SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Implementacija agenta za IDS

Projekt

Tomislav Dukarić
br. indeksa: 40503/11-R

Mentor:
Doc. dr. sc. Markus Schatten

Varaždin, 5. veljače 2014.

Sadržaj

Poglavlje 1

Uvod

U današnje vrijeme sve više i više osjetljivih informacija se prenose putem interneta. Time su ujedno i te informacije dostupne osobama koje žele neovlašteno doći do njih. Iz tog razloga danas se dosta ulaže u samu sigurnost prenošenja informacija te zaštiti mreža po kojima ono prolazi. Prije je bilo dovoljno postaviti vatrozid koji bi štitio mrežu do vanjskih napada no ti napadi su postajali sve napredniji i složeniji te klasični oblik vatrozida ne predstavlja dovoljno snažan mehanizam obrane. Iz tog razloga danas se sve više i više ulaže u kreiranje i korištenje naprednijih oblika vatrozida koji su u stanju osjetiti napad pomoću svojih senzora te reagirati na njega.

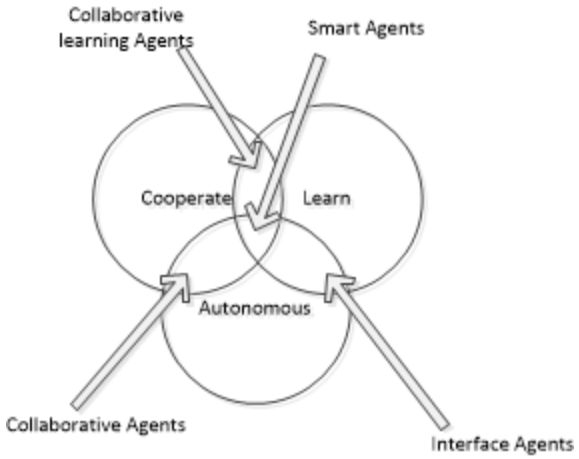
Jedan od načina kako se aktivno štiti od mogućih napada s drugih mreža ili interneta jest korištenje IDS-a. IDS (Intrusion Detection System) se koristi kako bi se pratile aktivnosti na mreže. Ukoliko dođe do anomalija tokom rada mreža te anomalije se prijavljuju administratoru mreže ili osobi koja je zadužena za sigurnost. Zatim su se razvili IPS (Intrusion Prevention System) sustavi koji su omogućili automatsku zaštitu subjekata na mreži. Ti sustavi kad su upareni s IDS-om su u stanju odmah reagirati na moguće anomalije te zabraniti pristup sumnjivim subjektima na mrežu ili ograditi subjekt od mrežnog pristupa ukoliko se uvidi da je on meta napada ili izvor napada te se o tome obavijesti administrator.

U ovome radu jest implementiran višeagentni sustav koji se koristi kao napredna verzija IDS-a. Ona skuplja informacije o mogućim prijetnjama te ih prosljeđuje drugim agentima na mreži koji se mogu doraditi na način da javljaju vatrozidu moguće prijetnje te ovisno o tim prijetnjama izvršiti potrebne radnje.

Poglavlje 2

Teorijska osnova

Danas još ne postoji jedinstvena definicija što je to agent. Kod računalnih agenata navode se različite definicije, dok neka osnovna definicija agenta bi se mogla dati da je agent program ili sklopovlje koje se ponaša prema točno definiranim pravilima kako bi izvršili zadatke umjesto korisnika (Nwana, 1996). Odnosno, može se reći da su agenti programi koji obavljaju posao umjesto korisnika te su time njegova zamjena u određenim situacijama. Kako je sam pojam agenta dosta jednostavan za široko područje koje ono opisuje tako se radi podjela agenata. Agenti se dijele na svoje podvrste pomoću svojih svojstava. Postoje tri svojstva kojima se agenti mogu opisati, a to su svojstva da agenti mogu učiti iz okoline, da su autonomni i/ili da su kooperativni agenti.
Prema slici mogu se vidjeti podvrste agenata:



Slika 2.1: Podjela agenata (Nwana, 1996)

Prema navedenom dijagramu mogu se vidjeti kako su agenti podijeljeni. Oni su podijeljeni na kolaborativne agente, kolaborativne agente koji imaju sposobnost učenja, agente sušelja te pametne agente. Pametni agenti su agenti koji sadrže sva tri svojstva - kooperativni su, imaju sposobnost učenja te su autonomni. No postoje još podjela. Autor u svome članku (Nwana, 1996) također navode druge podjele. No sve podjele se svode na prvotnu podjelu, samo što se detaljnije definiraju.

Primjerice, navodi se sljedeća podjela:

- kolaborativni agenti

- agenti za sučelja
- mobilni agenti
- informacijski agenti
- reaktivni agenti
- proaktivni agenti
- hibridni agenti
- pametni agenti

No i ta podjela jest dosta štura te nije u mogućnosti predočiti pravu podjelu svih vrsta agenata koje postoje. No, i ova podjela je detaljnija od prethodne te je moguće detaljnije definirati koje vrste je agent ili skupina agenata. Prvi tip agenata, kolaborativni agenti su oni agenti koji međusobno surađuju s ostalim agentima te su sami po sebi autonomni. Drugi tip agenata, agenti za sučelja su agenti koji obavljaju zadaće svoga vlasnika, odnosno, korisnika. Pomoću tih agenata korisnik može zadati zadatke agentima koji ih potom izvode. Mobilni agenti si agenti koji nisu vezani uz jedno mjesto nego se sele s jednog mjesta na drugi. Ti agenti uglavnom prikupljaju razne informacije s interneta. Jedan tip tih agenata su *crawleri* koji prate što se događa na internetskim stranicama te dohvaćaju sve informacije koje mogu. Informacijski agenti se koriste za obradu podataka. Oni se često vežu uz mobilne agente. Mobilni agenti prikupljaju informacije koje informacijski agenti pročišćuju. Zatim slijede reaktivni agenti. Reaktivni agenti reagiraju na promjenu u svojoj okolini te reagiraju nakon nastale promjene. Za razliku od njih, proaktivni agenti reagiraju prije nastajanja same promjene, odnosno oni djeluju samostalno kako bi postigli svoj cilj. Hibridni agenti su agenti čija svojstva odgovaraju više različitim svojstvima podjele agenata. Primjerice, ukliko je agent i mobilni i informacijski agent taj agent je zapravo hibridni agent. Pametni agenti su oni agenti koji upotpunjuju skoro sva svojstva agenata. To se može vidjeti prema prethodnom dijagramu gdje se vidi da se on nalazi u sjecištu svih osobina agenata.

2.1 Reaktivni agenti

Reaktivni agenti su jedni od najjednostavnijih agenata koji postoje. Oni prate vanjski svijet te ovisno o promjeni u vanjskom svijetu reagiraju na točno definiran način. Jedan od primjera takvih agenata su senzori u robotima. Također, jedna od njihovih osobina jest jednostavnost interakcije s ostalim agentima. Kod izrade reaktivnih agenata sama komunikacija se pokušava svesti na sam minimum te se pojednostavljuje što je više moguće. No, ta jednostavnost zavarava. Zapravo, reaktivni agenti koji međusobno surađuju mogu stvarati dosta komplicirane sustave agenata koji mogu obavljati dosta komplicirane radnje. Primjerice, sve radnje robota mogu se kontrolirati reaktivnim agentima.

Također, Ferber u svome članku objašnjava da su reaktivni agenti jednostavni i jednostavno razumljivi te da im je kognitivna ekonomija niska jer moraju pamtiti mali broj stvari (Ferber, 1994). Oni ne planiraju unaprijed nego njihovo ponašanje ovisi o trenutnoj situaciji. Velika prednost reaktivnih agenata te njihovoj jednostavnosti jest to što su oni uglavnom brzi agenti, brzo reagiraju na promjene u okolini, fleksibilni su te se mogu jednostavno prilagoditi novim situacijama. Također, ukoliko se neki agent izgubi to uglavnom ne snosi katastrofalne posljedice na cijeli sustav. Primjerice kod IDS sustava, ukoliko se jedan senzor izgubi ostatak sustava nastavlja sa svojim radom te ne narušava cjelovitost

sustava.

No, jednostavnost reaktivnih agenata snosi i svoje posljedice. Kako su svi reaktivni agenti napravljeni za određene situacije dolazi do problema - kako će reagirati na drugačiju okolinu? Kako će se prilagoditi novoj okolini? Problem kod njih jest to što nema točno definirane ciljeve te pitanje do njih. Oni su jednostavni, prosti agenti koji rade točno ono za što si programirani te se ne mijenjaju. Iz tog razloga postoji relativno mali broj čisto reaktivnih agenata te se češće koriste hibridi agenti gdje se način rada reaktivnih agenata usklađuje s drugim vrstama agenata. Često se oni prilagođavaju mrežnim agentima kako bi se mogli "snaći" u novim situacijama.

Uz čiste reaktivne agente postoje još jedna vrsta agenata koji su po svojem opisu slični reaktivnim agentima. To su agenti koji se dogovaraju, dogovorljivi agenti. Oni su po svojoj strukturi skoro isti reaktivnim agentima te ih stoga mnogi znaju miješati. Postoji jedna velika razlika između reaktivnih agenata i dogovorljivih agenata. Dok reaktivni agenti izvršavaju svoju dužnost bez obzira na ostale agente, dogovorljivi agenti obavljaju svoju dužnost s obzirom na druge agente. Odnosno, dogovorljivom agentu drugi agent prenese informacije o tome što treba napraviti. Za razliku od njih, reaktivni agenti sami dolaze do informacija, odnosno, reagiraju s obzirom na okolinu a ne s obzirom na drugog agenta. U ovome radu dio agenata su reaktivni agenti dok je dio agenata zapravo dogovorljivi agenti, iako bi se mogli pretpostaviti na prvu ruku da su svi agenti reaktivni agenti.

Poglavlje 3

Obrada znanstvenih članaka

U ovome dijelu biti će ukratno opisani članci koji su bili preuzeti s interneta koji opisuju navedeni model višeagentnog sustava koji se koristio za potrebe izrade projekta.

3.1 I. članak

Naslov članka:	An Architecture for Intrusion Detection using Autonomous Agent (Jai Sundar Balasubramaniyan, 1998)
Autori:	Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, Diego Zamboni
Datum objave:	7-11.12.1998
Mjesto objave:	Computer Security Applications Conference, 1998. Proceedings. 14th Annual

Svha članka jest dati uvid u trenutno stanje IDS sustava te na čemu se oni temelje, dali su njihove prednosti i nedostatke te su dali primjer kako bi se IDS mogao poboljšati korištenjem autonomnih agenata. ID (eng. Intrusion detection) se može definirati kao problem identificiranja individua koji koriste računalni sustav bez autorizacije i onih koji imaju pristup sustavu no iskorištavanju njegove privilegije (B. Mukherjee and Levitt, 1994). Ono što nedostaje u navedenoj definiciji jest pokušaj. ID bi trebao otkriti i pokušaje upada u računalni sustav, a ne samo uspjele pokušaje u upad. Također su predstavili da se model ID-a kategorizica na dva modela. Prvi model se odnosi na traženje sigurnosnih rupa u sustavu koji se iskorištavaju dok drugi model se odnosi na detekciju anomalija kod korištenja sustava. Također, IDS bi se trebao izvršavati u realnom vremenu te bi trebao brzo reagirati na bilo kakav pokušaj iskorištenja sustava koje nisu u skladu s dopuštenjima. Što znači da bi se morao izvršavati konstantno s minimalnom ljudskom interakcijom, trebao bi biti otporan na padove sustava, trebao bi pratiti samog sebe u slučaju da dolazi do internih promjena u sustavu, trebao bi minimalno opterećivati sustav, trebao bi biti konfigurabilan te bi se trebao prilagođavati promjenama sustava i korisnika. Također bi se trebao moći skalirati s obzirom na opterećenje na mreži. Također su predstavili nedostatke, tj. limitacije trenutnih ID sustava. Jedan od tih nedostataka jest to što ID uglavnom ima jednu centralnu točku o kojoj ovisi cijeli sustav. Ukoliko ta točka ispadne iz sustava, pada cijeli sustav. Pomoću autonomnih agenata sami agenti bi se prilagodili novoj situaciji te bi nastavili s radom zbog njihove hijerarhijske strukture. Jedostavno bi se priklonili drugom stablu agenata te nastavili sa svojim radom. Također, problem je kod skalabilnosti. Sami agenti mogu preuzeti na sebe dio odgovornosti te obrade informacija te time

rasteretiti glavno računalo koje prati agente. Kod rekonfiguriranja također postoje prednosti jer svaki agent stoji za sebe te je time jednostavni prolagoditi određene agente svojim potrebama te se time ne narušavaju ostali agenti, odnosno, promjenom jednog agenta ne utječe na rad drugog agenta te drugi agenti nastavljaju sa svojim radom. Također, kod multiagentskog sustava administratori sustava mogu pokretati i zaustavljati razne agente koji su na samom rubu hijerarhije a da ne utječu na rad cijelog sustava.

Autori su također dali primjer radova na istu temu. Primjerice, postoji GrIDS projekt koji koristi različite module koji se svaki izvršava na svojem računalu te obavještava centralni poslužitelj o svemu. NADIR sustav radi na sličan način, no sve informacije obrađuje ekspertni sustav. Kod CSM sustava svaki IDS radi za sebe, no međusobno se mogu povezati te razmjenjivati informacije. Također postoji EMERALD projekt koji se svodi na distribuiranu arhitekturu za ID koja se sastoji od više servisa koji se izvršavaju na računalima te prate rad. S obzirom na trenutne projekte autori su dali prijedlog AAFID (Autonomous Agents For Intrusion Detection) arhitekture za kreiranje IDS-a koji koristi autonomne agente kao najniže elemente u hijerarhiji za prikupljanje podataka i njihovu analizu. Agenti su ti koji drže cijeli sustav stabilnim te ga opskrbljuje novim informacijama o okolini. Oni bi trebali moći učiti o svojoj okolini pomoću genetičnog programiranja, trebali bi pratiti sesije kod konekcija te bi se trebali moći migrirati između različitih računala kao mobilni agenti. Osim samih agenata postoje još dva elementa sustava, a to su transiveri i monitori. Transiveri kontroliraju agente te obrađuju njihove podatke. Monitori su na najvišoj razini te oni upravljaju cijelom strukturom te nude pristup cijelom sustavu. Ukoliko u sustavu postoji samo jedan monitor tada on predstavlja najslabiju točku jer ukoliko napadač uspije ugasiti monitor cijeli IDS pada. Kao prijedlog komunikacije između dijelova sustava te sustava s okolinom je predloženo korištenje SNMP (Simple Network Management Protocol) za konfiguraciju sustava te za njegovo održavanje.

U svrhu testiranja rada njihovog sustava kreirana su dva prototipa. Drugi prototip je napisan u Perl-u te se izvršavao na UNIX platformi. Tu su ujedno uvijdjeli nedostatke sustava. Jedan od najvećih problema jest korištenje programskog jezika Perl. Kako se većina informacija izvršava na razini Kernela zbog programskog jezika Perl velik dio podataka mora prelaziti iz Kernel moda u korisnički mod što dodatno opterećuje sustav. Zbog toga su predložili da se cijeli sustav napiše u programskom jeziku C kako bi se smanjilo opterećenje sustava u vidu memorije i procesora. Korištenjem C-a kao programskog jezika mogli bi omogućiti višedretveno korištenje sustava te da se sve izvršava na razini kernela. Također, trebalo bi balansirati opterećenje samih računala na kojima se izvršava agent te dio procesiranja informacija prebaciti na njega kako bi se smanjilo opterećenje mreže i centralnih poslužitelja. Tu bi trebalo pronaći zlatnu točku balansa između dva ekstrema - obrađivati sve informacije kod klijenata ili kod poslužitelja. Također predlažu daljnji razvoj korisničkog sučelja kako bi se olakšalo praćenje sustava te napada. Također su predložili modifikaciju sustava da podržava CIDF format koji se koristi kod ostalih ID sustava kako bi mogao komunicirati s ostalim sustavima. Iz navedenog rada može se vidjeti kako ima još dosta posla u vezi uporabe agenata, odnosno višeagentnog sustava u izradi ID sustava koji sam sebe održava te koji se može koristiti u realnim sustavima.

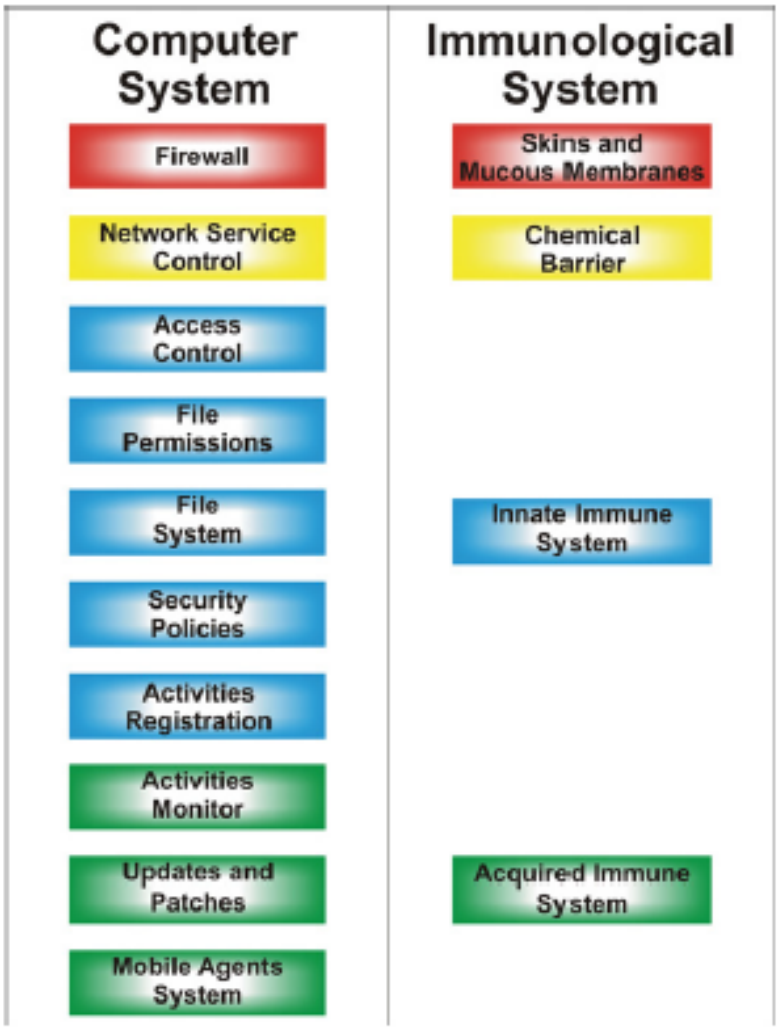
3.2 II. članak

Naslov članka:	An agent based and biological inspired real-time intrusion detection and security model for computer network operations (Azzedine Boukerche, 2007)
Autori:	Azzedine Boukerche, Renato B. Machado, Kathia R.L. Jucá, João Bosco M. Sobral, Mirela S.M.A. Notare
Datum objave:	26.9.2007
Mjesto objave:	Computer Communications 30 (2007)

Drugi članak govori o internoj strukturi ID sustava, odnosno i tome kako bi IDS trebao uočavati promjene nad sustavima te kako bi trebao reagirati. Kao prijedlog osnovnog dizajna autori su dali izgled, tj. ponašanje obrambenog sustava u prirodi. Autori tvrde da postoji snažna korelacija između ljudskog imunološkog sustava i računalnog mrežnog sigurnosnog sustava. Ono što su oni htjeli dati na uvid jest korištenje računalnih agenata kao zamjena za receptore opasnosti za ljudsko tijelo, odnosno, uporaba agenata u računalnim sustavima u istu stvrhu kao što receptori imaju u ljudskom tijelu. Prema njihovom dizajnu, IDS sustav generira tri grupe podataka ovisno o tipu detekcije. To su napadi, sigurnosni događaji te kršenje sigurnosti. Ti događaji su analizirani, distribuirani i pohranjeni korištenjem agenata. Njihov sustav je baziran na detekciji anomalija dok konstantno prati aktivnosti korisnika. Za agente su preldožili korištenje mobilnih agenata koji se mogu seliti s jednog sustava na drugi kako bi kombinirali lokalnu interakciju s mobilnost koda u mobilnih agentima te kako bi izbjegli tipičnu server-client arhitekturu.

Autori su također zamislili da agenti vrše provjeru rada sustava pomoću logova koji su na poslužitelju, točnije, pomoću alata *Logcheck*. S tim alatom bi automatski pratili zapise sustava te time pratili postoje li anomalije u radu sustava. Osim tog alata mislili su koristiti i *syslog-ng*. Za dizajn sustava predlažu korištenje distribuiranog agentnog sustava i arhitekture temeljenem na domaćinima. Kao odgovor na mogući napad predlažu korištenjem dva odgovora. Prvi odgovor se bazira na slanju e-mail poruke mrežnom administratoru ako je došlo do pokušaja proboja u sustav te drugi, aktivni odgovor koji se aktivira kod neovlaštenog ulaska u sustav koji gasi kompromirani servis kako bi napadač izgubio vezu sa sustavom. Svi agenti međusobno komuniciraju pomoću SSL-a. Sama arhitektura se sastoji od četiri različita agenta. Prvi agenti su agenti za praćenje sustava. Ti agenti prate događaje incijalizirane pomoću alata za praćenje zapisa (logova). Drugi agent jest agent koji služi za prenošenje podataka. Oni su zaduženi za slanje obavjesti administratoru putem emaila ili za kreiranje novog agenta koji služi za obranu od napada. Treći agenti su reaktivni agenti. Oni se kreiraju od strane agenata za prenošenje podataka. Četvrti agenti su perzistivni agenti koji se koriste za praćenje nestabilnosti mreže i da garantira distribuciju podataka te njihovu perzistenciju.

Kako autori žele primijeniti način rada ljudskog obrambenog sustava na računalni sustav dali su prikaz usporedbe rada ta dva sustava. On je dan slikom:



Slika 3.1: Računalni sustav vs imunološki sustav (Azzedine Boukerche, 2007)

Iz navedene slike može se vidjeti koji dio računalnog sustava služi kako bi se implementirao određeni dio obrambenog sustava kod ljudi. Za sva četiri sustava su odgovorni agenti koji prate i modificiraju sustav prema potrebama. Za testiranje sustava autori su koristili sustav s korištenjem SSL-a te sustav koji nije koristio SSL za prenošenje informacija. Kao rezultat performansi može se vidjeti da su performanse bolje kad se koristi SSL socket nego kada se ono ne koristi. Također su analizirali i veličinu paketa s obzirom na potrebno vrijeme prenošenja podataka putem mreže. Također su uvidjeli da povećanjem veličine paketa samo vrijeme se ne povećava linearno. Osim što su analizirali odaziv također su pratili i koliko je sam IDS učikovit, odnosno, koliko prijetlji je uspješno detektirao te koliko prijetnji je detektirao a da nisu bile prijetnje (eng. false positive). Ima sveukupno četiri stanja koje IDS može generirati. To su *normal events, false positives, true positives* i *true negative*. Prema njihovim rezultatima sustav je detektirao 53.13% normalnih događaja, 94.67% lažnih pozitiva, 5.33% točnih pozitiva te 46.87% točnih negativa.

Poglavlje 4

Praktični dio

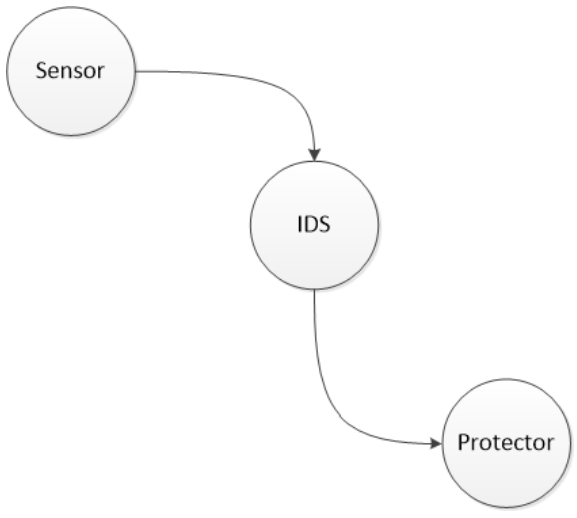
U ovome poglavlju opisan je praktični dio projekta. Dan je uvod u sam praktični dio, razrada rada agenata te način korištenja.

4.1 Opis agenata

Sustav agenata se sastoji od tri tipa agenata. Prvi agent, nazvan *sensor* koristi se za praćenje okoline. Drugi agent, nazvan *IDS* služi za komunikaciju između agenata. Treći agent, nazvan *protector* jest zadnji agent u lancu koji se koristi za obranu računala i/ili servisa od napada koje je *sensor* detektirao.

Svi agenti imaju reaktivno ponašanje, dok su drugi i treći agenti zapravo dogovorljivi agenti iako se mogu definirati i kao reaktivni agenti. Reaktivni agenti su oni agenti koji reagiraju na podražaje iz okoline, dok dogovorljivi agenti izvršavaju svoju zadaću kad dobe pobudu od drugog agenta putem poruke. Time se mogu definirati vrste agenata u ovome projektu.

Sama shema višeagentnog sustava se može dati sljedećom slikom, tj. dijagramom:



Slika 4.1: Slika višeagentnog sustava

Poruke generira prvi agent nazvanim *Sensor*. U cijelom sustavu može biti više takvih agenata. Zatim se ta poruka prosljeđuje agentu *IDS*. Agent pretraži sustav za sve agente tipa *Protector* te mu šalje obavjest o tome što treba štititi. Poruka završava kod agenta *Protector* te tu završava njen životni

vijek. Zadnji agent se koristi kako bi se zaštitio određeni servis i/ili računalo, IDS se koristi kako bi se prosljeđivale poruke svim protectorima te kako bi se održavala komunikacija dok Sensor prati okolinu te šalje obavjesti agentima. Svi agenti će biti detaljnije opisani u pratećem tekstu.

4.2 Sensor

Sensor je prvi agent koji pokreće cijeli val slanja poruka. Sam agent se sastoji od više faza kroz koje on prolazi. Na početku se instancira klasa samog agenta, nakon što se agent pokrene prvo se registrira u direktorij servisa te se zatim doda njegovo ponašanje. Ponašanje se sastoji od slanja poruke agentu IDS. Cijeli kôd je objašnjen u nastavku:

Početak prvog agenta je definiran sljedećim kodom:

```
if __name__ == "__main__":
    for arg in sys.argv[ 1: ]:
        naziv, sadrzaj = arg.split( '=' )
        if naziv == 'ime':
            ime = sadrzaj + '@127.0.0.1'
        if naziv == 'server':
            server = sadrzaj + '@127.0.0.1'
        if naziv == 'warning':
            warning = sadrzaj
        if naziv == 'service':
            service = sadrzaj
        if naziv == 'ip':
            remoteIP = sadrzaj
        if naziv == 'level':
            level = sadrzaj

    a = Posiljatelj(ime, "tajna" )

    a.server = server
    a.serverAdresa = 'xmpp://' + server
    a.warning = warning
    a.service = service
    a.localIP = socket.gethostbyname(socket.gethostname())
    a.ip = remoteIP
    a.level = level

    a.start()
    a._kill()
```

Iz navedenog dijela programskog koda može se vidjeti da agent treba zaprimiti parametre putem komandne llinije. Ti parametri su *ime, server, warning, service, ip* i *level*. Parametar *ime* koristi se kako bi se agenti navelo ime agenta, *server* označava adresu servera gdje se nalazi IDS, *warning* kao opis samog sigurnosnog propusta koji se dogodio, ili napada, *service* naziv servisa nad kojim se stvorio sigurnosni problem, *ip* koji definira IP adresu na kojoj servis poslužuje svoje usluge, i *level* koji opisuje

razinu opasnosti koju je donio sigurnosti problem. Zatim se instancira objekt klase *Posiljatelj* koji je zapravo agent. Zatim se postave varijable objekta prema primljenim varijablama s konzole. Na kraju se pokreće agent pomoću naredbe **a.start()**.

```
class Posiljatelj( spade.Agent.Agent ):
    class InformBehav( spade.Behaviour.OneShotBehaviour ):
        def _process( self ):
            primatelj = spade.AID.aid( name=self.server,
 addresses=[self.serverAdresa])
            self.msg = spade.ACLMessage.ACLMessage()
            self.msg.setPerformative( "inform" )
            self.msg.setOntology( "IDS" )
            self.msg.setLanguage( "english" )
            self.msg.addReceiver( primatelj )
            self.msg.setContent( 'service:' + self.service + '#warning:'
+ self.warning + '#localIP:' + self.localIP + '#remoteIP:' + self.ip
+ '#level:' + self.level)
            self.myAgent.send( self.msg )
            print "Message send!"

    def _setup(self):
        print "Senzor: starting ...."
        b = self.InformBehav()
        b.serverAdresa = self.serverAdresa
        b.server = self.server
        b.warning = self.warning
        b.service = self.service
        b.localIP = self.localIP
        b.ip = self.ip
        b.level = self.level

        self.agnt = DfAgentDescription()
        self.agnt.setAID( self.getAID() )
        SD = ServiceDescription()
        SD.setName( self.server )
        SD.setType( 'sensor' )
        self.agnt.addService( SD )
        self.registerService( self.agnt )

        self.addBehaviour( b, None )
```

U navedenom isječku može se vidjeti ostatak kôda skripte. Kad se instancira objekt klase *Posiljatelj* prvo se kreira novi objekt klase *InformBehav*. Zatim se definiraju njegove varijable ovisno o primljenim varijablama s konzole. Zatim se kreira opis agenta te njegovog servisa. Pomoću ovog dijela moguće je pretražiti repozitorij servisa za tim agentom. Ta funkcionalnost se koristi kod traženja agenata treće vrste. Zatim se doda ponašanje agenta koji je tipa *OneShotBehaviour* što znači da će se samo jednom

izvršiti. Navedeno ponašanje se svodi na kreiranje poruke sa sadržajem koji je bio preuzet s konzole tokom pokretanje skripte. Jezik poruke je englesko, ontologija je **IDS**. Poruka se šalje pomoću funkcije *send*. Nakon što je agent kreirao poruku i poslao ju završava sa svojim radom unutar main funkcije s naredbom *_kill*.

4.3 IDS

Drugi agent koji je zapravo središnji agent je agent koji nema svojeg kraja, odnosno završetka. Od kad se agent pokreće konstantno osluškuje za nove poruke koje mu agent tipa *Sensor* može poslati te ovisno o primljenoj poruci kreira novu poruku koju šalje svim agentima tipa *Protector*. Ovaj agent je agent tipa raktivnih agenata, točnije, dogovorljivih agenata. Također, ovisno o značajnosti prijetnje agent šalje email poruku mrežnom administratoru putem Google Mail-a. Rad samog agenta biti će prikazan kroz objašnjenje kôda.

```
if __name__ == "__main__":
    a = Primatelj( "idsserver@127.0.0.1", "secret" )
    a.start()
```

U glavnoj funkciji, main funkciji kreira se nova instanca klase Primatelj s oznakom *a*. Navedena instanca klase jest zapravo agent koji djeluje s nazivom *idsserver* na adresu *127.0.0.1* koja je ujedno i localhost adresa računala. Lozinka za spoj na agenta je *secret*. Agent se pokreće naredbom **start**.

Zatim se izvršava sljedeći dio kôda:

```
    def _setup(self):
        rb = self.SvePoruke()
        self.setDefaultBehaviour( rb )

        self.agnt = DfAgentDescription()
        self.agnt.setAID( self.getAID() )
        SD = ServiceDescription()
        SD.setName( 'coordinator' )
        SD.setType( 'ids' )
        self.agnt.addService( SD )
        self.registerService( self.agnt )

        ids_template = spade.Behaviour.ACLTemplate()
        ids_template.setOntology( "IDS" )
        mt=spade.Behaviour.MessageTemplate(ids_template)

        ab = self.PorukeIDSa()
        self.addBehaviour( ab, mt )
```

Iz navedenog isječka koda može se vidjeti instanciranje klase. Prvo se instancira objekt klase SvePoruke te se objekt pridodaje kao predloženo ponašanje agenta. Zatim se generira opis agenta, odnosno njegovog servisa. Tip servisa je označen kao *ids*. Ova opcija se koristila kako bi se IDS agent mogao jednostavno pronaći u skupini agenata. Zatim se on registrira na Spade-u. Nakon registracije definira se predložak poruka koje agent osluškuje. Defirano je da se agent aktivira kod svake poruke

čija je ontologija *IDS*. Zatim se instancira novi objekt klase *PorukeIDSa* te se dodaje novo ponašanje na agent gdje se definira objekt i predložak poruke nad koje se ponašanje okida.

```
class PorukeIDSa( spade.Behaviour.Behaviour ):
        pravila = []
```

Prvo se definira naziv klase te se definira lista s nazivom *pravila* koja se kasnije koristi za definiranje svih definiranih pravila u IDS-u.

```
        def _process( self ):

            self.msg = None
            self.msg = self._receive( True )
            if self.msg:
                aad = spade.AMS.AmsAgentDescription()
                aad.ownership = "BUSY"
                result = self.myAgent.modifyAgent(aad)
                if result:
                    print "BUSY"
```

Nakon početka procesa prvo se agent definira kao zaposleni agent.

```
                lista = self.msg.content.split("#")
                for elem in lista[0:]:
                    naziv, sadrzaj = elem.split( ':' )
                    if naziv == 'warning':
                        warning = sadrzaj
                    if naziv == 'service':
                        service = sadrzaj
                    if naziv == 'remoteIP':
                        remoteIP = sadrzaj
                    if naziv == 'level':
                        level = sadrzaj
                    if naziv == 'localIP':
                        localIP = sadrzaj
                    if naziv == 'level':
                        level = sadrzaj
```

Pomoću *for* petlje iz primljene poruke se razdjeljuju sve informacije koje su potrebne IDS-u kako bi mogao obraditi prijetnju. Za svaki element liste se provjerava kojeg je tipa par te ovisno o tome se definira vrijednost varijable. Navedene varijable će se kasnije koristiti kod *Protector* agenta te kod provjere razine prijetnje.

```
                print self.msg.content
                self.pravila.append(self.msg.content)
                print self.pravila
```

Zatim se ispiše trenutno pravilo, pravilo se doda u listu pravila te se ispiše lista svih pravila kako bi se mogao kontrolirati rad same aplikacije.

```
                if(int(level) > 3):
                    print 'High threat!'
                    print 'Sending email...'
                    fromaddr = 'tdukaric.vas@gmail.com'
                    toaddrs  = 'tdukaric.vas@gmail.com'
                    msg = "Service= %s \nWarning= %s \nRemote IP= %s
\nLocal IP= %s \nLevel= %s " % (service, warning, remoteIP,
localIP, level)
                    username = 'tdukaric.vas'
                    password = 'nikolina20'
                    server = smtplib.SMTP('smtp.gmail.com:587')
                    server.starttls()
                    server.login(username,password)
                    server.sendmail(fromaddr, toaddrs, msg)
                    server.quit()
                    print 'Mail sent!'
```

Ukoliko je razina prijetnje veća od 3 ispisuje se obavjest da se radi o prijetnji visoke razine te se kreira email adresa. Email adresa se šalje s adrese **tdukaric.vas@gmail.com** na adresu **tdukaric.vas@gmail.com**. U navedenom dijelu je definirana i lozinka i korisničko ime te na koji poslužitelj se spaja kako bi se poslala email adresa. Nakon što je email uspješno poslan korisnika se obavještava o uspješnosti slanja emaila.

```
                a = spade.Agent.Agent('trazi@127.0.0.1', 'tajna')
                a.start()

                sd = spade.DF.ServiceDescription()
                sd.setType('minion')
                dad = spade.DF.DfAgentDescription()
                dad.addService(sd)
                rezultat = a.searchService( dad )
```

Kreira se novi agent koji pretražuje direktorij servisa za agentima čiji je tip *minion*. Taj tip je definiran nad svim agentima tipa *Protector*. Pomoću funkcije **searchService** moguće je pronaći sve agente koji odgovaraju definiranom opisu. Lista agenata se zatim sprema u listu koja se naziva *rezultat*.

```
                for i in rezultat:
                    primatelj = i.getAID()
                    inf = spade.ACLMessage.ACLMessage()
                    inf.setPerformative( "inform" )
                    inf.setOntology( "minion" )
                    inf.setLanguage( "english" )
                    inf.addReceiver( primatelj )
                    inf.setContent( 'service:' + service + '#warning:'
+ warning + '#localIP:' + localIP + '#remoteIP:' + remoteIP
```

```
+ '#level:' + level)
                    a.send( inf )
                    print "Message send!"
```

Nakon što se dobi popis svih agenata koji mogu primiti poruku svima se šalje poruka. Petlja *for* se koristi kako bi se prošlo kroz polje agenata te da bi se svakome poslala poruka. Svaka poruka se zasebno kreira za svakog agenta te se ispisuje obavjest o poslanoj poruci agentima. Ovime se omogućuje jednostavno proširenje sustava s dodatnim agentima koji štite servise i/ili poslužitelje unutar IDS sustava te time postaje lagano proširiv.

```
                a._kill()

                aad = spade.AMS.AmsAgentDescription()
                aad.ownership = "FREE"
                result = self.myAgent.modifyAgent(aad)
                if result:
                    print "FREE"
```

Na kraju agent za pretraživanje prekida s radom te se agent *IDS* oslobađa, odnosno, definira mu se status *FREE*. Time se završava slanje poruke te cijeli ciklus agenta. Zatim agent miruje dok agent *Sensor* ne pošalje drugu poruku.

4.4 Protector

Zadnji agent, s nazivom *Protector* se koristi kao agent koji je zadužen za obranu servisa i/ili računala ovisno o poruci koju mu pošalje IDS. Samu skriptu agenta je potrebno modificirati prema potrebama poslužitelja, odnosno, prema potrebama servisa koje on nudi. Zbog tog je navedeni agent, odnosno skripta, predstavljen samo kao ljuska cijele skripte. U kasnijem dijelu će biti prikazan dio gdje je potrebno dodati dio implementacije kako bi skripta zapravo štitila samo računalo i/ili servis.

```
if __name__ == "__main__":
    for arg in sys.argv[ 1: ]:
        naziv, sadrzaj = arg.split( '=' )
        if naziv == 'ime':
            ime = sadrzaj + '@127.0.0.1'
        if naziv == 'server':
            server = sadrzaj + '@127.0.0.1'
    a = Primatelj( server, "secret" )
    a.naziv = ime
    a.start()
```

Skripta počinje s main funkcijom te prima dva parametra iz konzole, a to su *ime* i *server*. Iz parametra imena dobi naziv samog agenta dok iz parametra *server* dobi naziv agenta pod kojim se izvršava. Zatim se instancira objekt klase **Primatelj** s nazivom i adresom te lozinkom. Zatim se agent pokreće.

```
    def _setup(self):
        rb = self.SvePoruke()
```

```
        self.setDefaultBehaviour( rb )

        self.agnt = DfAgentDescription()
        self.agnt.setAID( self.getAID() )
        SD = ServiceDescription()
        SD.setName( self.naziv )
        SD.setType( 'minion' )
        self.agnt.addService( SD )
        self.registerService( self.agnt )
        print "registriran!"

        minion_template = spade.Behaviour.ACLTemplate()
        minion_template.setOntology( "minion" )
        mt=spade.Behaviour.MessageTemplate(minion_template)

        ab = self.Minion()
        self.addBehaviour( ab, mt )
```

Unutar početnog dijela agenta prvo se definira opis agenta tj. njegovog servisa te se ono registrira u bazu servisa, odnosno agenata. Svaki agent ovog tipa je potrebno registrirati kako bi agent *IDS* mogao pronaći odgovarajuće agente kojima treba proslijediti obavjest senzora. Nakon što se definira opis agenta, odnosno servisa definira se predložak po kojim agent prati poruke koje se šalju unutar sustava. U predlošku se definira ontologija poruka na vrijednost *minion*. Zatim se kreira instanca klase *Minion* koji definira ponašanje agenta te se pridodjeljuje opis poruke koji okida navedeno ponašanje agenta te koji nosi informacije koje su potrebne agentu.

```
    class Minion( spade.Behaviour.Behaviour ):
        def _process( self ):

            self.msg = None
            self.msg = self._receive( True )
            if self.msg:
                aad = spade.AMS.AmsAgentDescription()
                aad.ownership = "BUSY"
                result = self.myAgent.modifyAgent(aad)
                if result:
                    print "BUSY"


                lista = self.msg.content.split("#")
                for elem in lista[0:]:
                    naziv, sadrzaj = elem.split( ':' )
                    if naziv == 'warning':
                        warning = sadrzaj
                    if naziv == 'service':
                        service = sadrzaj
```

```
                if naziv == 'remoteIP':
                    remoteIP = sadrzaj
                if naziv == 'level':
                    level = sadrzaj
                if naziv == 'localIP':
                    localIP = sadrzaj
                if naziv == 'level':
                    level = sadrzaj

            print self.msg.content
            aad = spade.AMS.AmsAgentDescription()
            aad.ownership = "FREE"
            result = self.myAgent.modifyAgent(aad)
            if result:
                print "FREE"
```

Klasa *Minion* sadrži metodu **_process** koja se izvršava kod pokretanja ponašanja agenta. Nakon što se pokrene skripta prvo se dohvati poruka te se spremi u varijablu *self.msg*. Zatim se stanje agenta definira kao zauzeti agent te se ispisuje obavjest da je agent statusa zauzet. Zatim se u listu *lista* spremaju parovi vrijednosti iz poruke te se unutar *for* petlje definiraju vrijednosti varijabli ovisno o članovima liste. Nakon *for* petlje potrebno je definirati dodatno ponašanje agenta ovisno o time što agent treba štititi. Ukoliko se radi o agentu koji treba definirati nova pravila u vatrozidu moguće je koristiti library *py-pf* kako bi se definirala nova pravila u Packet Filteru. Packet Filter se nalazi na svim BSD-baziranim sustavima kao što se netBSD, freeBSD ili OS X. Ili je moguće upravljati servisima na način da ih ugase ukoliko je napadač uspio ugroziti cijeli sustav pomoću određenog servisa. U ovom slučaju samo se ispiše sadržaj zaprimljene poruke te se oslobađa agent kako bi mogao nastaviti sa svojim radom.

Poglavlje 5

Prikaz rada aplikacije

Kako se sustav sastoji od tri agenata u ovome dijelu biti će unakrsno pozivani svi agenti. Svi pozivi agenata su točno definirani te će biti opisani u ovome dijelu.

Kod inicijalnog pokretanja sustava, prije nego što se pokrene agent tipa *Sensor* potrebno je pokrenuti agente *IDS* i jedan ili više agenata tipa *Protector*. Sam redoslijed pokretanja agenata kod njih nije bitan, bitno je samo da su pokrenuti prije nego što se pokreće agent tipa *Sensor*. Agenta tipa *IDS* pokreće se naredbom

```
.\ids.py
```

Samom agentu, odnosno skripti nije potrebno definirati niti jedan ulazni parametar. Kad se pokrene agent u konzoli (terminalu) prikaže se sljedeći tekst:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./ids.py
Message received!
Message received!
Message received!
```

Navedbene primljene poruke su poruke iz svijeta agenata koje nisu vezane iz sam IDS.

Zatim se pokreće agenti tima *Protector*. Oni se mogu pokrenuti prije ili nakon IDS-a. Njima je potrebno navesti dodatne parametre tokom pokretanja kao što je ime i server. Primjer pokretanja agenata:

```
./protector.py ime="minion1" server="mini1"

./protector.py ime="minion2" server="mini2"
```

Navedena dva primjera su mogući načini kako pravilno pokrenuti agente. Kako bi se agenti, odnosno skripte pravilno pokrenule potrebno je proslijediti dva parametra, *ime* i *server*. Oni se koriste za definiranje naziva agenta te adrese agenta.
Primjer poziva navedenih agenata s ispisom:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion1"
        server="mini1"
registriran!
Message received!
```

```
Message received!
```

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion2"
        server="mini2"
registriran!
Message received!
Message received!
```

Navedena dva izlaza su početni izlazi iz skripti. Prvo agent javlja da se uspješno registrirao, a zato obavjesti da je primio dvije poruke koje nisu vezane uz njegovo funkcioniranje.

Nakon što su svi potrebni agenti pokrenuti pokreće se agent tipa *Sensor* koji šalje poruku agentu *IDS* o sigurnosnoj pogrešsci. Kad se taj agent pokreće potrebno je poslati niz argumenata kako bi se ona ispravno izvršila. Primjer poziva jednog agenta:

```
./sensor.py ime='sensor1' server='idsserver' warning='Opasnost!'
        service='apache2' ip='test' level='5'
```

Prema primjeru navedenog poziva može se vidjeti da je naziv agenta *sensor1*, naziv IDS servera kojem se informacija šalje *idsserver*, opis opasnosti je *Opasnost!*, naziv servisa je *apache2*, ip adresa je *test* te da je razina opasnosti *5*. Nakon što se navedeni senzor pokrenuo on je time pokrenuo val promjena koje su se dogodile u cijelom sustavu. Cijeli ispis poziva agenta:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./sensor.py ime='sensor1'
        server='idsserver' warning='Opasnost!' service='apache2'
        ip='test' level='3'
Senzor: starting . . .
Message send!
```

Nakon što je on pozvan prvo se dogodila promjena kod IDS-a:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./ids.py
Message received!
Message received!
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
['service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3']
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
Message send!
Message send!
FREE
Message send!
```

Kod njegovog ispisa može se vidjeti da je zaprimio poruku, da je postao zauzet, koje pravilo je došlo, te listu pravila. Također se može vidjeti da je poslao dvije poruke, svaku poruku zasebnom agentu tipa *Protector*. Oni su se također pokrenuli te se njihov izlaz promijenio na:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion1"
        server="mini1"
registriran!
Message received!
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
FREE
```

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion2"
        server="mini2"
registriran!
Message received!
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
FREE
```

Može se vidjeti kod oba primjera da su prihvatili poruke, ispisali su pravilo koje su zaprimili te oslobodili agente.

Napravljen je i drugi primjer pozivanja skripte *Sensor* koja je dobila drugačiji utjecaj na ostale agente, posebice na *IDS*. Poziv:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./sensor.py ime='sensor2'
        server='idsserver' warning='Opasnost!' service='apache2'
         ip='test' level='5'
Senzor: starting . . .
Message send!
```

Rezultat poziva skripte može se vidjeti kod odgovora agenta IDS koji se promijenio:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./ids.py
Message received!
Message received!
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
['service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3']
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:3
Message send!
Message send!
```

```
FREE
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:5
['service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
␣␣␣␣␣␣␣␣level:3', 'service:apache2#warning:Opasnost!#localIP:172.20.10.3#
␣␣␣␣␣␣␣␣remoteIP:test#level:5']
High threat!
Sending email...
Mail sent!
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
        level:5
Message send!
Message send!
FREE
```

Ovdje se može vidjeti razlika između poziva agenta *Sensor* s razinom prijetnje koja je manja ili jednaka 3 s obzirom na one koje su veće od 3. Ukoliko se desio incident koji ima značaj koji je veći od 3 šalje se obavjest mrežnom administratoru o prijetnji. Također se ispiše poruka da se radi o visokoj prijetnji. EMail koji je poslan na email adresu tdukaric.vas@gmail.com je oblika:

```
Return-Path: <tdukaric.vas@gmail.com>
Received: from Tomislavs-MacBook-Pro.local ([46.188.180.5])
        by mx.google.com with ESMTPSA id o43sm95675286eef.12.2014.02.04
        for <tdukaric.vas@gmail.com>
        (version=TLSv1 cipher=RC4-SHA bits=128/128);
        Tue, 04 Feb 2014 17:43:01 -0800 (PST)
Message-ID: <52f19725.435f0e0a.4796.ffffe1d7@mx.google.com>
Date: Tue, 04 Feb 2014 17:43:01 -0800 (PST)
From: tdukaric.vas@gmail.com
Service= apache2
Warning= Opasnost!
Remote IP= test
Local IP= 172.20.10.3
Level= 5
```

Pomoću navedenog emaila mrežni administrator može vidjeti o kojoj se pogrešci radi, na kojoj adresi se nalazi servis, koji servis je u pitanju, razina rizika te kratki opis upozorenja. Može se vidjeti koji agent je poslao poruku, tj. na kojoj IP adresi se nalazi.

Nakon što je email uspješno poslan šalju se obavjesti ostalim agentima tipa *Protector*. S time se mijenja njihov izlaz u:

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion1"
server="mini1"
registriran!
Message received!
```

```
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
level:3
FREE
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
level:5
FREE
```

```
Tomislavs-MacBook-Pro:komunikacija dux$ ./protector.py ime="minion2"
server="mini2"
registriran!
Message received!
Message received!
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
level:3
FREE
BUSY
service:apache2#warning:Opasnost!#localIP:172.20.10.3#remoteIP:test#
level:5
FREE
```

Poglavlje 6

Kritički osvrt

Reaktivni i dogovorljivi agenti su agenti čija se primjena još nije do kraja istražila. Kao i sami agenti, još nije definirana prava definicija što je to agent u domeni računalstva. Iako je navedeno područje relativno novo područje znanosti ono se intenzivno razvija jer višeagentni sustavi donose dosta prednosti te rješavaju veliki dio problema. Primjerice, reaktivni i mrežni te dogovorljivi agenti zajedno mogu tvoriti umjetnu inteligenciju za relativno kompleksne strojeve kao što su roboti, iako su agenti sami za sebe primitivni i jednostavni. Čak i kod računalnih mreža postoji velika primjena višeagentnih sustava. U ovome radu bio je dan primjer jednog takvog sustava koji se brine o prenošenju informacija o mogućim prijetnjama. Također, velika prednost agenata, posebice reaktivnih agentata jest njihova jednostavna adaptibilnost novim situacijama ili okruženju. Mrežni administrator može jednostavno promijeniti agenta tako da mu koristi za ono što mu je potrebno u danom trenutku. Također ne mora toliko misliti o tome jesu li agenti međusobno povezani te kako se odnosi situacija kad agenti mogu sami uspostaviti veze s ostalim agentima te vršiti izmjenu podataka. Kao posebna primjena višeagentnih sustava može se dati baš kod obrana raznim računalnih sustava gdje jednostavna skalabilnost, proširivost te iznimna izdržljivost višeagentnih sustava je potrebna kako bi se osigurala postojana zaštita koja je konzistentna. Kako agenti međusobno komuniciraju moguće je složiti distribuiranost pravila te poveznica kako ispad jednog agenta nebi ugrozio pad ostalih agenata ili cijelog sustava. To je ono što je nedostajalo kod obrane informacijskog sustava. Primjeri takvih sustava su dani u člancima koji su bili prethodno navedeni te sam projekt koji je u ovoj dokumentaciji opisan koji je bio napravljen prema navedena dva članka. Sustavi koji su skalabilni, lako prolagodljivi te potpomažu trenutne sustave obrane. Također, praktična izvedivost nije kompliciranа kad trenutno postoji cjelovita podrška za izradu višeagentnih sustava te računala su i više nego dovoljno jaka za održavanje takvih sustava, pogotovo zato jer se svi agenti ne izvode na centralnom poslužitelju nego se opterećenje održavanja sustava distribuira na sva računala koja sudjeluju u njemu. Danas se kod istraživanja područja višeagentnih sustava više pita što s njime napraviti, tj. gdje ih primijeniti a ne kako ih kreirato kad već postoje teorijske osnove za širok spektar različitih agenata. Trenutno je potrebna primjena tih teorijskih osnova na sustave koje je moguće jednostavno nadograditi na višeagentne sustave.

Poglavlje 7

Zaključak

Višeagentni sustavi kao IDS još nije dovoljno istraženo područje no prema rezultatima ovog projekta te prethodnih istraživanja može se reći da se višeagentni sustavi mogu primijeniti kod izrade IDS te IPS sustava. Prednosti koje višeagentni sustavi nude su zapravo nedostaci trenutnih ID sustava. Jedan od tih stvari je skalabilnost te otpornost na moguće napade na cijeli IDS sustav. Zbog toga što su agentu autonomni može se stvoriti cijeli IDS sustav koji je međusobno toliko povezan da pad jednog agenta ne znači da će pasti cijela mreža, nego samo jedan mali dio. Tu dolazi prednost pretrage svijeta agenata za agentima koji obavljaju dužnost IDS-a. Također se može uspostaviti i hijerarhijska struktura ukoliko se želi definirati lakša kontrola nad samim IDS-om. Također, kreiranje općenitih modela IDS-a koji se mogu prilagođavati situacijama ovisno o tome kako mrežni administrator to želi je dobar početak za izradu autonomnog sustava koji je prilagodljiv bilo kojoj situaciji ili mjestu iskorištenja agenta. Primjerice, u ovome projektu napravljen je općeniti model IDS-a koji se može prilagoditi potrebama administratora.

Uvođenjem mrežnih agenata može se proširiti trenutna mreža agenata te time učvrstiti poveznice između agenata te očuvati konzistentnost pravila koja se propagiraju cijelom mrežom. Ovo područje je još relativno novo područje primjene višeagentnih sustava te postoji veliki potencijal u iskorištenju takvih sustava u aktivnoj obrani računalnim sustava i mreža. Pitanje je kamo će razvoj ovog područja doći te gdje će se još pronaći primjena za ovakve sustave, pogotovo za reaktivne agente koji se relativno jednostavno implementiraju, jednostavni su te su time i robusni.

Bibliografija

Azzedine Boukerche, Renato B. Machado, K.R.J.J.B.M.S.M.S.N., 2007. *An agent based and biological inspired real-time intrusion detection and security model for computer network operations*. Computer Communications 30 (2007).

B. Mukherjee, T.L.H. and Levitt, K.N., 1994. *Network intrusion detection*. IEEE Network, 26–41.

Ferber, J., 1994. *Simulating with Reactive Agents*. Many Agent Simulation and Artificial Life, 8–28.

Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, D.I.E.S.D.Z., 1998. *An Architecture for Intrusion Detection using Autonomous Agent*. Computer Security Applications Conference, 1998. Proceedings. 14th Annual.

Nwana, H.S., 1996. *Software Agents: An Overview*. Knowledge Engineering Review, Vol. 11, 3, 1–40.