijama. Korištenjem opcije *-c* moguće je izvesti i simulaciju navedenih akcija, bez zapisivanja ikakvih promjena na disk.

Sljedeći često korišteni dodatni program je *Barnyard*. Barnyard se koristi u situacijama i okruženjima gdje je brzina izvođenja kritični faktor kod izvođenja programa. Program se koristi za naknadnu analizu Snortovih datoteka dnevnika zapisanih u unificiranom (eng. *unified*) formatu te pretvaranje zapisa u neki pogodniji oblik, npr. zapisivanje u relacijsku bazu podataka. Glavna prednost ovakvog pristupa pred direktnim korištenjem npr. Snort *database* izlaznog modula jer činjenica da je zapisivanje unificiranog zapisa vrlo brzo i nije procesorski zahtjevno. Zapisivanje datoteka dnevnika u unificiranom obliku postavlja se u konfiguracijskoj datoteci *snort.conf*:

```
output alert_unified: filename snort.alert, limit 128
output log_unified: filename snort.log, limit 128
```

Ukoliko je Snort već bio pokrenut, potrebno ga je ponovno pokrenuti jednom od sljedećih naredbi (ovisno o distribuciji koja se koristi):

```
# /etc/init.d/snort restart
# invoke-rc.d snort restart
# service snort restart
```

Sve relevantne opcije za program Barnyard navode se u konfiguracijskoj datoteci barnyard.conf. Opcije za specificiranje načina zapisivanje izlaza programa istovjetne su opcijama Snort izlaznih modula. Dodatno je potrebno paziti da se Barnyard pokrene prije nego što je pokrenut sam Snort.

Najčešće korišteni dodatni program uz Snort je ACID (eng. *Anaylsis Console for Intrusion Databases*). ACID je web aplikacija pisana u PHP-u, a glavna joj je namjena statistička analiza i vizualizacija podataka primljenih od Snorta. Za korištenje programa potrebno je podesiti kompletno LAMP okruženje, odnosno potrebno je imati instaliran web poslužitelj s PHP modulima, te podesiti Snort za spremanje upozorenja u relacijsku bazu podataka (obično MySQL ili PostgreSQL). Program je moguće koristiti i na operacijskom sustavu Windows, a u tom slučaju moguće je koristiti i Microsoft SQL server. Ključne mogućnosti programa *ACID* su sljedeće:

- sučelje za pretraživanje spremljenih podataka
- detaljni pregled događaja po svim relevantnim parametrima
- stvaranje grafova, statistike i dijagrama
- upravljanje događajima

Uz analizu Snort zapisa, ACID može analizirati i zapise programa ipchains, iptables te ipfw. Za korištenje ovih zapisa potrebno je dodatno instalirati program *logsnorter*. ACID je i standardni dio već prije spominjanoga objedinjenog NIDS rješenja Honeynet. Na osnovama izvornog koda ACID-a nastao je i srodni projekt BASE (eng. *Basic Analysis and Security Engine*).

4. Praktični rad

4.1. Opis problema

Cilj praktičnog dijela rada je izraditi programsko riješenje koje će međusobno kombinirati mogućnosti otkrivanja i prepoznavanja napada zajedno s mogućnostima sprječavanja njihovog daljnjeg pojavljivanja. Razvijeni program mora otkrivati mrežne napade s izvorom na lokalnoj ili vanjskoj mreži, te korištenjem pogodnih alata zabraniti njihovo daljnje pojavljivanje na mreži. Djelovanje po svojoj funkcionalnosti mora biti blisko sustavima za sprječavanje napada sa mreže. Zabranjivanje određenih vrsta napada mora se moći navesti u odgovarajućoj datoteci sa sigurnosnom politikom (eng. *policy file*) ili se može definirati u trenutku kada je određeni napad otkriven za vrijeme rada programa, obavještavajući korisnika. Koristeći unaprijed zadane datoteke sa sigurnosnom politikom, sustav može automatski zabranjivati određene klase napada. Nadalje, sustav mora imati mogućnosti neprekidnog praćenja više od jednog računalnog sustava. U tom smislu potrebno je realizirati raspodijeljeni sustav čijim se funkcijama upravlja putem centralnog nadzornog programa. Razvijeno rješenje mora biti neosjetljivo na prekid rada programa, odnosno nakon ponovnog pokretanja moraju biti zapamćene sve postavke koje su se koristile kod prethodnog pokretanja programa. Cijeli sustav mora biti potpuno upravljiv putem odgovarajućih grafičkih korisničkih sučelja. Korištenjem grafičkog korisničkog sučelja moguće je upravljati svakom komponentom sustava, te pregledavati sakupljene podatke i na temelju njih poduzeti potrebne akcije. Također, u program moraju biti ugrađene mogućnosti pravovremenog obavještavanja korisnika o svim napadima koji su u tijeku, po uzoru na mogućnosti prepoznavanja napada sličnih popularnih programa poput raznih *personal firewalla*. Sve akcije razvijenog sustava moraju se zapisivati u odgovarajuće datoteke dnevnika radi mogućnosti naknadne analize rada programa. Same poruke u datotekama dnevnika klasificiraju se u nekoliko različitih vrsta radi lakše analize pojedinih klasa problema.

Ukratko, razvijeno rješenje mora u potpunosti podržavati sljedeće mogućnosti:

- prepoznavanje napada sa mreže
- sprječavanje napada sa mreže
- mogućnost raspodijeljenog rada
- upravljivost putem grafičkog korisničkog sučelja
- analiza prikupljenih podataka

Kod razvoja programa koristili su se isključivo slobodni alati i tehnologije, odnosno programi otvorenog programskog kôda. Konačno riješenje objavljeno je pod imenom *Netkeeper* na stranici Sourceforge gdje se nalazi čitav niz slobodno dostupnih programa raznih namjena. Kod objave programa odabrana je GNU GPL (eng. *General Public Licence*) licenca, koja omogućava slobodnu distribuciju programa, njegovog izvornog kôda, te daje mogućnost izmjene programskog kôda te daljnje distribuiranje s tim izmjenama, sve dok je program i dalje objavljen pod GPL licencom.

Arhitektura cijelog programa, građa i funkcija pojedinih komponenti kao i opis njihove međusobne međuovisnosti prikazan je u poglavlju o arhitekturi programskog rješenja. Uz opis arhitekture objašnjena je i kratka pozadina iza korištenih rješenja, te obrazloženje zašto je odabran baš takav pristup. Poglavlje o korištenim tehnologijama opisuje upotrebljavane alate za razvoj, kao i razne protokole te programska rješenja koja su se koristila za realizaciju pojedinih komponenti sustava.

4.2. Izrada programa

4.2.1. Arhitektura rješenja

Program *Netkeeper* je po svojoj arhitekturi *klijent – poslužitelj* aplikacija (eng. *client server application*). Dijelovi ove arhitekture nazivaju se *čvorovi*, a pojedini čvorovi međusobno mogu komunicirati razmjenom poruka putem mreže. Najjednostavniji tip ovakve arhitekture je sustav sa samo dva različita čvorova. Ta dva čvora se nazivaju *klijentski sustav* te *poslužiteljski sustav*. Uloga poslužiteljskog sustava je obrada informacija, te pružanje obrađenih informacija i usluga drugim elementima sustava. Jedan poslužitelj može opsluživati jedna ili više klijentskih sustava. Klijentski sustavi su drugi dio ovakve arhitekture i služe kao posrednik u interakciji korisnika s uslugama koje pruža poslužitelj. Klijent prima upute od korisnika, šalje zahtjeve ili upute k poslužitelju, čeka da poslužitelj obradi podatke ili završi zadanu akciju, te zatim prima obrađene podatke od poslužitelja i vraća ih nazad korisniku. Klijent je zadužen i za prezentaciju obrađenih podataka u obliku pogodnom korisniku sustava. Danas je u većini slučajeva klijent web preglednik, a u manjem broju slučajeva zasebna aplikacija. Primjeri raznih klijent – poslužitelj mogu se naći na svakom koraku, npr. sustavi za upravljanje relacijskim bazama podataka, mrežne igre, poslužitelji elektroničke pošte itd.

Ovisno o zadacima koje obavljaju, razlikuju se dvije osnovne vrste klijenata:

- tanki klijent
- debeli klijent

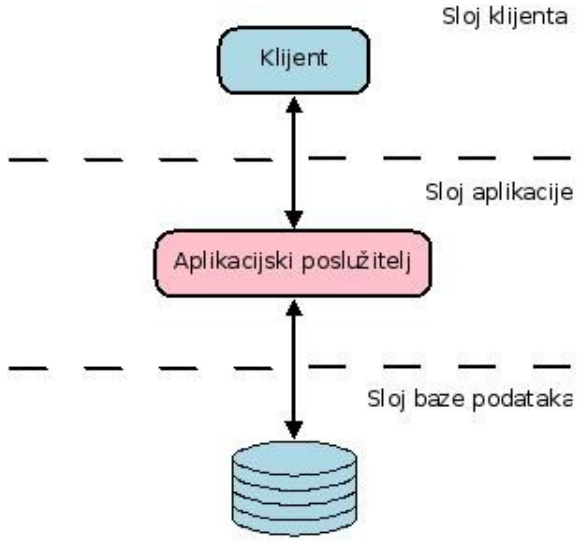
Tanki klijent (eng. *thin client*) obavlja samo mali dio funkcionalnosti cijelog sustava, dok je većina posla kao i kompletna logika programa prebačena na poslužitelj. U ovom slučaju zahtjeva se veća procesorska snaga na strani poslužiteljskog sustava, dok klijenti mogu biti slabija računala ili aplikacije sa tek osnovnim mogućnostima. Primjer ovakvih sustava su web aplikacije, a u takvome kontekstu tanki klijent je web preglednik. Uobičajena uloga tankog klijenta je prezentiranje rezultata obrade na strani poslužitelja. Za razliku od tankog klijenta, debeli klijent (eng. *fat client*) ima puno veće mogućnosti obrade podataka. Dio programske logike u ovakvom sustavu može biti izveden i na strani klijenta. Dodatna dobra karakteristika debelog klijenta je smanjenje zahtjeva za resursima na strani poslužitelja.

Program *Netkeeper* koristi izmijenjeni klijent - poslužitelj model arhitekture. Razlika od prethodno opisanog osnovnog modela je postojanje više poslužiteljskih čvorova. Normalno korištenje razvijenog programa uključuje korištenje više poslužiteljskih čvorova, kao i korištenje jednog ili više klijentskih čvorova. Pojedini poslužiteljski sustavi primaju upute

od klijentskih sustava, poduzimaju odgovarajuće akcije te vraćaju tražene informacije nazad u klijentski sustav. *Neetkeeper* poslužiteljski sustav ima usko specifičnu namjenu otkrivanja napada te njihovog sprječavanja. Poslužiteljski sustavi ovakvog uskog područja namjene koji u svojem radu koriste i druge programe te su upravljivi od strane klijenata se nazivaju *agentima.* U kontekstu razvijenog programa također se dalje koristi naziv agent.

Po svojoj funkcionalnosti, klijentski sustav programa Netkeeper je debeli klijent, tj. uz mogućnosti prezentacije rezultata dobivenih od agenata, posjeduje i mogućnost obrade podataka i upravljanja drugim komponentama sustava. Korištenjem Neetkeeper klijenta moguće je u potpunosti upravljati radom agentskih sustava. Za upravljanje sustavom i prezentaciju podataka, klijenti koriste grafičko korisničko sučelje. Komunikacija klijenata s agentima ostvarena je korištenjem XMLRPC protokola. XMLRPC protokol za razmjenu poruka detaljnije je opisan u poglavlju o korištenim tehnologijama.

Postojanje više poslužitelja, odnosno agenata, nije jedina razlika arhitekture realiziranog programa od klasične klijent – poslužitelj arhitekture. Umjesto u kontekstu vrsta čvorova u sustavu, često se govori o slojevima aplikacije (eng. t*iers*). Na ovaj način, klasični sustav se opisuje kao dvoslojni (eng. *two tier architecture*). Slojevi u klasičnom sustavu su sloj poslužitelja i sloj klijenta, a funkcionalnost pojedinih slojeva uglavnom odgovara funkcionalnosti klijentskih odnosno poslužiteljskih čvorova. Dvoslojni sustav se u praksi koristi dosta rijetko, a veći naglasak se stavlja na troslojnu arhitekturu (eng. *three tier architecture*). Troslojnu arhitekturu prikazuje slika 4.1.

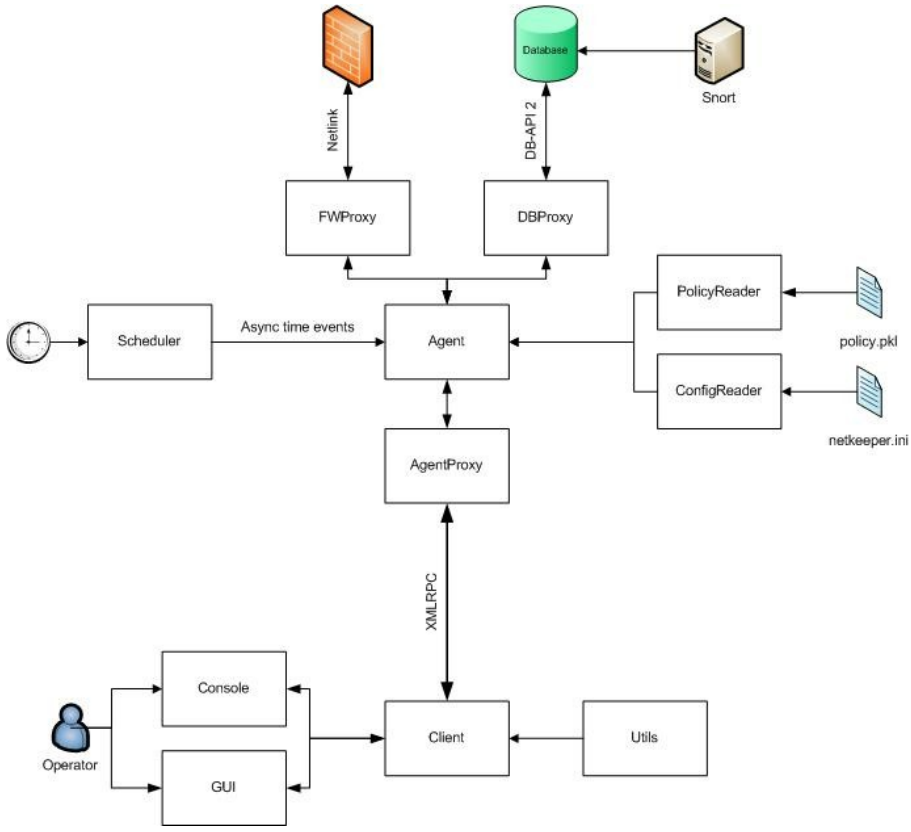


Slika 4.1: Troslojna klijent - poslužitelj arhitektura

Troslojna arhitektura se razlikuje od dvoslojne arhitekture po tome što je sloj aplikacije dodatno podijeljen na dva sloja, sloj baze podataka (eng. *database tier*) i sloj aplikacije (eng. *application tier, middleware tier*). Uloga sloja baze podataka je pohranjivanje svih podataka neovisno o aplikaciji koja ih koristi te pružanje standardnog sučelja za pristup i mijenjanje podataka. Aplikacijski poslužitelj na sloju aplikacije sadržava logiku programa koji rukuje tim podacima na sloju baze podataka te ih šalje klijentima. Sloj klijenta ponegdje se naziva i prezentacijskim slojem te ima ulogu prikaza podataka te pružanja

sučelja za upravljanje cijelim sustavom. Troslojni model se često pogrešno uspoređuje i poistovjećuje sa takozvanim *model – pogled – kontroler* konceptom (eng. *model view controller*, MVC). Temeljna razlika ova dva modela je što se kod troslojnog modela nikad ne može dogoditi izravna komunikacija sloja klijenta sa slojem baze podataka. MVC model je stariji i koristio se uglavnom na sustavima koji su u cijelosti bili implementirani na jednom računalu, dok se u današnjim distribuiranim sustavima pretežno koristi troslojni model. Potrebno je i napomenuti da je u troslojnom modelu moguće da i sloj baze podataka sadrži dio logike za upravljanje podacima. U takvim slučajevima podaci se obrađuju i mijenjaju korištenjem specijaliziranih programskih jezika na razini baze podataka, npr. korištenjem pohranjenih procedura (eng. *stored procedures*) jezika SQL. Obrada podataka na sloju baze podataka je obično jednostavnija od obrade podataka na sloju aplikacije, dodatno rasterećuje aplikacijski poslužitelj i pridonosi ukupnoj modularnosti sustava. U većini troslojnih sustava koristi se upravo pristup s obradom podataka na sloju baze podataka korištenjem pohranjenih procedura.

Program *Netkeeper* također koristi troslojni model. Snort pohranjuje poruke upozorenja unutar baze podataka. Snort pristupa bazi podataka putem standardnih izlaznih modula. Za sustav upravljanja relacijskom bazom podataka u programu se koristi program PostgreSQL, ali uz manje prepravke program se može koristiti uz gotovo svaki sustav za upravljanje relacijskim bazama podataka podržan od strane Snorta. Podaci spremljeni unutar baze podataka se obrađuju unutar agenta, dakle ne koriste se pohranjene procedure. Podaci se između agenta i baze podataka izmjenjuju korištenjem DB-API 2.0 protokola. Važno je napomenuti da se baza podataka ne mora nalaziti na računalu na kojem je instaliran *Netkeeper* agent. Sve tri komponente *Netkeeper* sustava (klijent, agent i baza podataka) mogu se nalaziti na različitim računalima. Cjelokupnu arhitekturu razvijenog rješenja prikazuje slika 4.2.



Slika 4.2: *Netkeeper* arhitektura i objekti

Na slici navedena engleska imena pojedinih komponenti odgovaraju imenima klasa u programskom kôdu. Na strelicama je, gdje je to moguće, prikazan protokol kojim se izmjenjuju podaci između pojedinih komponenti sustava.

Centralnu ulogu u *Netkeeper* sustavu imaju agenti. Agent je obavezna komponenta sustava, tj. mora biti pokrenuta u svakom trenutku. Uloga agenta je otkrivanje napada i postavljanje pravila sigurnosne stijene za sprječavanje daljnjih napada. Za rad agenta koriste se dodatni programi, Snort za otkrivanje napada, odnosno iptables za upravljanje pravilima sigurnosne stijene. Nadziru se i odlazni i dolazni mrežni paketi. Agent se sastoji od nekoliko komponenti:

- centralni dio agenta (eng *agent engine*)
- sigurnosna politika
- proxy poslužitelj
- proxy baze podataka
- proxy sigurnosne stijene
- generator događaja (eng. *scheduler*)

Kontrolu nad zabranjivanjem odnosno propuštanjem određenih mrežnih paketa korisnik ima korištenjem sigurnosne politike agenta. Sigurnosna politika može biti jednaka za sve

korištene agente, ali i svaki pojedini agent može imati vlastitu sigurnosnu politiku. Sigurnosna politika može se definirati na tri različita načina:

- ručno
- poluautomatski
- automatski

Ručno definiranje sigurnosne politike dostupno je korištenjem odgovarajućeg korisničkog sučelja u klijentu. Poluautomatsko definiranje dostupno je putem skakajućih poruka (eng. *popup messages*) nakon otkrivanja nove vrste napada za koju još nije definirana sigurnosna politika. Automatsko definiranje sigurnosne politike dostupno je putem specijalne akcije. Sigurnosna politika sastoji se od niza pravila. Svako pravilo sastoji se od sljedećih komponenti:

- izvorišna adresa
- odredišna adresa
- potpis
- zadržavanje
- akcija

Izvorišna i odredišna adresa određuju polazište odnosno odredište paketa na koje se pravilo odnosi. Potpis je naziv Snort pravila koje će, ukoliko je otkriveno, aktivirati pravilo u kojem se nalazi. Zadržavanje (eng. *rule retention*) je vremenski parametar koji govori koliko dugo treba zadržati pravilo prije nego što postane nevažeće, tj. prije nego što se obriše iz liste pravila. Kod navođenja izvorišne i odredišne IP adrese, kao i kod navođenja potpisa, može se pojaviti posebni identifikator *any* koji će biti zadovoljen za bilo koju ulaznu vrijednost. U kontekstu IP adresa, riječ *any* označava mrežu `0.0.0.0/0`, a u kontekstu Snort pravila označava bilo koje pravila. Zadnji parametar koji označava akciju, važan je samo ukoliko su zadovoljena prva tri uvjeta u pravilu, odnosno adrese i potpis. Ukoliko su zadovoljena svi prethodno navedeni uvjeti program će za pakete koji zadovoljavaju to pravilo poduzeti akciju navedenu tim parametrom. Dostupne su sljedeće akcije:

- blokiranje pošiljatelja
- blokiranje primatelja
- blokiranje pošiljateljeve mreže
- blokiranje primateljeve mreže
- blokiranje po potpisu

Blokiranje pošiljatelja odnosno primatelja blokirati će sve daljnje pakete s određene adrese. Blokiranje pošiljatelja obično se koristi nakon što je otkriven napad iz vanjske na unutrašnju mrežu, a blokiranje primatelja kod otkrivanja nedozvoljenih aktivnosti s unutrašnje mreže (npr. pristup poslužitelju s pornografskim sadržajem). Na sličan način funkcioniraju i akcije blokiranja pošiljateljevih odnosno primateljevih mreža, samo što se u ovom slučaju koriste mrežni rasponi (CIDR zapis) umjesto pojedinačnih IP adresa. Akcija

blokiranja po potpisu (unutar programa označeno kao akcija ADD_RULE) se ponaša kao dinamičko pravilo. Ukoliko je nekom potpisu pridružena akcija blokiranja po potpisu, svaki puta kada se otkrije taj potpis, program će automatski blokirati adresu pošiljatelja paketa koji je stvorio događaj.

Centralni dio agenta (*engine*) je glavna komponenta koja djeluje kao poveznički dio za sve ostale komponente agenta. Na slici 4.2 ova komponenta predstavljena je klasom `Agent`. Primjerice, uloga ove komponente je da svakih nekoliko sekundi nadzire bazu podataka za pojavljivanjem novih događaja, te ukoliko je potrebno obavijesti korisnika da poduzme odgovarajuću akciju. Ukoliko već postoji sigurnosna politika za događaj, moguće je i djelovanje bez znanja korisnika.

Komunikacija klijenta s agentom odvija se posredstvom proxy poslužitelja agenta. Ova komponenta definira sučelje za pristup putem kojeg klijentska komponenta može pristupati podacima unutar baze podataka odnosno upravljati akcijama cijelog sustava. Komponenta je također zaslužna za prosljeđivanje poziva (delegiranje) k ostalim komponentama agenta, poput proxya sigurnosne stijene i baze podataka. Sve poruke koje dolaze ili odlaze od proxy poslužitelja razmjenjuju se korištenjem XMLRPC protokola. Uz proxy poslužitelj zadužen za komunikaciju s klijentom, postoje još dvije proxy komponente agenta. Prva komponenta je zadužena za komunikaciju s bazom podataka, odnosno pruža jednostavno programsko sučelje čime se izbjegava složenost ručnog pisanja SQL upita. Druga dodatna proxy komponenta je zadužena za pružanje jednostavnog sučelja za programsko definiranje iptables pravila. U protivnome bi svaki puta bilo potrebno ručno kreirati pravila te pozivati program iptables. Proxy sigurnosne stijene također posjeduje potrebne strukture podataka koje omogućuju što manju potrebu komunikacije sa samim programom iptables. Programsko rješenje koristi posebne objekte za definiranje iptables pravila i lanaca.

Generator događaja (eng. *scheduler*) zadužen je za stvaranje događaja u pravilnim vremenskim razmacima. Funkcija ovog periodičkog signala je signaliziranje agentu da mora provjeriti bazu podataka za prisustvo novih događaja. Vremenski razmak se može definirati u konfiguracijskoj datoteci agenta. Događaji se generiraju unutar zasebne programske niti (eng. *thread*) korištenjem standardnih modula programskog jezika Python.

4.2.2. Funkcionalna analiza

Nakon pokretanja programa i podizanja svih potrebnih podsustava, prva akcija koja se obavlja je čišćenje svih prethodno definiranih pravila sigurnosne stijene. Zatim se učitava prethodno spremljena datoteka s definiranom sigurnosnom politikom te se postavljaju pravila sigurnosne stijene da bi odgovarala pročitanim pravilima unutar sigurnosne politike. Većinu vremena agent provodi u stanju čekanja (eng. *idle*) stanju. Postoje dvije vrste događaja koje mogu prekinuti stanje čekanja:

- XMLRPC poziv od strane klijenta
- događaj stvoren od strane generatora događaja

XMLRPC promet između klijenta i agenta odvija se u dva smjera. Klijent poziva udaljene procedure na agentu, a pozivi dolaze do proxya agenta koji ih dalje prosljeđuje odgovarajućoj komponenti. U većini slučajeva agent vraća neki rezultat izvođenja u obliku

XMLRPC odgovora. Ukoliko akcija nema povratne vrijednosti, klijentu se vraća vrijednost `True` pošto XMLRPC standard propisuje da se nakon poziva uvijek mora vratiti nekakav odgovor.

Nakon što generator događaja stvori događaj, prvi korak je zaključavanje posebnih struktura podataka da bi se spriječilo ugnježđivanje obrade događaja. U protivnome bi se mogla dogoditi situacija da obrada događaja traje dulje nego što traje zadani vremenski interval između dva događaja stvorena od strane generatora događaja. Sljedeći korak je korištenje poziva proxya baze podataka za ispitivanje novih događaja stvorenih od strane Snort IDS-a. Ako je otkriveni novi događaj, agent provjerava učitana pravila unutar sigurnosne politike. Ukoliko se nađe postojeće pravilo koje zadovoljava uvjete u jednom o pohranjenih pravila, poduzima se akcija navedena u tom pravilu, što će rezultirati dodavanjem novog pravila sigurnosne stijene. U protivnome se postavljaju zastavice da je otkriveni novi napada, čime se šalje obavijest klijentima i traži definiranje novog pravila sigurnosne politike. Prekidom rada programa program prazni sve međuspremnike i sprema trenutno važeće postavke tako da budu dostupne pri slijedećem pokretanju programa.

Rad sustava otporan je na ispade nekih od komponenti. Primjerice, jedine komponente koje su obavezne za ispravno funkcioniranje programa su agent i sigurnostna zaštitna stijena (iptables). U slučaju pada baze podataka, Snort će nastaviti sa radom do prvog događaja koji će htjeti zapisati u bazu podataka. Ukoliko je u tom trenutku baza podataka spuštena, Snort će prekinuti s radom, ali agent nastavlja sa radom. U slučaju prestanka dostupnosti baze podataka, agent također ne može vraćati podatke klijentu, ali i dalje može obavljati operacije postavljanja sigurnosne politike, odnosno akcije postavljanja pravila na sigurnosnoj stijeni. Za rad programa također nije nužno postojanje pokrenutog klijentskog programa. Važno je napomenuti da sve opisano vrijedi za stacionarno stanje programa, u kojemu su uspješno incijalizirane sve komponente. Kod pokretanja agenta ipak se provjerava dostupnost svih sustava, što uključuje i provjeru mogućnosti spajanja na bazu podataka, mogućnost korištenja iptables programa te niz drugih provjera vezanih uz instanciranje raznih Python objekata.

Dodatna napomena vezana uz funkcionalnost programa je vezana iz a program iptables. Iako je moguće ručno dodavati vlastita pravila sigurnosne stijene, takve akcije se ne preporučaju jer mogu spriječiti ispravan rad programa. Kod pokretanja agenta brišu se sva postojeća pravila sigurnosne stijene.

4.2.3. Korištene tehnologije

Za implementaciju prethodno opisanog programskog rješenja korišten je programski jezik Python. Python je interpretirani, dinamički i objektno orijentirani programski jezik opće namjene. Glavne značajke jezika su nepostojanje striktno definiranih tipova podataka (eng. *weakly typed*), mogućnost korištenja nekoliko programskih paradigmi (objektno orijentirana, funkcijska i proceduralna) te bogata standardna biblioteka modula. Programi pisani u programskom jeziku Python mogu se izvoditi na svakome operacijskom sustavu za koji je napisan Python interpreter. U slučaju programa razvijenog u sklopu ovog diplomskog rada, program je ograničen na izvođenje na operacijskom sustavu Linux jedino

iz razloga što koristi netfilter odnosno iptables infrastrukturu za definiranje pravila sigurnosne stijene.

Jedna od glavnih dobrih karakteristika programskog jezika Python je veliki broj već gotovih modula s objektima za gotovo svako moguće područje primjene. Specifično za program Netkeeper, bilo je moguće upotrijebiti čitav niz gotovih modula za izvođenje mrežne komunikacije, pristup relacijskim bazama podataka, te pokretanje programa u pozadini ili u obliku zasebne niti ili Unix demona (eng. *daemon*). Kod implementacije raznih objekata agenta, korišteni su sljedeći objekti iz standardnih Python modula:

- `ConfigParser`
- `Scheduler`
- `Subprocess`
- `LogWriter`
- `SimpleXMLRPCServer`

Objekt `ConfigParser` koristi se za zapisivanje i čitanje kofiguracijskih datoteka zapisanih u takozvanom *ini* formatu. Ini datoteke sastoje se od sekcija i članova tih sekcija. Članovi sekcije sastoje se od naziva člana, te od njegove vrijednosti, Naziv i vrijednost člana sekcije međusobno su odvojeni jednakošću ("="). Unutar datoteke moguće je koristiti komentare navođenjem točke zareza. Sljedeći primjer pokazuje konfiguracijsku datoteku programa *Netkeeper* u ini formatu:

```
[Agent]
Address = localhost
Port = 6585
Timeout = 5
WorkingDir = /tmp
LogFile = agent.log

[iptables]
Path=/sbin/iptables

[Database]
dbhost = localhost
dbname = snortdb
dbuser = snort
dbpass = snort
```

Sekcija *Agent* koristi se za definiranje općih postavki agenta poput adrese i pristupnih vrata na kojima će agent slušati, radnog direktorija, datoteke dnevnika i vremenskog intervala za stvaranje sinkronizacijskog događaja. Sekcija iptables ima samo jedan član koji se koristi za definiranje staze do programa iptables. Posljednja sekcija, *Database,* koristi se za definiranje parametara potrebnih za spajanje na bazu podataka u koju Snort zapisuje rezultate otkrivanja napada. Za čitanje i pisanje u ovako definiranu datoteku unutar objekta `ConfigParser` koriste se jednostavne metode poput `get()` i `write()`.

Sljedeći korišteni objekt je `Scheduler`. Ovaj objekt je sastavni dio standardnog Python modula time koji sadrži ostale funkcije vezane uz datume i vrijeme. Korištenje navedenog objekta pokazuje sljedeći primjer:

```
scheduler.enter(delay, 1, event_dispatcher, (event, ))
```

Prvi parametar u pozivu govori o vremenskom intervalu između dva događaja koji će biti stvoreni od strane objekta. Nakon što istekne brojač, tj. navedeni vremenski interval dođe do nule, pozvat će se funkcija navedena trećim parametrom. Četvrti parametar funkcije proslijedit će se kao argument funkcije koja se poziva istekom navedenoga vremenskog intervala. Navedeni objekt se poziva unutar zasebne programske niti, tako da ne utječe na rad drugih komponenti. Nakon poziva *dispatcher* funkcije, program obavlja već prije navedene funkcije vezane uz generirani događaj.

Modul LogWriter koristi se za zapisivanje datoteka dnevnika u koje se spremaju informacije o radu agenta. Umjesto ručnog zapisivanja raznih formata, moguće je koristiti standardni način zapisivanja s vremenskom oznakom i jednom od već unaprijed definiranih klasa zapisa:

- debug
- info
- warning
- error
- critical

Objekt `Subprocess` iz standardnog modula `popen` koristi se za pokretanje vanjskih naredbi, odnosno naredbi ljuske. U slučaju programa *Netkeeper*, koristi se za pozivanje naredbe iptables.

Modul `SimpleXMLRPCServer` koristi se u implementaciji objekta `AgentProxy`. Svrha ovog objekta je podići XMLRPC poslužitelj na zadanoj adresi i pristupnih vratima, te prosljeđivati pozive posebnim *callback* funkcijama napisanim od strane korisnika. XMLRPC je vrlo jednostavan protokol, a koristi se za pozivanje udaljenih procedura. Za razliku od puno kompleksnijih protokola, opis XMLRPC protokola je vrlo kratak i praktički se može opisati na samo nekoliko stranica čistog teksta. Za kodiranje sadržaja paketa koristi se XML zapis, a kao prijenosni protokol koristi se HTTP. Umjesto XMLRPC-a može se koristiti i moćniji protokol SOAP, ali se zbog svoje jednostavnosti i lakše implementacije još uvijek vrlo često koristi XMLRPC. Postoje i slični protokoli za razmjenu podataka poput JSON-RPC. SOAP, XMLRPC i slični protokoli za razmjenu poruka su naglo dobili na popularnosti sve češćim korištenjem dinamičkih jezika, web aplikacija i web tehnologija, primjerice AJAX-a (eng. *Asynchronous Javascript And XML*). Kod XMLRPC poruka podržani su razni tipovi podataka. Moguće je koristiti primitivne tipove podataka poput brojeva, znakovnih nizova ili logičkih vrijednosti, ali i složenije tipove podataka poput polja, struktura, datuma itd.

Primjer XMLRPC zahtjeva prikazan je na sljedećem ispisu:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

U ovom slučaju poziva se udaljena procedura s imenom getStateName. Proceduri se prosljeđuje vrijednost argumenta 40. Parametri se prosljeđuju u obliku XML tagova, a cijeli poziv predstavlja ispravan (eng. *valid*) i dobro oblikovan (eng. *well formed*) XML dokument. Tipični XMLRPC odgovor prikazan je na sljedećem ispisu:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>South Dakota</string></value>
    </param>
  </params>
</methodResponse>
```

U odgovoru je vraćen niz znakova "South Dakota". Kao i u slučaju XMLRPC poziva, odgovor predstavlja ispravan i dobro oblikovani XML dokument. Unutar raznih XML tagova navodi se tip povratne vrijednosti, kao i sama vrijednost. Poseban format odgovora postoji i za poruke dojave pogreške kod poziva procedure.

Za oblikovanje grafičkog korisničkog sučelja korišten je GTK 2.0 grafički programski okvir. GTK (eng. *The GIMP Toolkit*). GTK je višeplatformski sustav za izradu komponenti grafičkih sučelja (eng. *widgets*), a primarno je nastao kao grafička podrška kod izrade programa za obradu fotografija GIMP. Umjesto korištenja jednog od postojećih sustava poput *Motif* ili *Qt*, autori programa su se odlučili na izradu potpuno novog grafičkog programskog okvira. GTK je prvenstveno namijenjen radu s programskim jezikom C, ali su vrlo brzo razvijene veze (eng. *bindings*) i za druge popularne programske jezike poput Pythona, Perla, Jave i C#. Prvobitno je GTK bio dostupan samo za X Windows sustav na operacijskom sustavu Linux, ali danas ga je moguće koristiti na čitavom nizu drugih operacijskih sustava, uključujući i operacijski sustav Microsoft Windows te MacOS.

U izgledu GTK pokušava koristiti najbolje osobine drugih grafičkih korisničkih sustava poput Motifa, Windowsa, Qt i NEXTSTEP. Osnova cijelog sustava su male nezavisne grafičke komponente zvane *widgeti* (eng. *widgets*). Svaka komponenta grafičkog sučelja poput prozora, gumba, alatne trake ili polja za unos teksta je widget. GTK sustav je izrađen po objektno orijentiranom modulu, tako da svaki widget nasljeđuje svojstva od nekog drugog widgeta. Tako npr. gumb sa slikom nasljeđuje svoja svojstva od običnog gumba itd. Na vrhu hijerarhije widgeta nalazi se objekt jednostavno nazvan *Widget*. Posebna klasa widgeta su takozvani kontejneri (eng. *containers*). Iako kontejneri nisu direktno vidljivi na korisničkim formama, njihova uloga je vrlo važna. Uloga kontejnera je sadržavanja ostalih widgeta, te automatsko poravnanje i promjena veličina sadržanih widgeta. Npr. widget *Vbox* (eng. *vertical box)* podijeliti će prostor u kojem se nalazi na dva ili više dijelova koji se nalaze jedan iznad drugoga. Ukoliko u te prostore postavimo npr. widget za gumb (eng. *button*), promjenom veličine prozora automatski će se mijenjati i veličina gumba. Korisniku odnosno programeru je dostupan je veliki broj raznih kontejnera. Dodatna dobra osobina GTK sustava je mogućnost korištenja već nekoliko unaprijed definiranih standardnih dijaloga. U trenutnoj verziji GTK-a, dostupni su sljedeći dijalozi:

- dijalog s informacijama (`AboutDialog`)
- dijalog za odabir fonta (`FontDialog`)
- dijalog za odabir datoteke (`FileDialog`)

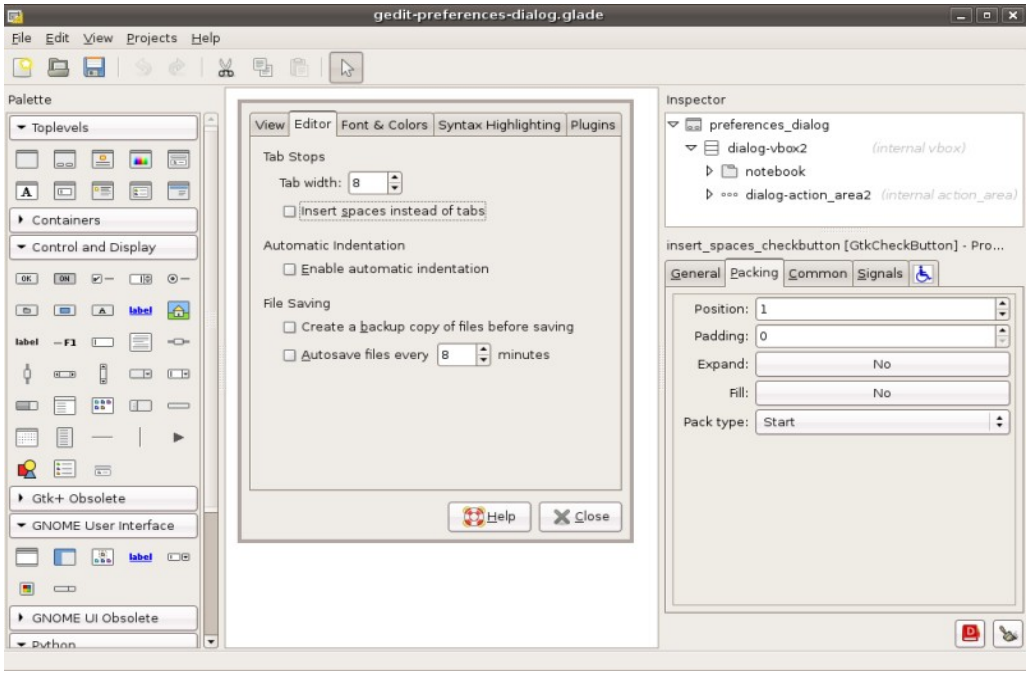
- dijalog za odabir boje (`ColorDialog`)
- dijalog za korisnički unos (`InputDialog`)
- dijalog za prikaz poruke (`MessageDialog`)
- dijalog za odabir često korištenih datotekama (`RecentDialog`)
- dijalog za pomoć korisniku (`AssistantDialog`)

Korištenjem unaprijed definiranih dijaloga izbjegava se bespotrebno programiranje komponenti koje se vrlo često koriste, a omogućuje se da razne aplikacije koriste jednake načine obavljanja raznih poslova. Programsko korištenje unaprijed definiranih dijaloga svodi se na instanciranje specifične klase, njegovo pokretanje te provjeru dobivenoga odgovora (na primjer imena datoteke kod dijaloga za odabir datoteke). Korisnik naviknut na ovakve dijaloge vrlo lako može primijeniti znanje stečeno korištenjem jedne aplikacije na svaku aplikaciju koja koristi te standardne dijaloge. Moguće je i stvaranje vlastitih dijaloga koji se kasnije mogu koristiti kao jedinstvene komponente.

Obrada događaja u GTK sustavu temelji se na sustavu obrade signala. Svaka akcija korisnika nad nekim widgetom generira signal. Postoji veliki broj raznik signala, vezanih uz akcije mišom, tipkovnicom, stvaranje vremenskih događaja itd. Za signale koje je potrebno obrađivati mora se navesti posebna callback funkcija. Callback funkcija se povezuje (eng. *connect*) na određeni signal, a kao argumente prima pokazivač na widget na kojem je generiran događaj kao i dodatni argument čija namjena ovisi o kontekstu u kojem je stvoren događaj. Na primjer, ukoliko je stvoreni događaj pute akcije korisnika korištenjem miša, dodatni argument može sadržavati informacije o tome koja je tipka miša pritisnuta, da li je još uvijek pritisnuta, koordinate pokazivača itd.

U izradi programa *Netkeeper* korištene su GTK veze za programski jezik Python, poznate pod imenom PyGTK. Iako je moguće napisati kompletnu aplikaciju korištenjem samo API poziva PyGTK, takav pristup se koristi vrlo rijetko, a češće se koriste razni alati za brzi razvoj grafičkih aplikacija (eng. *Rapid Application Development*, RAD). Puno jednostavniji i elegantniji pristup je korištenje programa i sustava Glade. Glade je program za vizualno stvaranje grafičkih korisničkih sučelja temeljenih na GTK sustavu, a posjeduje i dodatke za okruženje GNOME. Glade je neovisan o programskom jeziku i ne proizvodi nikakav automatski stvoreni programi kod. Rezultat stvorenog sučelja je XML datoteka koja se može dinamički učitati u bilo kojem programskom jeziku koji podržava GTK i Glade, primjerice u programskom jeziku Python. Ovakva XML datoteka definira samo izgled sučelja, te imena callback funkcija koje će se pozivati za događaje stvorene od strane korisnika. XML datoteka sa definicijom sučelja mora se distribuirati zajedno sa ostatkom programa, a ovakav način definiranje sučelja ima niz prednosti. Primjerice, moguće je napraviti više datoteka sa definicijom određenih widgeta, ali raspoređenih na potpuno različiti način. Dodatno, moguće je napraviti više verzija sučelja na različitim jezicima. Nakon inicijalne izrade sučelja, sve promjene se mogu obavljati i ručno uređivanjem same XML datoteke. Izgled Glade dizajnera prikazuje slika 4.3.

Kod izrade programa *Netkeeper* korišteni su widgeti sustava GTK. Dodatno je korištena biblioteka za stvaranje iskakajućih prozora (eng. *popups*) u dijelu korisničkog sučelja



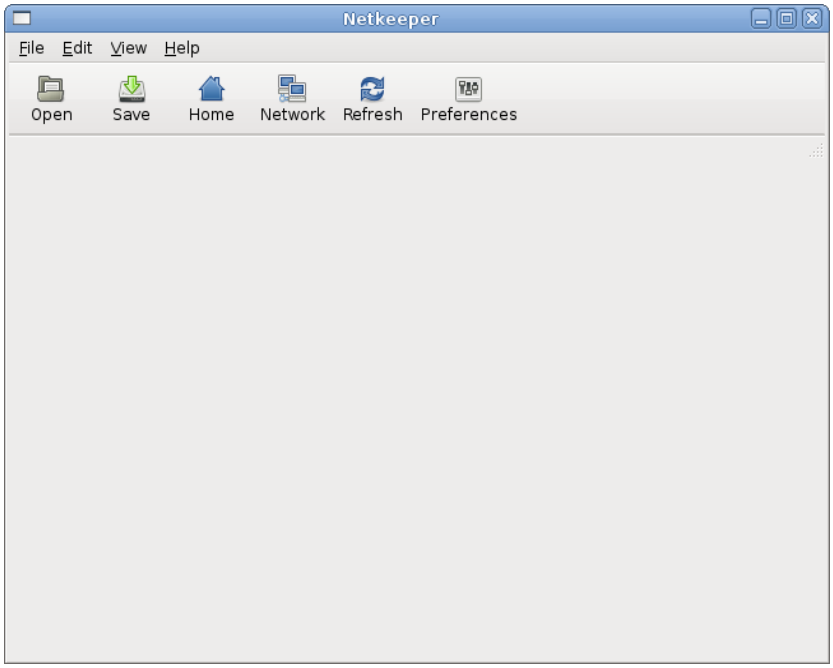
Slika 4.3: Glade dizajner

rezerviranom za razne statusne ikonice (eng. *status area*). Standardna biblioteka koja se koristi za ovu namjenu je *libnotify*. Iskakajući prozori služe za prikaz napada koji su u tijeku i sadrže gumbe putem kojih korisnik može ignorirati napad ili napraviti dodati odgovarajuće pravilo u sigurnosnu politiku.

Glavni prozor programa *Netkeeper* podijeljen je korištenjem odgovarajućeg kontejnera na četiri dijela u vertikalnog rasporedu (eng. *vertical box*). Izgled glavnog prozora prikazuje slika 4.4. Gledajući od vrha prozora prema njegovom dnu, mogu se razlikovati četiri komponente:

- izbornik (eng. *menu*)
- alatna traka (eng. *toolbar*)
- glavni dio prozora
- statusni redak (eng. *status bar*)

Izbornik služi za definiranje dodatnih izbornika (eng. *submenus*) sa često korištenim akcijama. U ovom konkretnom primjeru dostupne su akcije poput spremanja postavki klijenta, učitavanja postavki klijenta, izlaska iz programa itd. Izbornik *Edit* uobičajeno sadrži akcije vezane uz trenutno odabranu komponentu glavnog prozora. U programu *Nekeeper* dostupne su akcije vezane uz odabrani tekst, poput izrezivanja (eng. *cut*), kopiranja (eng. *copy*) i lijepljenja (eng. *paste*). Izbornik *View* koristi se za definiranje raznih postavki prikaza prozora i ostalih dijelova sučelja. Primjerice može se odabrati koja od navedene četiri komponente mora biti prikazana. Dodatno, može se odabrati prikaz ili sakrivanje statusne ikonice.

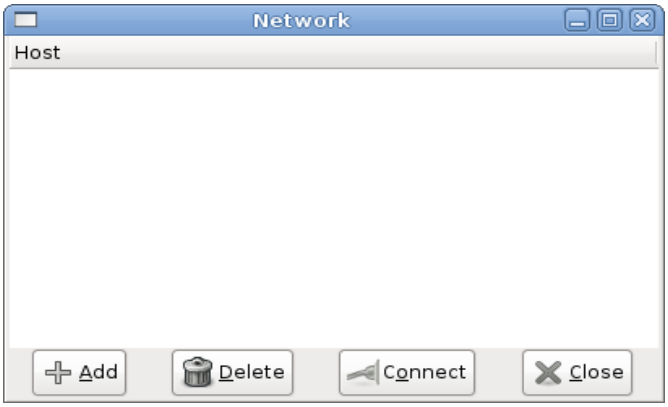


Slika 4.4: Izgled glavnog prozora nakon pokretanja

Izbornik pomoći (eng. *help*) sadrži samo jednu akciju koja prikazuje prozor s informacijama o programu (eng. *about window*). U ovom programu dostupna je informacija o autoru, adresama za kontakt kao i licenci pod kojom je program objavljen. Izgled prozora s informacijama prikazuje slika 4.5.

Slika 4.5: Prozor s informacijama

Sljedeće komponenta glavnog prozora je alatna traka. Alatna traka sadrži neke funkcije koje su dostupne i preko prethodno opisanih izbornika, ali ponovno su navedene i u alatnoj traci iz ergonomskih razlog. Većina programa najčešće korištene funkcije stavlja i u izbornike i u alatnu traku. U ovom slučaju, to su funkcije spremanja odnosno učitavanja postavki klijenta. Dodatni gumbi u alatnom okviru kontrola služe za brzo odlazak na stranicu s sumarnim informacijama o pokrenutim agentima (*Home*), osvježavanje podataka s agenata te definiranje postavki programa. Posebnu ulogu ima gumb *Network* koji služi za definiranje adresa agenata koji će se pratiti pute klijentskog programa. Izgled prozora za dodavanje novih agenata pokazuje slika 4.6.

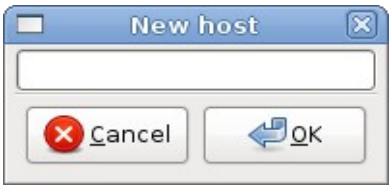


Slika 4.6: Prozor za dodavanje agenata

U prozoru za dodavanje agenata dostupne su sljedeće funkcije:

- dodavanje novog agenta za praćenje
- uklanjanje agenta iz liste za praćenje
- prebacivanje prikaza glavnog prozora na odabrani agent
- zatvaranja prozora

Odabirom gumba za dodavanje agenta pojavljuje se dijalog za upis adrese agenta prikazan na slici 4.7. Moguće je koristiti IP adresu ili DNS labelu računala na kojemu je pokrenut *Netkeeper* agent. U slučaju da agent na navedenom računalu nije pokrenut, prijavit će se poruka s pogreškom, u protivnome će se agent dodati u listu prethodno prikazanog prozora i početi će se pratiti događaji generirani na računalu na kojem je agent pokrenut.



Slika 4.7: Dodavanje novog agenta

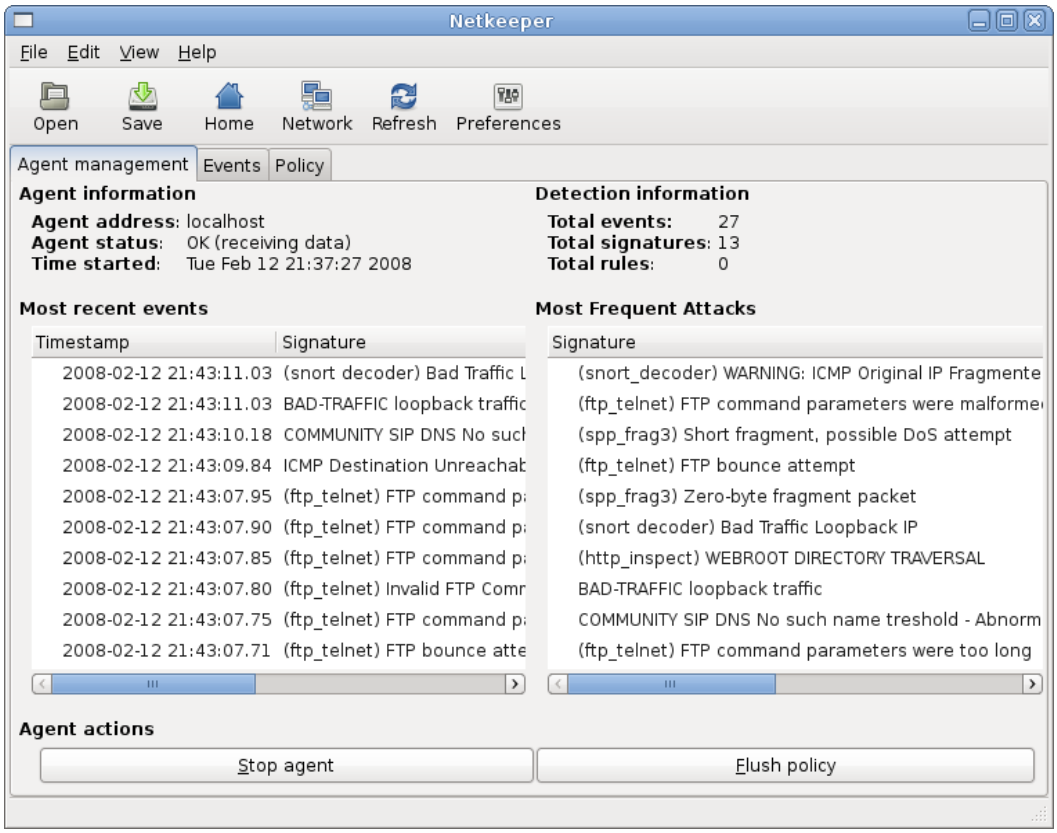
Potrebno je i napomenuti da svi gumbi u sustavu GTK imaju mogućnost korištenja velikog broja standardnih ikonica (eng. *stock icons*), Primjer takvih ikonica vidi se na glavnom prozoru u području alatne trake ili npr. na gumbima prozora za dodavanje agenta.

Uklanjanjem agenta iz liste agenata u prozoru koji se dobije odabirom ikonice *Network* sa alatne trake klijentski program prestaje pratiti rad tog agenta, ali agent nastavlja sa radom i koristi definiranu sigurnosnu politiku. Gumb *Connect* prikazuje prikaz glavnog dijela prozora na detaljnije informacije o odabranom agentu. Pritiskom na posljednju tipku zatvara se prozor za definiranje liste agenata.

Na dnu glavnog prozora nalazi se statusni redak. Statusni redak obično ima namjenu prikaza trenutno izvođene akcije i općenitih informacija o radu programa. Često se u statusnom retku ispisuje i kontekstno ovisni tekst pomoći. Prelaskom pokazivača preko

određene tipke sučelja u statusnom retku može se ispisati kratak tekst s opisom funkcije te tipke. U programu *Netkeeper* statusni redak nema nikakve specijalne funkcije.

Većina prikaza glavnog prozora je nakon pokretanja prazna. Razlog tome je što program nakon pokretanja nije povezan niti na jedan agent. Da bi se dobio prikaz i u ovom dijelu prozora, potrebno je ili učitati prethodno definiranu listu agenata odabirom opcije *Open* u izborniku *File,* ili definiranjem novog agenta u prozoru s listom agenata. Nakon što je klijentski program uspješno uspostavio vezu sa *Netkeeper* agentom, pojavljuje se prikaz u glavnom dijelu prozora, kao što pokazuje slika 4.8.

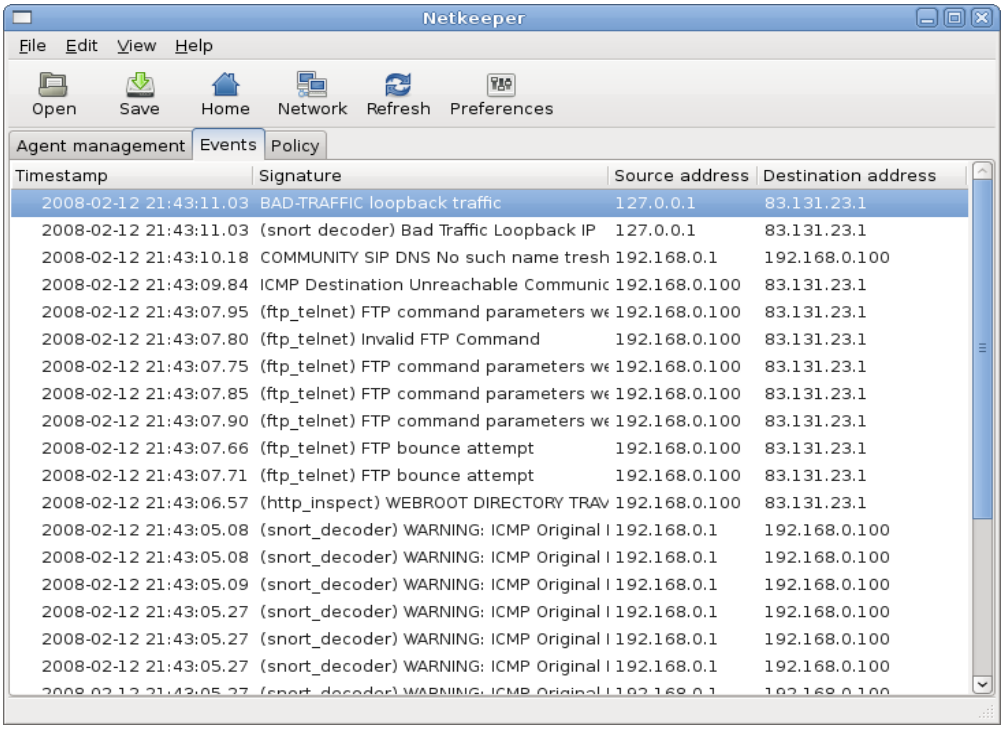


Slika 4.8: Glavni prozor za vrijeme rada programa

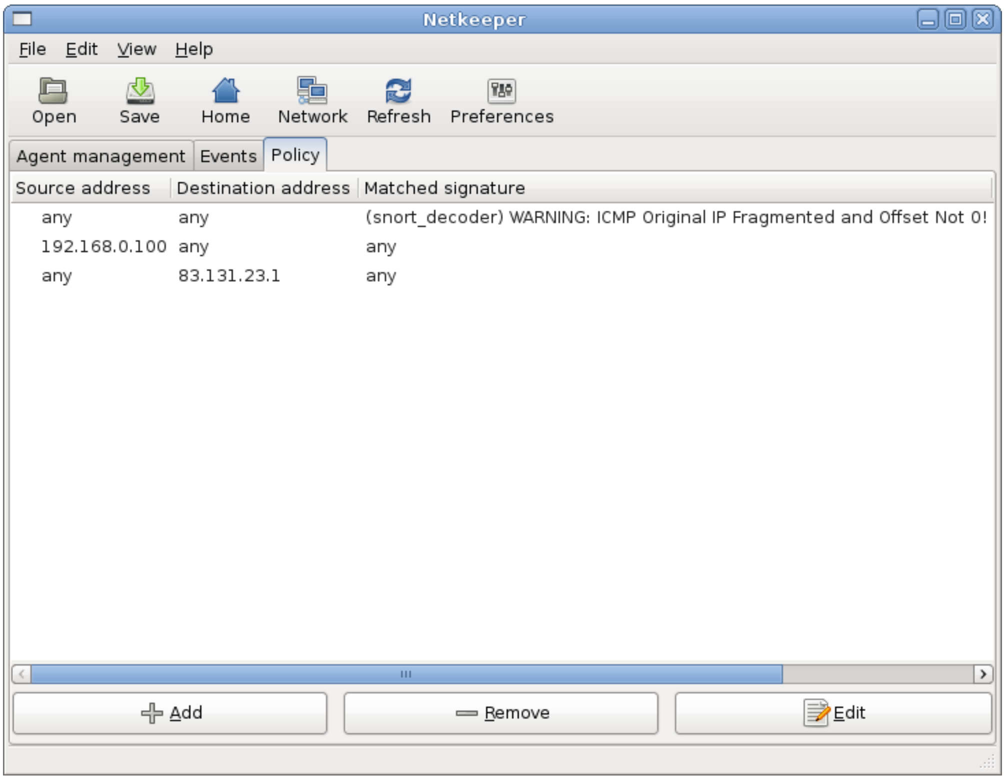
U gornjem dijelu prikaza nalaze se informacije o agentu te otkrivenim napadima i broju pravila u sigurnosnoj politici programa. Za agent se ispisuju informacije o adresi te vremenu kada je pokrenut. Glavnina prikaza otpada na dvije liste s popisom napada. Lijevi dio prikaza pokazuje popis zadnjih deset otkrivenih napada, zajedno s vremenom kada su otkriveni te opisom napada. Desni dio daje informacije o najčešćih deset napada. Prikazane su informacije o broju pojavljivanja određenog napada, kao i opis tog napada. U dnu ekrana nalazi se gumb pomoću kojeg je moguće zaustaviti rad agenta na koji smo trenutno povezani, kao i gumb za pražnjenje sigurnosne politike tog agenta.

Za vrijeme rada, glavni prozor sadrži standardni GTK widget nazvan *notebook*. Notebook se sastoji od nekoliko odvojenih okvira s prikazom, a prikaz pojedini okviri se može mijenjati odabirom kartica (eng. *tabs*) na vrhu notebooka. Nakon povezivanja na jedan od

agenta, po definiciji se pokazuje prvi okvir koji je prethodno opisan. Dodatna dva okvira prikazuju detaljne informacije o otkrivenim napadima (*Events* kartica), kao i informacije o sigurnosnoj politici (*Policy* kartica). Izgled prethodno dva opisana okvira opisuju slike 4.8 i 4.9.



Slika 4.9: Okvir za pregled otkrivenih napada



Slika 4.10: Okvir za definiranje sigurnosne politike

Pritiskom desne tipke miša na određeni redak u listama okvira, pojavljuje se izbornik sa dostupnim akcijama. Akcije za definiranje sigurnosne politike opisane su u poglavlju o arhitekturi programa. U okviru za prikaz sigurnosne politike moguće je odabrati jedan od gumba za dodavanje, uređivanje ili brisanje postojećeg pravila.

4.3. Eksperimentalno korištenje programa

Funkcionalnost programa isprobana je na kućnom računalu s instaliranim Snortom verzije 2.7.0, bazom podataka PostgreSQL 8.3, te programom iptables verzije 1.3.8. za generiranje sigurnosnih upozorenja koristio se program *idswakeup*. Idswakeup je skup alata koji se uobičajeno koriste za ispitivanje mogućnosti otkrivanja raznih sustava za otkrivanje mrežnih napada. Za ispravan rad programa potrebno je dodatno imati instaliran program *hping2*. Sintaksa korištenja je sljedeća:

```
# idswakeup <source address> <destination address>
```

Primjerice, ukoliko želimo simulirati niz napada sa računala `graph.zemris.fer.hr` (`161.53.65.248`) na računalo `faramir.zemris.fer.hr` (`161.53.65.246`), možemo koristiti sljedeću sintaksu:

```
# idswakeup 161.53.65.248 161.53.65.246

-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
-   IDSwakeup : false positive generator                   -
```

```
-  Stephane Aubert                                          -
-  Hervé Schauer Consultants (c) 2000                       -
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

  src_addr:161.53.65.248  dst_addr:161.53.65.246  nb:1   ttl:1
  sending : teardrop ...
  sending : land ...
  sending : get_phf ...
  sending : bind_version ...
  sending : get_phf_syn_ack_get ...
  sending : ping_of_death ...
  sending : syndrop ...
  sending : newtear ...
  sending : X11 ...
  sending : SMBnegprot ...
  sending : smtp_expn_root ...
  sending : finger_redirect ...
  sending : ftp_cwd_root ...
  sending : ftp_port ...
  sending : trin00_pong ...
  sending : back_orifice ...
  sending : msadcs ...
```

Razvijeni sustav je uspješno uspio prepoznati napade generirane na ovaj način, kao i niz stvarnih vrsta napada poput skeniranja pristupnih vrata, djelovanja crva, automatiziranih pokušaja provale (npr. *brute force* napadom na SSH protokol) itd. Jedini nedostatak uočen pri radu programa je korištenje automatiziranog blokiranje adresa korištenjem dinamičkih pravila. Ukoliko je neko korištenje krivo okarakterizirano kao napad, moguće je da se bez razloga zabrani promet s određenim adresama.

5. Zaključak

Sustavi za otkrivanje napada sa mreže su tehnologija koja se još uvijek brzo razvija i poboljšava. S vremenom su postali nezamjenjivi dio cjelokupnog sigurnosnog sustava i toliko ih je uobičajeno naći koliko je uobičajeno da se koristi i sigurnosna stijena. Međutim, brzim razvojem tehnologije NIDS-ovi se također moraju prilagođavati sve većim zahtjevima tržišta i moraju odolijevati sve lukavijim tehnikama napada. U samo dvadesetak godina koliko postoje evoluirali su od najprimitivnijih metoda nadziranja do današnjih nevidljivih distribuiranih sustava za rad u stvarnom vremenu koji mogu nadzirati i gigabitne mreže s izrazito velikom količinom prometa.

Danas je gotovo nemoguće govoriti isključivo o NIDS-u, a da se ne spomene u kontekstu složenijih sustava poput sustava za prevenciju napada (IPS). Otkrivanje napada samo po sebi nije previše korisno, pa je za očekivati da će se industrija u budućnosti u potpunosti orijentirati razvoju sustava za prevenciju napada i razvoju objedinjenih sustava koji će objedinjavati više sigurnosnih komponenti unutar jednog programskog paketa (poput već spomenutog Honeynet projekta). Također, danas je sve manja razlika između računalno i mrežno zasnovanih sustava za otkrivanje napada, tako da su sve češći proizvodi koji objedinjuju obje tehnike (primjerice sustav ISS). Postoje čak i potpuno nova rješenja kod kojih se neka načela sustava za otkrivanje napada koriste i u ojačavanju jezgre operacijskog sustava. Primjer takvoga rješenja je projekt *LIDS*.

Pred sustavima za otkrivanje mrežnih napada je nekoliko velikih izazova. Jedan od najvećih će zasigurno biti rješenje problema nadziranja kriptiranih kanala koji se sve češće koriste, a problem će doživjeti vrhunac potpunim prelaskom na IPv6. Jedan od čestih prigovora na implementaciju NIDS rješenja unutar ide račun na ukupne cijene postavljanja i održavanja (eng. *total cost of ownership, TCO*). Čak i u slučajevima kada se koriste rješenja otvorenog koda, a ne komercijalne inačice IDS-a, postoje relativno veliki dodatni troškovi koji posebice rastu ukoliko je potrebno napraviti robusno rješenje u danas već standardnim mrežama koje se zasnivaju na radu s preklopnicima. U takvim slučajevima je potrebno intenzivno koristiti mrežne priključnice koje su još uvijek vrlo skupi uređaji. Nadalje, problem predstavlja i sve veći rast propusnosti mreža.

Bez sumnje, sadašnja NIDS rješenja će još dugo vremena biti nezamjenjivi dio mreže na sigurnosnom planu, a razvijaju se i sve bolje verzije programa. Rastući broj sigurnosnih problema ide samo u korist ovoj tvrdnji. Posljednji trendovi u razvoju NIDS-a su sve veća upotreba matematičkih koncepata te uvođenje umjetne inteligencije (eng. *artificial intelligence, AI*) u proces otkrivanja. Neki proizvođači su u svoje produkte (npr. ISS) počeli ugrađivati tehnologije kojima će sustav moći modificirati sam sebe na osnovu prethodno dobivenih informacija o napadima, a sustav neće više biti pasivan tj. neće više samo slušati na nekom mrežnom sučelju nego će provoditi aktivnu analizu mreže poput traženja propusta i skeniranja pojedinih segmenata mreže. Primjer ovakvih modernih rješenja su i Cisco *self – aware* ili *self – defending* mreže koje u sebi imaju objedinjene i IDS i IPS sustave. Budućnost svakako donosi još puno prostora za poboljšavanje NIDS-ova koji su nedvojbeno već i sada tehnologija koju na svojoj mreži mora imati svatko tko drži do sigurnosti vlastitih podataka i računalnih sustava.

6. Literatura

1. R. Rehman, *Intrusion Detection Systems with Snort*, Prentice Hall PTR, Upper Saddle River, New Jersey, 2003.
2. J. McGibney, *Intrusion Detection Systems & Honeypots*, Waterford Institute of Technology, Ireland, Prezentacija sa konferencije INET/IGC, Barcelona, 2004
3. B. Laing, *Implementing a Network Based Intrusion Detection System*, Sovereign House, Reading, 2000.
4. CARNet CERT i LS&S, *Sustavi za sprečavanje neovlaštenih aktivnosti*, CCERT-PUBDOC-2006-01-145, Zagreb, 2006.
5. Intrusion Detection System, Wikipedia, http://en.wikipedia.org/wiki/Intrusion_detection_system (6/1/2008)
6. Sans Institute, Intrusion Detection FAQ, URL: http://www.sans.org/resources/idfaq/ (30/01/2008)
7. Carnegie Melon University, CERT/CC Statistics 1998 – 2006, URL: http://www.cert.org/stats (26/06/2006)
8. Snort, Wikipedia, http://en.wikipedia.org/wiki/Snort_(software) (1/2/2008)
9. IETF, RFC 768, User Datagram Protocol, URL: http://tools.ietf.org/html/rfc768
10. IETF, RFC 791, Internet Protocol, URL: http://tools.ietf.org/html/rfc791
11. IETF, RFC 793, Transmission Control Protocol, URL: http://tools.ietf.org/html/rfc793
12. Berkley Packet Filter (BPF), http://www.gsp.com/cgi-bin/man.cgi?section=4&topic=bpf (1/2/2008)
13. Net Optics, Network Taps and Aggregation Solutions for Passive Network Access, URL: http://www.netoptics.com (19/2/2008)
14. Finisar, High Speed Data Communication for Networking and Storage, http://www.finisar.com (20/2/2008)
15. Common Intrusion Detection Format, URL: http://gost.isi.edu/cidf (18/2/2008)
16. Honeynet Project, URL: http://www.honeynet.org/papers/honeynet (20/2/2008)
17. Barnyard, URL: http://sourceforge.net/projects/barnyard (17/2/2008)
18. Prelude IDS, URL: http://www.prelude-ids.org (20/2/2008)
19. Nessus, URL: http://www.nessus.org/nessus (20/2/2008)
20. Samhain, URL: http://www.la-samhna.de/samhain (20/2/2008)
21. Sniffing cable, URL: http://www.snort.org/docs/faq/1Q05/node31.htm (20/2/2008)
22. Network Tap, http://en.wikipedia.org/wiki/Network_tap (20/2/2008)