Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave

Upotreba genetskih algoritama u sustavima za otkrivanje napada

Igor Krnić

diferencijski ispit iz kolegija Operacijski sustavi 2

Zagreb, 2009.

Sadržaj

Uvod

Moderno društvo sve više postaje ovisno o velikoj količini informacija kao i o što bržem pristupu i procesiranju istih. Svijet računala, raznih računalnih mreža i interneta omogućio je skoro trenutačni pristup informacijama iz različitih, te uglavnom dalekih lokacija. U takvom umreženom društvu očita je potreba za reguliranjem pristupa i manipulacije računalnim resursima te naravno informacijama. Sve to dovelo je do raznih ideja i mehanizama čiji je cilj što sigurniji računalni sustav. Uzme li se u obzir kako se okolina, računalni sustavi, napadi na njih pa čak i kriteriji sigurnosti kontinuirano mijenjaju jasno je da je računalna sigurnost aktualno, ali i izazovno područje.

Standardni mehanizmi (sustavi zaštičeni lozinkama, enkripcija, vatrozidovi,...) koji čine prvu liniju obrane, sve se češće nadopunjuju pomoćnim mehanizmima čija je uloga pospješiti i preventivni dio, ali i djelovati kada dođe do sigurnosnog propusta. Sustavi za otkrivanje napada predstavljaju jedan od novijih i obečavajućih mehanizama u računalnoj sigurnosti. Njihova je glavna uloga sprečavanje i otkrivanje računalnih napada. Paradigme umjetne inteligencije uvelike se koriste unutar takvih sustava. Razna istraživanja su pokazala da bi upotreba genetskih algoritama mogla imati važnu ulogu u razvoju "laganih" i točnih sustava u usporedbi sa klasičnim sustavima za otkrivanje napada temeljenim na paradigmama mašinskog učenja i data mininga.

Cilj je ovog seminara definirati osnovne pojmove vezane za sustave za otkrivanje napada i navesti nekoliko primjena genetskih algoritama u njihovom razvoju.

Računalna sigurnost

Računalna sigurnost grana je računalne znanosti zadužena za osiguranje sigurnosti računalnih sustava. Sigurnost računalnih sustava definira se na osnovu ispunjavanja određenih sigurnosnih zahtjeva. Najstandardniji i najčešće spominjani zahtjevi su:

- *Povjerljvost* – informacije moraju biti dostupne samo ovlaštenim korisnicima
- *Integritet* – informacije ne smiju biti uništene, promijenjene ili izgubljene od strane neovlaštenog korisnika
- *Raspoloživost* – informacije moraju biti dostupne u trenutku kada ih korisnik zatraži
- *Autentičnost* – korisnik se mora na jedinstven način identificirati
- *Autorizacija* – za korisnika se mora definirati skup dopuštenih radnji te informacija
- *Neporecivost* – korisnik ne smije biti u mogućnosti poreći valjanost poruke koju je poslao

Prijetnje sigurnosti računalnih sustava mogu dolaziti sa različitih izvora kao što su prirodne sile (npr. poplave, potresi), nesreće (npr. požari), nedostatak resursa (npr. pomanjkanje električne energije), grešaka u samom sustavu (npr. greške u programskoj logici, neispravnost fizičkih komponenti sustava ili njihov prestanak rada) te od zlonamjernih osoba često zvanih uljezima (eng. intruders).
Zaštitni mehanizmi od prijetnji kao što su prirodne sile, nesreće i nedostatak resursa svode se na standardnu problematiku zaštite imovine i nisu svojstvene računalnim sustavima te ih nećemo posebno navoditi. Greške u programskoj logici se nažalost uz sve veće i kompliciranije sustave, teško mogu izbjeći, ali suvremene metode za razvoj softvera svakim danom napreduju. O ovoj problematici također nećemo govoriti iako su upravo programske greške često meta napada. Naglasak u seminaru biti će prvenstveno na prijetnjama od strane uljeza.
Uljeze uglavnom svrstavamo u dva tipa:

1. *vanjski uljezi* - nemaju autorizirani pristup sustavu kojeg napadaju
2. *unutrašnji uljezi* - imaju autorizirani pristup sustavu uz određene restrikcije.

Kako se zaštititi od uljeza? Standardni zaštitini mehanizmi poput autentifikacije, autorizacije, enkripcije, vatrozidova djeluju kao prva linija zaštite ali nažalost nisu dostatni. Što ako je lozinka kompromitirana? Tada autentifikacija ne može pomoći. Što ako je vatrozid krivo konfiguriran? Što u slučajevima kada sam uljez ima prava pristupa, ali ih zloupotrebljava? Očito je da postoji potreba za dodatnim mehanizmima koji bi detektirali napade te djelovali kao zadnja linija obrane u slučaju da sve ostalo zakaže.

Teorija računalne sigurnosti definira pojam ***napada*** kao pokušaj ili sam čin kompromitiranja nekog od osnovnih sigurnosnih zahtjeva: povjerljivosti, raspoloživosti, besprijekornosti (integriteta), autentičnosti, autorizacije ili neporecivosti.
Neki od najčešćih napada su: napadi na mrežne servise, napadi na aplikacjie, neovlašteno

povećanje ovlasti, neovlaštena prijava i pristup informacijama te napadi zlonamjernim aplikacijama tj. virusima, crvima i trojanskim konjima.
Postupak ***otkrivanja napada*** je najčešće, algoritmizirani pokušaj da se otkrije napad.
Napade možemo prema tipu upada razvrstati u šest kategorija:

- *Pokušaj upada* (eng. attempted break-in):
  najčešće se otkriva zbog netipičnog uzorka postupaka ili kršenja sigurnosnih postavki.

- *Napad lažnim predstavljanjem* (eng. masquerade attack):
  najčešće se otkriva zbog netipičnog uzorka postupaka ili kršenja sigurnosnih postavki.

- *Proboj sigurnosnog sustava* (eng. penetration of security control system):
  najčešće se otkriva praćenjem specifičnih uzoraka postupaka.

- *Propuštanje* (eng. leakage):
  najčešće se otkriva zbog netipične upotrebe ulazno/izlaznih resursa

- *Odbijanje usluge* (eng. denial of service):
  najčešće se otkriva zbog netipične upotrebe sistemskih resursa.

- *Zlonamjerno korištenje* (eng. malicious use):
  najčešće se otkriva zbog netipičnog uzorka postupaka ili kršenja sigurnosnih postavki te korištenja privilegiranih akcija ili vrijednosti.

Slično, prema načinu otkrivanja napada imamo sljedeću podjelu:

- *Otkrivanje nepravilnosti* (eng. anomaly detection):
  napadi se otkrivaju zbog nepravilnog (sumnjivog) ponašanja i korištenja resursa. Na primjer, ako korisnik najčešće koristi računalo između 9h i 17h onda se korištenje istog u kasne večernje sate smatra neuobičajenim. Ovaj način nastoji opisati uobičajena odnosno dozvoljena ponašanja te kao nepravilno označava sve ono što u to ne spada.

- *Otkrivanje zlonamjernog korištenja* (eng. misuse detection):
  otkrivanje se temelji na podudaranju sa poznatim uzorcima napada koji koriste slabosti u samom sustavu ili aplikativnom softveru. Za razliku od prethodnog načina ovdje se direktno traži sumnjiva ponašanja.

Neovisno o načinu klasifikacije napada najčešće se za njihovo otkrivanje koriste *podaci o događajima* (eng. audit data) koje bilježi sustav. *Datoteka sa nizom događaja* (eng. audit trail) je kronološki zapis svih događaja u sustavu pomoću koje je moguće rekonstruirati okvirno stanje i aktivnost sustava u nekom promatranom periodu.
Glavne odnosno najčešće korištene metode za otkrivanje napada su upravo statistički pristup kod otkrivanja nepravilnosti te traženje dobro poznatih napada kod otkrivanja zlonamjernog korištenja. Svaki od pristupa ima svoje prednosti i mane koje zaslužuju par komentara.

Otkrivanje nepravilnosti kreće od pretpostavke da su napadi podskup skupa svih sumnjivih

radnji odnosno da je svaki napad ujedno i sumnjiva radnja. Ova se pretpostavka čini razumnom uzevši u obzir da vanjski uljez ne zna uzorak normalnog ponašanja. No što ako se radi o unutarnjem uljezu ili o nizu normalnih radnji koje u sumi daju napad ili samo sumnjivu radnju? Kako odbaciti samo sumnjive radnje koje nisu napadi? Prepoznaju li se svi? Imamo točno četiri slučaja:

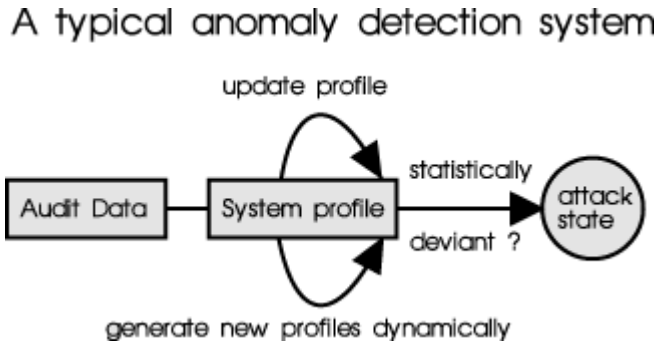
1. Radnje koje su napadi, ali nisu nepravilne (čest je i naziv netočno negativne (eng. false negatives)).

2. Radnje koje nisu napadi, ali jesu nepravilne (čest je i naziv netočno pozitivne (eng. false positives)).

3. Radnje koje nisu napadi i nisu nepravilne (čest je i naziv točno negativne (eng. true negatives)).

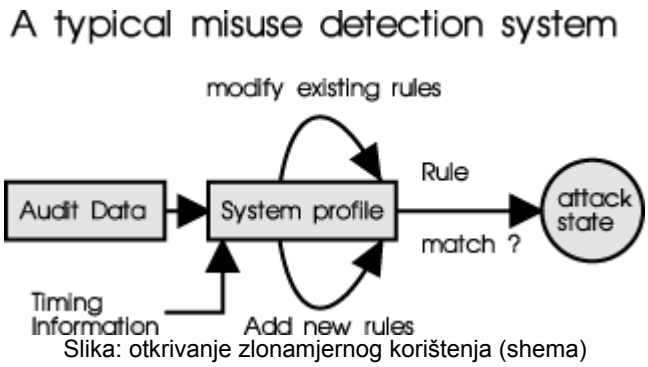
4. Radnje koje jesu napadi i jesu nepravilne (čest je i naziv točno pozitivne (eng. true positives)).

Kada želimo izbjeći neotkrivanje radnji pod 1 (netočno negativne), prilično je jasno da postupak otkrivanja mora biti dosta rigorozan tj., granica koja definira sumnjivo ponašanje je postavljena nisko. Naravno to može rezultirati da se neke od radnji pod 2 (netočno pozitivne) proglase napadima. Previše pogrešaka u postupku otkrivanja dovodi u pitanje može li se postupak automatizirati jer mora postojati dodatni objekt (čovjek) koji bi istražio greške. Također, ovakav način otkrivanja napada zahtijeva kontinuirano praćenje i ažuriranje uzoraka koji definiraju normalno ponašanje što može biti udar na performanse.



Slika: otkrivanje nepravilnosti (shema)

Otkrivanje zlonamjernog korištenja ide suprotnim putem. Pretpostavka je da se poznati napadi mogu po koracima dovoljno precizno definirati i kodirati tako da se razne varijacije istog napada mogu prepoznati. Glavni problemi kod ovog pristupa su kako precizno definirati i kodirati poznate napade, a da se pritom izbjegne mogućnost proglašavanja normalne radnje sumnjivom ili čak napadom te činjenica da se traže sličnosti samo sa već poznatim napadima. Dakle, nove napade je u biti nemoguće prepoznati.



Slika: otkrivanje zlonamjernog korištenja (shema)

Otkrivanje napada može se provoditi ručno ili automatizirano. Oba načina naravno koriste podatke o događajima. Kako datoteke sa događajima znaju biti prilično velike (pogotovo u većim sustavima) ručno otkrivanje jednostavno nije dovoljno brzo te je u biti neupotrebljivo. Automatizirano otkrivanje napada provodi se uglavnom putem sustava poznatih pod imenom Sustavi za otkrivanje napada .

Sustavi za otkrivanje napada

Kao što i samo ime kaže glavna zadaća sustava za otkrivanje napada (eng. Intrsuion detection system - IDS) je otkrivanje računalnih napada. Točnije ***sustav za otkrivanje napada***, je hardwer ili softwer čija je zadaća otkrivanje napada, otkrivanje zlonamjernog korištenja računala i generiranje obavijesti te nešto rjeđe, poduzimanje neke vrste protudjelovanja. Sustave se slikovito uspoređuje sa kućnim alarmnim sustavima jer "dižu alarm" kad se dogodi sumnjiva situacija.
Svrha sustava za otkrivanje napada nije da djeluje kao samostalno (i jedino) sigurnosno rješenje nego kao dodatna karika u sigurnosnom lancu. Zamišljen kao zadnja linija obrane kad primarni tj. kada preventivni načini obrane (identifikacija, enkripcija, vatrozidovi,...) zakažu.
Radi lakšeg shvaćanja objasnimo razliku između vatrozida i sustava za otkrivanje napada. Svrha vatrozida je da kontrolira tok informacija prema mreži i iz nje. U skladu sa prije danim primjerom kućnog alarmnog sustava, vatrozid se može shvatiti kao ograda ili čuvar kuće. Sa druge strane svrha sustava za otkrivanje napada je da otkrije je li mreža napadnuta odnosno u gorem slučaju, je li sigurnosni sustav već probijen.

Klasičan sustav za otkrivanje napada sastoji se od senzora koji traže sigurnosne prijetnje, upravljačkog dijela koji služi za praćenje događaja i uzbuna te središnjeg uređaja koji na temelju podataka o događajima te modela za otkrivanje napada generira uzbune te eventualno poduzima neku radnju. U jednostavnijim sustavima sve tri komponente dio su istog uređaja.

Najopćenitije ih svrstavamo prema:

1. *metodi otkrivanja napada*:
    - sustav za otkrivanje napada temeljen na ***otkrivanju nepravilnosti*** (eng. anomaly-based IDS, AIDS) – traži nepravilnosti odnosno odstupanja od normalnih događaja.
    - sustav za otkrivanje napada temeljen na ***otkrivanju zlonamjernog korištenja*** (eng. signature-based IDS, SIDS) – traži uzorke poznatih napada među događajima.
2. *smještaju unutar sustava*:
    - ***Mrežno orijentirani*** (eng. network-based IDS, NIDS) – promatra mrežni promet tražeći zlonamjerni sadržaj.
    - ***Lokalno orijentirani*** (eng. host-based IDS, HIDS) – promatra radnje na jednom računalu.

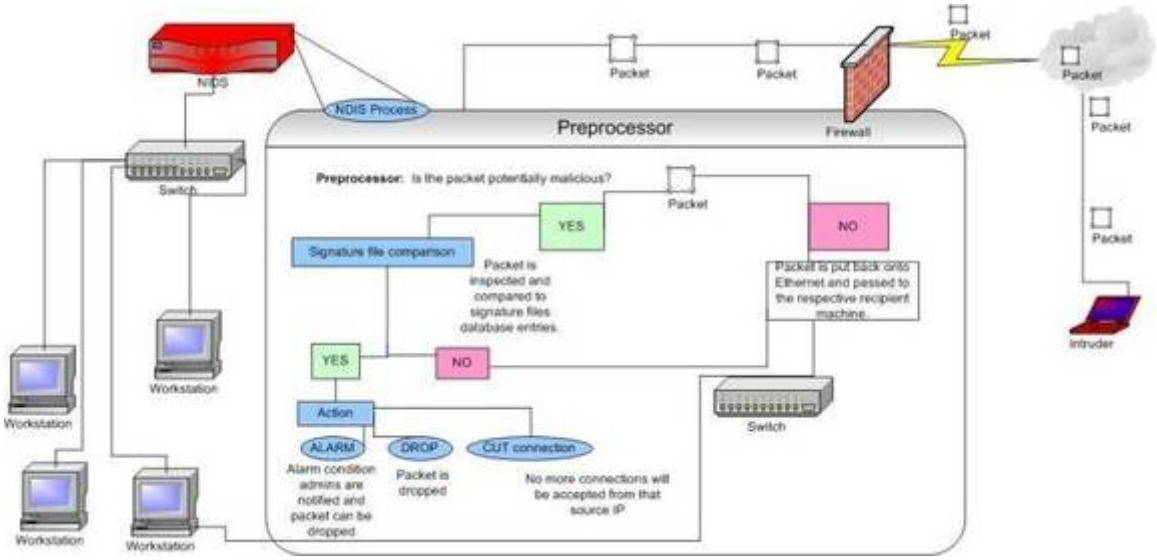
Da bi se iskoristile prednosti i HIDS i NIDS sustava često se stvaraju i hibridni sustavi. Također, postoji i podjela na pasivne i reaktivne sustave. Pasivni sustavi samo generiraju obavijesti dok reaktivni sustavi (eng. intrusion prevention system, IPS) poduzimaju i akciju (npr. zatvaraju konekciju, reprogramiraju vatrozid,...).

Navedimo i nekoliko primjera sustava za otkrivanje napada. Najpoznatiji besplatni ovakav sustav je Snort (www.snort.org). Snort je primjer mrežno orijentiranog sustava koji kombinira obje metode za otkrivanje napada. Također besplatan sustav, ali lokalno orijentiran je OSSEC (www.ossec.net). Od komercijalnih sustava izdvojimo lokalno orijentirani sustav Symantec Intruder Alert (www.symantec.com) te mrežno orijentirane Cisco Secure IDS sustave (www.cisco.com).
Izdvojimo još i besplatan STIDE sustav za otkrivanje napada (www.cs.unm.edu/~immsec/data-sets.htm) koji je orijentiran na otkrivanje nepravilnosti. Upravo zbog otvorenosti koda i metode otkrivanja STIDE će biti testni sustav u jednom od kasnije obrađenih primjera.

Kroz poglavlja koja slijede navesti ćemo osnovne karakteristike mrežno i lokalno orijentiranih sustava za otkrivanje napada. Razlike između sustava ovisno o metodi otkrivanja napada nećemo posebno navoditi jer je problematika spomenuta u prošlom poglavlju. Ipak, navesti ćemo aktualne metode korištene u AIDS i SIDS sustavima.

Mrežno orijentirani sustavi za otkrivanje napada

Mrežno orijentirani sustavi za otkrivanje napada smještaju se kao aktivni dio unutar strukture računalne mreže. Namjena im je da analiziraju mrežni promet na segmentu na kojem su smješteni. Smještaj NIDS-a u mreži (npr. unutar ili izvan vatrozida) doprinosi njegovoj efikasnosti te postoje različita stajališta o tome gdje bi trebao biti smješten. Međutim, pokazano je da više NIDS-a na istoj mreži učinkovitije djeluje što donekle smanjuje problem njihovog smještanja. Također, jasno je da su kritična mjesta u mreži, pogotovo ona "sklona" sigurnosnim propustima, dobri kandidati za smještaj NIDS-a.

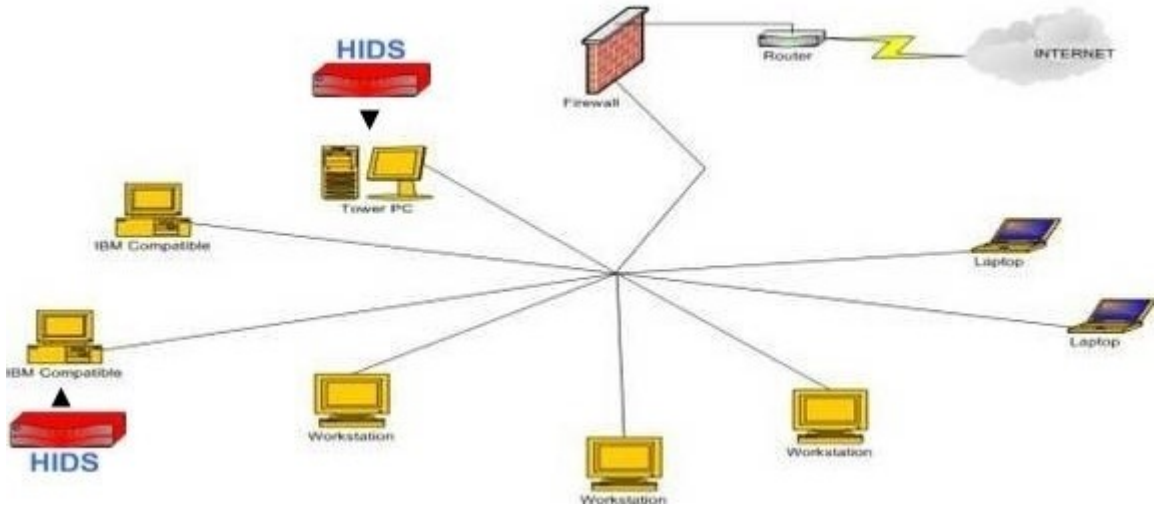


slika: smještanje mrežno orijentiranog sustava za otkrivanje napada

Mrežno orijentirani sustavi imaju i svoje nedostatke. Kriptirane pakete sa napadima koji putuju unutar mreže ovi sustavi ne mogu otkriti.

Lokalno orijentirani sustavi za otkrivanje napada

Lokalno orijentirani sustavi za otkrivanje napada bili su prvi razvijeni i implementirani ovakvi sustavi. Sustav je smješten na lokalnom računalu i cilj mu je pratiti aktivnost samo na njemu. Najčešće se smještaju na serverska ili informacijski osjetljiva računala.



slika: smještanje lokalno orijentiranog sustava za otkrivanje napada

Lokalno računalo najbolja je lokacija za odgovor na napad. Također, sustav prati događanja na samo jednom računalu te nije opterećen velikom količinom informacija pa je efikasniji. Vrlo je efikasan protiv unutrašnjih uljeza. Sa druge strane, može zauzeti dosta resursa na lokalnom računalu (procesor, radna memorija). Također, velika mreža sa puno računala na kojima je smješten ovakav sustav dovela bi do problema u skupljanju i obrađivanju podataka o napadima.

Tehnike za razvoj sustava za otkrivanje napada

U nastavku ćemo navesti nekoliko klasičnih tehnika koje se koriste za razvoj sustava za otkrivanje napada. Kod metode otkrivanja nepravilnosti često se koriste statistički pristup, predviđanje uzorka te paradigme umjetne inteligencije kao na primjer neuronske mreže, dok su kod metode otkrivanja zlonamjernog korištenja bitnije tehnike analiza promjene stanja te otkrivanje napada na osnovu modela. Sve navedene tehnike kratko ćemo opisati radi konzistencije i boljeg razumijevanja problematike samih sustava za otkrivanje napada.

Statistički pristup

Kod statističkog pristupa (eng. statistical approache) prvo se promatra aktivnost korisnika te se generiraju uzorci (profili) njihova ponašanja. Uzorci korisničkih profila generiraju se na temelju mjera (pokazatelja) kao na primjer: mjera aktivnosti, mjera korištenja datoteka, mjera zauzeća procesora, broja mrežnih konekcija u nekom periodu... Uzorci se kontinuirano ažuriraju kako bi se mogućnost greške svela na minimum.

Prilikom traženja nepravilnosti izračunava se vrijednost (mjera) koja pokazuje koliko je trenutno odstupanje od promatranog korisničkog profila. Ako je vrijednost veća od dopuštenog praga trenutna aktivnost označava se nepravilnom.

Glavna prednost ove tehnike je da vremenom uzorci korisničkih profila postaju sve točniji. Također, omogućava upotrebu dobro razrađene teorije statistike. No, postoje i neke mane. Problem postavljanja praga (previsok generira previše lažnih uzbuna, a nizak propušta neke nepravilne radnje), mogućnost da se sustav trenira od strane uljeza da ne primjećuje neke nepravilnosti (često se ovaj problem rješava hibridnim sustavima, tj. sustav ujedno otkriva i zlonamjerno korištenje), statistika ne uzima u obzir redoslijed događaja, problem odabira karakterističnih mjera.

Predviđanje uzoraka

Predviđanje uzoraka (eng. predictive pattern generation) kreće od pretpostavke da događaji nisu nasumični nego da postoji neka pravilnost u redoslijedu ili uzorku. Kako se kontekst događaja uzima u obzir otkrivanje napada je bolje. Opet, u početnom stadiju generira se skup pravila koji se kontinuirano ažurira. Za primjer uzmimo pravilo oblika

$$D1 – D2 – D3 \Longrightarrow (D4 = 65\%, D5 = 25\%, D6 = 10\%).$$

Pravilo kaže da (prema do sada generiranom uzorku) nakon slijeda događaja D1, D2, D3 (redom) najvjerojatnije slijedi jedan od događaja D4, D5, D6 sa navedenim vjerojatnostima. Nepravilnost se otkriva ako postoji niz događaja koji se poklapa sa lijevom stranom nekog pravila, ali događaj koji bi trebao slijediti odskače od pravila.

Primijetimo da je ovako zadan skup pravila vrlo fleksibilan i lako se modificira. Također, uzorci manje vjerojatnosti se konstantno eliminiraju dok ne ostane skup pravila koji dobro opisuje ponašanje danog korisnika. Lakše se otkriva pokušaj treniranja od strane uljeza. Glavni je nedostatak što postoji mogućnost neotkrivanja napada koji se ne poklope sa lijevom stranom niti jednog pravila.

Neuronske mreže

Ideja je istrenirati neuronsku mrežu (eng. neural networks tako da može na temelju n prethodnih radnji ili događaja, predvidjeti sljedeću korisnikovu radnju odnosno događaj. Jednom istrenirana neuronska mreža daje mjeru koliko korisnikova radnja odstupa od naučenog profila.

Prednost je ovog pristupa što se ne traži bilo kakve statističke pretpostavke o podacima nad kojim se radi te što neuronske mreže dobro izlaze na kraj sa šumovima u podacima. Nažalost, pokazalo se da je broj n (broj prethodnih radnji na temelju kojih se izvodi zaključak) osjetljiv. Mali n vodi ka mreži koja nije baš upotrebljiva, dok veliki n zagušuje mrežu sa nepotrebnim podacima. Također, proces treniranja mreže često se svodi na metodu pokušaja i pogrešaka (topologija, težine).

Analiza promjene stanja[1]

Postupak je u biti sličan djelovanju konačnog automata. Zlonamjerne radnje modeliraju se kao niz promjena stanja unutar promatranog sistema. Dva stanja su susjedna ako postoji neka radnja koja vodi iz jednog stanja u drugo. Niz zlonamjernih radnji dovesti će automat iz početnog dozvoljenog stanja u nedozvoljeno.

Primijetimo da ovakav model dozvoljava da se na temelju trenutnog stanja i radnji koje iz njega mogu slijediti automatski zaključi koja mogu biti rezultantna stanja što omogućuje i preventivno djelovanje. Sa druge strane, samo zlonamjerne radnje koje su produkt niza uzastopnih radnji biti će prepoznate što isključuje ili jako otežava prepoznavanje kompliciranijih napada.

Otkrivanje napada na osnovu modela[2]

Misao vodilja je da se buduća stanja sustava mogu pretpostaviti ovisno o trenutnim aktivnostima odnosno stanju sustava. Podrazumijeva se postojanje baze uzoraka ili opisa napada od kojih je svaki zapisan kao niz radnji. Otkrivanje ide na sljedeći način. Odabere

1 eng. State transition analysis
2 eng. Model-based intrusion detection

se neki podskup skupa napada i pokuša se naći početne radnje tih napada u datoteci događaja. Ovaj posao obavlja dio modela koji se zove prorok (eng. anticipator). Ako se pronađu pokazatelji postojanja potencijalnih napada (postoje radnje u datoteci događaja koje odgovaraju početnom dijelu nekog napada iz odabranog podskupa) tada se posao prebacuje na drugi dio modela koji se zove planer (eng. planner). Njegova je zadaća da generira nove datoteke događaja koje će odgovarati hipotetskom stanju u kojem bi se napadi (koji su zadovoljili u prethodnom koraku) stvarno odvili. Sada na scenu stupa treći dio modela, interpreter (eng. interpreter) koji u originalnoj datoteci događaja traži pokazatelje iz novo-generiranih datoteka. Postupak se dalje nastavlja sve dok sustav ne dođe do praga nakon kojega objavi postojanje napada.

Kako prorok i planer znaju točno što traže velika količina redundantnih podataka se može odbaciti, što naravno pozitivno djeluje na performanse. Također, model može predvidjeti potencijalne napade i djelovati preventivno. Međutim, postoji i nekoliko problema. Opisi napada u bazi se moraju moći lako prepoznati i njihovi dijelovi se ne smiju poklapati sa normalnim radnjama te naravno, moguće je otkriti samo poznate napade.

Postoje još brojne tehnike koje se koriste za razvoj sustava za otkrivanje napada. Navedimo neke praćenje tipki (eng. keystroke monitoring), usporedba uzoraka (eng. pattern matching), data mining, neizrazita logika (eng. fuzzy logic) te još priličan broj paradigmi umjetne inteligencije. Fokus u ovom seminaru biti će stavljen na genetske algoritme. Pokazalo se da upotreba genetskih algoritama dovodi do točnih a opet relativno jednostavnih sustava. Također, njihova možda najveća prednost je u činjenici da su sposobni sami ponuditi zadovoljavajuće rješenje za probleme za koje nije jasno kako bi se naspram njih trebalo postaviti. U nastavku ćemo iznijeti par primjera upotrebe genetskih algoritama u razvoju sustava za otkrivanje napada.

Genetski algoritam za mrežno otkrivanje napada

Uvod

Rad mrežno orijentiranog sutava za otkrivanje napada temelji se na prikupljanju i analizi mrežnog prometa. Promatrana varijanta koristi mrežni promet kako bi na temelju njega definirala skup pravila koji mrežnu konekciju karakteriziraju zlonamjernom. Možemo li nekako efikasno primijeniti genetski algoritam? Pokazati ćemo kako genetski algoritam upotrijebiti da bi se evoluirao skup pravila koji karakteriziraju zlonamjerno korištenje konekcije.

Genetski algoritam

Svaku mrežnu konekciju karakerizira određen broj svojstava kao što su polazna i završna IP adresa, broj korištenog porta, odabrani protokol, trajanje konekcije, količina prenesenih podataka, količina primljenih podataka... Sva ta svojstva mogu se iskoristiti kao polazna točka za otkrivanje napada. Uzmimo na primjer slučaj kada je polazna IP adresa na listi nedozovljenih ili kada trajanje konekcije premašuje dopušteni prag. Tada jednostavno usporedbom vrijednosti promatrane konekcije sa oglednim skupom podataka možemo utvrditi postoji li sumnja da se konekcija koristi u zlonamjerne svrhe. No, situacija ne mora biti toliko jednostavna. Moguće je da tek veliki broj svojstava odnosno njihovih vrijednosti te i neki drugi parametri definiraju sumnjivu konekciju. U svakom slučaju vidimo da svojstva konekcije možemo shvatiti kao preduvjet za izvođenje zaključka. Tako dolazimo do jednostavnih pravila oblika $if-then$ koji definiraju što poduzeti ako konekcija zadovolji neki uvjet. U našem primjeru pravila ćemo korisiti da bi karakterizirali zlonamjerne konekcije i koristiti ćemo genetski algoritam da bi evoluirali skup pravila koji će kasnije sustav za otkrivanje napada koristiti u realnom vremenu.

Ovako modelirani sustav za otkrivanje napada možemo shvatiti kao sustav zasnovan na pravilima (eng. rule-based system), a genetski algoritam kao pomoćnika u generiranju referentnog skupa pravila.

Dakle, promatrati ćemo pravila sljedećeg oblika:

$$if\ (\ condition\ )\ \ then\ \ (\ act\ )$$

Pogledajmo primjer jednostavnog pravila koje kaže da je konekcija koja ima polaznu IP adresu $191.23.34.124$ sumnjiva i da je treba prekinuti.

$$if\ (\ source\ IP : 191.23.34.124\ )\ \ then\ \ (\ stop\ the\ \ connection\ )$$

Cilj je genetskog algoritma da definira što točnija pravila čije će uvjete zadovoljavati samo sumnjive konekcije. Podaci na temelju kojih će genetski algoritam evaluirati generirana pravila od velike su važnosti. Što su podaci točniji i bolje definiraju sumnjive konekcije to će i algoritam proizvesti bolja pravila. Za sakupljanje podataka o mrežnom prometu postoji veliki broj aplikacija (kao Wireshark, www.wireshark.com ili Snort, www.snort.com). Sakupljene podatke klasificiraju stučnjaci sa velikim iskustvom upravo kako bi relevantnost podataka bila što veća.

Sada kada znamo što želimo evoluirati pogledajmo kako ćemo kodirati pravila u kromosome. Zbog jednostavnosti prilikom definiranja strukture kromosoma ograničiti ćemo se samo na 8 svojstava mrežnih konekcija definiranih u sljedećoj tablici.

Attribute	Range of Values	Example Values	Descriptions
Source IP address	0.0.0.0~255.255.255.255	d1.0b.**.** (209.11.??.??)	A subnet with IP address 209.11.0.0 to 209.11.255.255
Destination IP address	0.0.0.0~255.255.255.255	82.12.b*.** (130.18.176+?.??)	A subnet with IP address 130.18.176.0 to 130.18.255.255
Source Port Number	0~65535	42335	Source port number of the connection
Destination Port Number	0~65535	00080	Destination port number, indicates this is a http service
Duration	0~99999999	00000482	Duration of the connection is 482 seconds
State	1~20	11	The connection is terminated by the originator, for internal use
Protocol	1~9	2	The protocol for this connection is TCP
Number of Bytes Sent by Originator	0~9999999999	0000007320	The originator sends 7320 bytes of data
Number of Bytes sent by Responder	0~9999999999	0000038891	The responders sends 38891 bytes of data

slika: promatrana svojstva mrežene konekcije

Prvi stupac tablice definira promatrano svojstvo, drugi skup vrijednosti koje on može poprimiti, treći primjer vrijednosti danog svojstva, a zadnji kratki opis. Pravilo

if {the connection has following information: source IP address 209.11.??.??; destination IP address: 130.18.176+?.??; source port number: 42335; destination port number: 80; connection time: 482 seconds; the connection is stopped by the originator; the protocol used is TCP; the originator sent 7320 bytes of data; and the responder sent 38891 bytes of data }
then {stop the connection}

bi prema prethodnoj tablici kodirali kromosomom

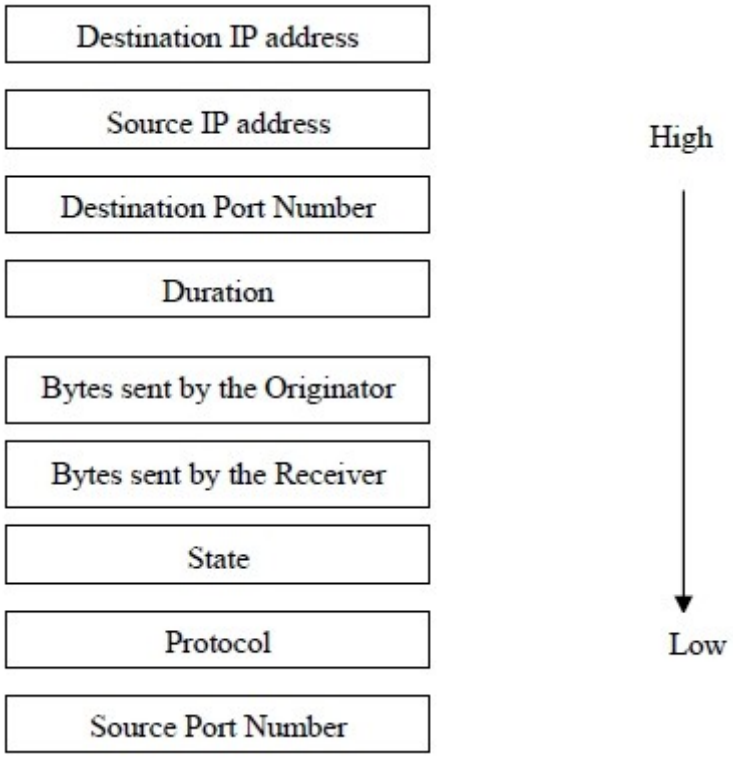
(d, 1, 0, b, -1, -1, -1, -1, 8, 2, 1, 2, b, -1, -1, -1, 4, 2, 3, 3,
5, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 4, 8, 2, 1, 1, 2, 0, 0, 0,
0, 0, 0, 7, 3, 2, 0, 0, 0, 0, 0, 0, 3, 8, 8, 9, 1)

Dakle, pravilo predstavljamo uređenom 57-orkom. Poredak gena unutar kromosoma odgovara poretku svojstava mrežne konekcije u tablici. Za prikaz IP adresa koristi se heksadecimalni prikaz. Znakovi ? i * u tablici se koriste kao specijalni znakovi koji mogu poprimiti bilo koju vrijednost dok prilikom kodiranja u tu svrhu koristimo znak -1. Namjera je jasna, znak -1 označava skup vrijednosti za koje dano pravilo vrijedi.

Dobrota svakog kromosoma biti će testirana na oglednim podacima. Ako kromosom bude odgovarao sumnjivoj ili zlonamjernoj konekciji vrijednost funkcije doborote biti će veća, a u suprotnom, ako odgovara normalnoj tada će biti manja. Odmah je jasno da ne postoji jednostavno i jedinstveno pravilo koje bi odvajalo sumnjive od normalnih konekcija. Naprotiv, da bi se dobila pravila koja nisu samo popis u prošlosti otkrivenih sumnjivih konekcija potrebno je pronaći pravilnosti u skupu podataka. Ovakav problem idealan je za genetske algoritme. No, uvelike ovisi prvenstveno o funkciji dobrote. Pogledajmo kako je definirana.

Prilikom konstrukcije funkcije dobrote autori su se vodili činjenicom da nisu sva svojstva jednako važna za konekciju te da ih treba različito vrednovati. Najvažnija svojstva su svakako polazna i odredišna IP adresa jer one određuju izvor i cilj napada. Zatim slijede

broj porta, trajanje konekcije i drugi. Sljedeća slika daje odnos važnosti promatranih svojstava.



slika: promatrana svojstva poredana po važnosti od više prema nižoj

Na temelju važnosti svojstva definira se njegova težina. Prvi korak u izračunu funkcije dobrote je određivanje ukupne težine. Ukupnu težinu računamo kao sumu težina svih svojstava ali pri tome uzimamo u obzir samo ona svojstva čija vrijednost zadovoljava uvjete referentnog skupa podataka. Težina gena identična je težini svojstva čiji je on dio. Dakle, svi geni koji kodiraju isto svojstvo imaju identičnu vrijednost težine. Formula za ukupnu težinu je sljedeća:

$$T_{ukupno} = \sum_{1}^{57} M_i T_i$$

gdje je M_i 0 ili 1 u ovisnosti o tome poklapa li se vrijednost i-tog gena sa vrijednostima u oglednom skupu podataka, a T_i je težina i-tog gena. Sada treba izračunati odstupanje ukupne težine od mjere opasnosti danog kromosoma (konekcije). Mjera opasnosti također je podatak koji se dobije usporedbom konekcije sa oglednim podacima.

$$\Delta = |T_{ukupno} - T_{mjeraopasnosti}|$$

Ako je došlo do odstupanja tada se kromosom kažnjava s time da se uzme u obzir koliko je danu konekciju (kromosom) bilo teško otkriti u slučaju da je zlonamjerna.

$$kazna = (\Delta * tezinaotkrivanja / 100)$$

Na kraju dobrotu kromosoma defniramo kao

$$dobrota = 1 - kazna$$

što je vrijednost između 0 i 1 .

Nakon što smo uspješno definirali funkciju dobrote recimo i nekoliko riječi o korištenim genetskim algoritmima. Za razliku od standardnog genetskog algoritma koji populaciju vodi prema jednom lokalnom maksimumu autori članka predlažu korištenje tehnike traženja potprostora (eng. niching technique) koja se koristi za traženje više različitih lokalnih maksimuma. I ovdje je povučena analogija sa prirodom gdje u istom okruženju, ali različitim potprostorima žive različiti organizmi. Motivacija za korištenje leži u činjenici da želimo što više različitih (ne nužno bliskih) rješenja kako bi pokrili što više mogućih napada.

Ova tehnika sadrži dvije bitne metode za evoluciju jedinki, razdijeljivanje (eng. sharing) i gomilanje (eng. crowding). Gomilanjem se eliminiraju jako slične jedinke (npr. ostavi se jedna, a jako slične njoj se zamijene) i zamjenjuju novima kako populacija ne bi konvergirala prema jednom lokalnom ekstremu. Razdijeljivanjem se smanjuje dobrota jedinki koje imaju jako slične dijelove opet sa ciljem da se populacija uputi prema različitim ekstremima. Sa određivanje slličnosti dvaju kromosoma može se koristiti Hammingova udaljenost ili u ovom slučaju sličnost svojstava dviju konekcija.

Rezultati i zaključak

Kako se temeljita i opsežna testiranja predloženog genetskog algoritma moraju tek provesti nažalost nismo ih u stanju izložiti. Zainteresirani čitatelj kao polaznu točku za daljnje istraživanje obrađenog primjera može uzeti (f).

GASSATA

Uvod

GASSATA (Genetic Algorithm for Simplified Security Audit Trails Analysis (d) je alat kojim se automatizira postupak traženja napada pregledom datoteka sa događajima. Iako ga se zbog jednostavnosti (na primjer ne uzima u obzir kronologiju događaja) ne može okarakterizirati kao moderan sustav za otkrivanje napada ideja predložena od strane autora vrlo je interesantna i još jednom demonstrira efikasan način djelovanja genetskih algoritama na zahtijevnim problemima.

GASSATA otkriva napade metodom otkrivanja zlonamjernog korištenja. Među podacima o događajima traži se "najopasniji" napad. Kako je ovakva pretraga NP-potpun problem pristupilo se heurističkoj pretrazi skupa rješenja gdje glavnu ulogu ima genetski algoritam. Glavni dio ovog poglavlja biti će posvećen konstrukciji funkcije dobrote zbog činjenjice što upravo ona najbolje ilustrira ideju pretrage koja se krije u pozadini.

Genetski algoritam

Kako se otkrivanje napada provodi metodom otkrivanja zlonamjernog korištenja pretpostavka je da postoji baza podataka sa kodovima napada (u ovom slučaju napad se predstavlja nizom događaja). Cilj nam je naći "najopasniji" napad za koji postoji opravdana sumnja (u smislu postojanja odgovarajućih događaja) da se dogodio. U tu svrhu te sa ciljem što formalnije konstrukcije funkcije dobrote uvesti ćemo nekoliko oznaka:

- Sa $N(e)$ označimo broj različitih događaja koji su evidentirani putem datoteke sa događajima.
- Sa $N(a)$ označimo broj različitih napada evidentiranih u bazi podataka.
- AE neka je $N(e)\times N(a)$ matrica koja svakom napadu pridružuje događaje koji se u njemu pojavljuju. Točnije na mjestu i,j u matrici AE nalazi se broj događaja i koji se pojavljuju u napadu j (broj veći ili jednak nuli).
- Neka je R $N(a)$ -dimenzionalni vektor težina, gdje i-ta koordinata vektora R označava rizik odnosno opasnost napada i.
- Neka je O $N(e)$ -dimenzionalni vektor, gdje i-ta koordinata vektora O označava broj događaja tipa i unutar datoteke sa događajima.
- Neka je H $N(a)$ -dimenzionalni vektor nula i jedinica, gdje i-ta koordinata vektora označava pretpostavku da li se i-ti napad dogodio ili ne.

Zanima nas vektor H takav da je opasnost za sustav najveća odnosno da je skalarni produkt vektora R i H maksimalan. Naravno, postoje i dodatne ograde. Broj događaja (nekog tipa) koji se dogodio pod pretpostavkom postojanja napada unutar vektora H mora biti manji ili jednak ukupnom broju događaja (tog istog tipa) unutar datoteke sa događajima. U našim oznakama to bi bilo:

$$AE[i]\leq O[i],(1\leq i\leq N(a))$$

Očito, izraz $R\times H$ sigurno će biti dio funkcije dobrote. No, obzirom na prethodnu ogradu je li to dovoljno dobro? Što sa vektorima napada čija je opasnost velika, ali zbog prethodne ograde nisu realni? Takva rješenja trebalo bi dodatno penalizirati da ne bi usmjerili

genetski algoritam u krivom pravcu. Uvedimo još jednu oznaku.

Neka je $T(e)$ broj različitih tipova događaja za koje gornja ograda ne vrijedi tj. vrijedi:

$$AE[i] > O[i], (1 \le i \le N(a))$$

tada funkciju dobrote možemo izraziti na sljedeći način:

$$F(H) = a + (\sum_{i=1}^{N(a)} R[i]H[i] - bT(e)^2) \quad .$$

Koeficijenti a i b ostavljeni su za fino podešavanje funkcije dobrote dok je kvadrat broja nerealnih događaja uzet kao faktor koji bi nerealne napade trebao penalizirati. Također, spomenimo da autori u svom radu ne dozvoljavaju negativnu vrijednost funkcije dobrote te u takvom slučaju uzimaju da je vrijednost jednaka nuli.

Ovako definirana funkcija dobrote ima i nekoliko nedostataka. Kao prvo, pojavljuje se problem određivanja parametara a i b . Nadalje, funkcija dobrote preferira (zbog sume) rješenja sa maksimalnim brojem napada. No jednom kada se takav skup napada nađe svako daljnje pretpostavljanje novih napada vodilo bi lažno pozitivnim dojavama sustava. Ovaj i još nekoliko potencijalnih problema kao i predložene modifikacije definirane funkcije dobrote zainteresirani čitatelj može pronaći u članku (vidi (g)).

Kodiranje rješenja prilično je jednostavno. Ponovimo, tražimo vektore H nula i jedinica takve da je skalarni produkt $R \times H$ maksimalan (uz definirane ograde). Dakle, kromosom predstavljamo kao niz nula i jedinica duljine $N(a)$.

Autori članka nisu posebno naglašavali koje načine selekcije te koje genetske operatore su upotrebljavali, ali zbog jednostavnosti načina na koji su prikazana rješenja moguće je eksperimentirati sa raznim poznatim verzijama. U nastavku slijedi kratak opis testnog okruženja kao i dobivenih rezultata.

Rezultati i zaključak

Kao što smo ranije naveli GASSATA koristi datoteke sa događajima. Autori su koristili AIX operativni sustav koji sadrži konfigurabilan sigurnosni podsustav za generiranje i praćenje dogđaja.

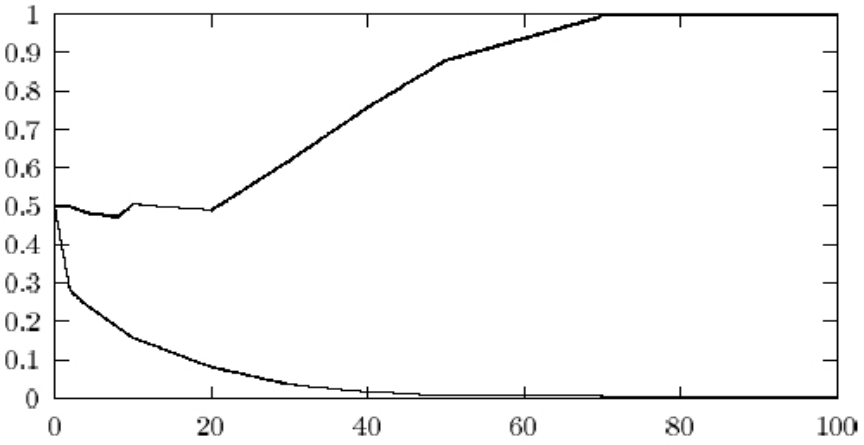
Datoteke sa događajima generirane su na temelju 30 minutnog promatranja. U slučaju dužeg perioda pokazalo se da algoritam brzo konvergira prema nekom jediničnom vektoru zbog ograda koje su ugrađene u funkciju dobrote. Također, datoteke se prije upotrebe filtriraju ovisno o korisniku sustava. U svrhu testiranja definirala su se četiri korisnika stupnjevana prema iskustvu u korištenju sustava. Slično, definirao se i skup od 24 različita napada. GASSATA je testiran na datotekama koje nisu sadržavale napade i na onima koje jesu. Kao pokazatelj efikasnosti uzeti su sljedeći omjeri:

- $T(p)$ je jednak broju jediniki unutar populaciie u kojima su bitovi koji odgovaraju napadima koji su se dogodili jednaki 1 naspram veličini populacije – dobro evoluirane jedinke.
- $T(a)$ je jednak broju jediniki unutar populaciie u kojima su bitovi koji odgovaraju napadima koji se nisu dogodili jednaki 1 naspram veličini populacije – loše evoluirane jedinke.

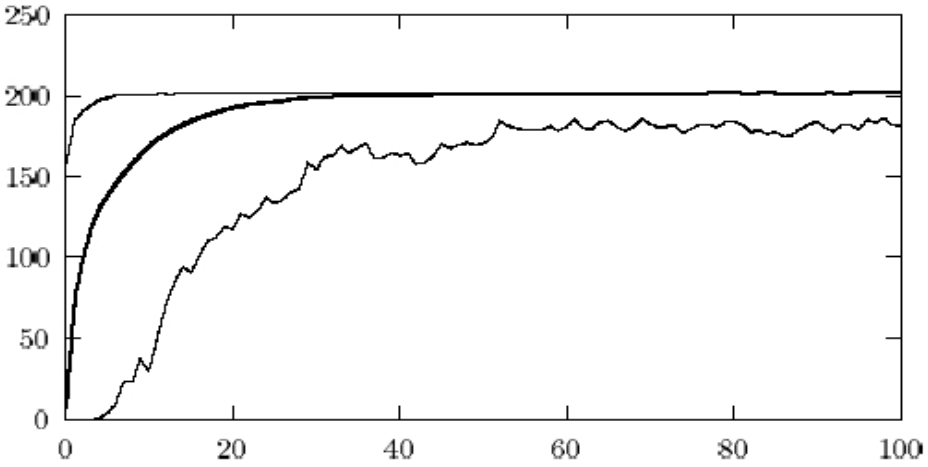
Želja je da $T(p)$ bude što bliži 1, a $T(a)$ 0.

Rezultati su pokazali da algoritam vrlo brzo konvergira prema rješenju. Također, $T(p)$ i

$T(a)$ su u svim testovima težili prema svojim idealnim vrijednostima. Pokazalo se i da broj napada u matrici AE traži prilagodbu broja generacija kako bi se sačuvala efikasnost te naravno i rast trajanja cijelog algoritma. Slijedi par grafova koji točnije prikazuju dobivene rezultate.



slika: kretanje $T(p)$ i $T(a)$ kroz generacije



slika: kretanje minimalne, prosječne i maximalne vrijednosti funkcije dobrote (u 10 ponavljanja te uz veličinu populacije 500, vjerojatnost križanja 0.6, vjerojatnost mutacije 0.002 i uz 2 napada prisutna u datoteci događaja) ovisno o broju generacija

Primjerom smo pokazali kako upotrijebiti genetske algoritme za otkrivanje napada pretraživanjem datoteke sa događajima. Algoritam traži najgoru moguću soluciju (tj. traži najopasniji poznati napad) i funkcijom dobrote vodi populaciju prema takvoj. Iako ovakva implementacija nije detaljno testirana (na većim količinama podataka) predstavlja zanimljivu mogućnost primjene genetskih algoritama.

Evoluiranje napada prekoračenja kapaciteta međuspremnika uz pomoć povratne informacije sustava za otkrivanje napada

Uvod

U ovom ćemo poglavlju pokazati kako se genetski algoritmi mogu iskoristiti u svrhu testiranja sustava za otkrivanje napada. Cilj je pokušati pomoću genetskih algoritama naći slabosti unutar samog sustava koje nisu jednostavno vidljive. U tu svrhu pokušati ćemo automatizmom "evoluirati" određene tipove napada sve do trenutka kada dovoljan broj istih sustav za otkrivanje napada više ne bude mogao označiti kao sumnjive.

Plan je sljedeći: naći ranjivu aplikaciju, odabrati ciljani napad kojeg želimo izvršiti, vidjeti kako sustav za otkrivanje napada reagira na ciljani napad te pokušati pomoću genetskog algoritma producirati napade ekvivalentne ciljanom napadu, ali koji neće biti proglašeni sumnjivima. Krenimo po redu. Kako je ipak primarni cilj ovog primjera pokazati kako genetskim algoritmom evoluirati skupinu napada, odabir ranjive aplikacije, odabir ciljanog napada te odabir samog sustava za otkrivanje napada stavljeni su u drugi plan.

Kao ranjiva aplikacija odabrana je Traceroute. Traceroute daje odgovor na pitanje kojim putem putuju paketi do udaljenog računala. Verziju 1.4a5 ove aplikacije bila je podložna napadu prekoračenja kapaciteta međuspremnika (eng. buffer overflow attack) koji omogućuje napadaču da si dodijeli administratorska prava na napadnutom sustavu (vidi http://www.securiteam.com/exploits/6A00A1F5QM.html). Povećanje ovlasti je upravo i ciljani napad u ovom primjeru, tj. napad koji želimo provesti mimo znanja sustava za otkrivanje napada. Već spomenuti STIDE sustav za otkrivanje napada zbog otvorenosti koda i metode otkrivanja napada (otkrivanje nepravilnosti) odabran je kao testni sustav. Upravo mjera odstupanja od normalnog ponašanja koju sustav (temeljen na otkrivanju nepravilnosti) vraća biti će ugrađena u genetski algoritam (točnije u funkciju dobrote) i pokušati će voditi evolucijski proces prema slabostima.

Kako bi izveli ciljani napad poslužit ćemo se tzv. maskiranim napadima (eng. mimicry attack). Maskirani napad je varijacija ciljanog napada koji pokušava proći pod normalno ponašanje. Dakle, ponovimo još jednom, cilj nam je evoluirati maskirane napade tako da prođu neopaženi ili uz što manju mjeru odstupanja od normalnog ponašanja.

Također, bitno je napomenuti da pristup u ovom primjeru neće pretpostavljati nikakve privilegirane informacije na temelju kojih bi "varanje" sustava za otkrivanje napada bilo jednostavnije, dapače u interesu je prikazati što realniju situaciju odnosno, onakvu situaciju u kakvoj bi se nalazio potencijalni napadač.

Genetski algoritam

Pogledajmo što trebamo definirati. Prvo, trebamo definirati što su nam kromosomi, zatim funkciju dobrote, postupak selekcije te na kraju genetske operatore (križanje i mutaciju).

Kako smo rekli da bi željeli evoluirati napade, jasno je da će kromosomi predstavljati upravo maskirane napade. Geni unutar kromosoma biti će pojedine instrukcije. Da bi smanjili mogućnost otkrivanja napada za njihovu definiciju pokušati ćemo koristiti samo instrukcije koje koristi aplikacija Traceroute. Namjera je očita, što se više približimo normalnom radu (u ovom slučaju radu aplikacije Traceroute) veća je vjerojatnost da napad neće biti otkriven. Pogledajmo koje instrukcije koristi aplikacija Traceroute prilikom svog rada.

System Call	Occurence	Frequency	System Call	Occurence	Frequency
gettimeofday	220	16.73%	mprotect	34	2.59%
write	142	10.8%	socket	29	2.21%
mmap	113	8.59%	recvfrom	28	2.13%
select	99	7.53%	brk	27	2.05%
sendto	99	7.53%	fcntl	26	1.98%
close	93	7.07%	connect	20	1.52%
open	86	6.54%	ioctl	15	1.14%
read	75	5.7%	uname	14	1.06%
fstat	73	5.55%	getpid	12	0.91%
munmap	49	3.73%	time	10	0.76%

slika: najčešće instrukcije koje aplikacija Traceroute izvodi

Na slici su prikazane frekvencije dvadeset instrukcija koje aplikacija Traceroute najčešće izvršava.

Bitno je primijetiti da korištenje ovih podataka ne pretpostavlja znanje o povjerljivim informacijama samog sustava za otkrivanje napada. Naime, dovoljno je upotrijebiti neki dijagnostički alat koji prati instrukcije poslane operativnom sustavu od strane aplikacija (autori navode aplikaciju Strace, http://sourceforge.net/projects/strace). Za konstrukciju maskiranih napada u našem primjeru mi ćemo se ograničiti na petnaest najčešće korištenih instrukcija. Sljedeća tablica definira arhitekturu instrukcija koje koristimo za izgradnju napada. Instrukcije su kodirane formatom fiksne dužine.

System Call	Parameter 1	Parameter 2
open	{"/etc/passwd", "/tmp/dummy"}	n/a
close	{"/etc/passwd", "/tmp/dummy"}	n/a
read	{"/etc/passwd", "/tmp/dummy"}	4 byte space address
write	{"/etc/passwd", "/tmp/dummy"}	{"toor::0:0:root:/root:/bin/bash", "Hello, world!"}
other	n/a	n/a

slika: arhitektura instrukcija za izgradnju napada

Prisjetimo se što točno želimo postići. Originalni napad koristi execve sistemski poziv koji poziva skriptu koja napadaču dodjeljuje administratorska prava. Kako napadnuta aplikacija Traceroute nikada ne koristi sistemski poziv execve velika je vjerojatnost da će sustav za otkrivanje napada otkriti ovaj napad. Sa druge strane, sljedeći niz instrukcija u potpunosti odgovara našim potrebama tj. ekvivalentan je učinku originalnog napada:

1. Otvori datoteku sa lozinkama (datoteka "/etc/passwd" na unix-oidnom operativnom sustavu koji se koristi u primjeru)
2. Dodaj liniju u datoteku koju napadaču omogućuje da se prijavi za rad na sistemu bez lozinke
3. Zatvori datoteku

Kako aplikacija Traceroute često koristi instrukcije open, close i write koje nam upravo trebaju veća je vjerojatnost da maskirani napad koji sadrži ove tri instrukcije prođe neopažen. Dakle, geneteski algoritam bi trebao voditi populaciju maskiranih napada prema onima koji sadrže ove tri instrukcije. Je li to dovoljno? Osim toga, očito nam je bitan i poredak navedenih instrukcija (na primjer: prvo se mora otvoriti ciljana datoteka da bi se u nju nešto upisalo), a ne smijemo zaboraviti ni činjenicu da želimo minimizirati mjeru odstupanja napada od normalnog ponašanja. Pa definirajmo funkciju dobrote kojom ćemo

pokušati usmjeravati maskirane napade prema konačnom rješenju. Iz prethodnih razmatranja da se naslutiti da će funkcija dobrote trebati vrednovati dvije stvari: ekvivalentnost traženom napadu te što manju mjeru odstupanja od normalnog ponašanja. Točnije, od maskiranih napada tražiti ćemo sljedeće:

1. Da sadrže instrukciju otvori datoteku "/etc/passwd", open("/etc/passwd").
2. Da sadrže instrukciju kojom si napadač dodjeljuje veća prava pristupa sustavu, write ("igor::0:0:root:/root:/bin/bash") (kreira korisnika igor koji ima admnistratorska prava).
3. Da sadrže instrukciju zatvori datoteku "/etc/passwd", close("/etc/passwd").
4. Da se instrukcija dodjeljivanja povlaštenih prava izvrši nakon instrukcije otvaranja datoteke.
5. Da se instrukcija zatvaranja datoteke izvrši nakon dodjeljivanja povlaštenih prava.
6. Da promatrani sustav za otkrivanje napada maskirani napada označi sa što manjom mjerom odstupanja od normalnog ponašanja.

Pseudo-kod funckije dobrote:

1. fitness = 0;
2. if ({open("/etc/passwd")} $\in$ maskirani_napad) then fitness++;
3. if ({open("igor::0:0:root:/root:/bin/bash")} $\in$ maskirani_napad) then fitness++;
4. if ({close("/etc/passwd")} $\in$ maskirani_napad) then fitness++;
5. if (open prije write) then fitness++;
6. if (write prije close) then fitness++;
7. fitness += (100 – mjera_odstupanja_od_normalnog_ponašanja) / 20;

Vidimo da je ukupan broj bodova koji se može postići deset. Od toga pet bodova otpada na ekvivalentnost ciljanom napadu, a preostalih pet na mjeru odstupanja od normalnog ponašanja. Preostaje nam definirati operator selekcije te genetske operatore.

Za operator selekcije uzeta je eliminacijska (eng. steady state) k-turnirska selekcija. Eliminacijska k-turnirska selekcija uvijek operira nad jednom populacijom (nema međupopulacije kao kod jednostavne selekcije) tako što odabere k-jedniki (s jednakom vjerojatnošću) i turnirom odabere najlošiju koju izbaci. Postupak se ponovi neki određeni broj puta te se nakon toga populacija nadopuni parenjem preživjelih jedinki.

Odabrano je križanje sa jednom točkom prekida. Generira se dvoje djece tako da se ista točka prekida koristi za generiranje obaju potomaka.

Mutacija je nešto složenije definirana. Koriste se kako nezavisno tako i kombinacijama četiri tipa mutacija.

Miješajuća mutacija – odaberu se dva gena (jednakom vjerojatnošću) tj. u našem slučaju dvije instrukcije te im se zamijene mjesta. Na ovaj način generiramo permutacije instrukcija istog napada.

Jednostavna mutacija – odabere se jedna instrukcija (svi geni imaju istu vjerojatnost) te se zamijeni sa nekom drugom instrukcijom iz skupa odabranih petnaest instrukcija (opet su sve instrukcije jednako vjerojatne).

Potpuna mutacija – sve instrukcije unutar napada (do na neku vjerojatnost) zamijene se

nekom drugom instrukcijom iz skupa odabranih petnaest instrukcija (sve instrukcije su jednako vjerojatne).

Pohlepna mutacija – odabire se najbolji napad (obzirom na funkciju dobrote) te se zatim izmijeni jedna instrukcija koja najviše doprinese poboljšanju rezultata obzirom na funkciju dobrote. Kako je ova mutacija računski zahtjevna jako se rijetko provodi.

Vjerojatnost jednostavne i potpune mutacije opada linearno sa brojem generacija sa ciljem da se pri kraju rada genetskog algoritma (kada su rješenja "bolja") veća pozornost usmjeri na miješajuću mutaciju kako bi se ispitalo što više različitih permutacija napada.

Na kraju pogledajmo i parametre genetskog algoritma koje su autori koristili u testiranjima.

Parametar	Vrijednost
Veličina populacije	500
Vjerojatnost križanja	0.9
Vjerojatnost jednostavne mutacije	0.5 (linearno opada)
Vjerojatnost potpune mutacije	0.001 (linearno opada)
Vjerojatnost pohlepne mutacije	Svakih 1000 turnira
Vjerojatnost miješajuće mutacije	0.5
Veličina turnira	4
Kriterij zaustavljanja	100 000 turnira

slika: parametri genetskog algoritma korišteni prilikom testiranja

Rezultati i zaključak

Za konfiguraciju STIDE sustava za otkrivanje napada korištena su dva scenarija. Jedan nad osnovnim skupom podataka te drugi na širem skupu. U oba slučaja originalni napad rezultirao je sa mjerom odstupanja od normalnog rada oko 67%. Dakle, cilj je genetskog algoritma da producira napade sa znatno manjom mjerom odstupanja.

Osim različitih scenarija prilikom konfiguriranja STIDE-a korištena su i tri različita scenarija rada genetskog algoritma u ovisnosti o upotrebljenom operatoru mutacije. Scenariji su: korištenje jednostavne mutacije, zatim korištenje potpune mutacije te u zadnjem slučaju korištenje i potpuna i pohlepna mutacija. U sva tri scenarija korištena je i miješajuća mutacija.

U svim scenarijima najbolja postignuta mjera odstupanja bila je oko 5% što je u biti teško prepoznatljivo kao napad. Također, pokazano je da složenije mutacije rezultiraju manjom mjerom odstupanja jer omogućuju pretraživanje po širem skupu rješenja.

Cilj ovog primjera bio je pokazati iskoristivost genetskog algoritma kao efikasnog načina optimiziranja klase napada u odnosu na promatrani sustav za otkrivanje napada. Također, promatrani scenarij konstruiran je tako da što realnije modelira situaciju u kojoj bi se stvarni napadač nalazio, tj. situaciju u kojoj napadač ne posjeduje povjerljive informacije. Jedini podatak bitan napadaču je mjera odstupanja koju vraća sustav te koja je dostupna korisniku. Genetski algoritmi opet su se pokazali kao vrlo jednostavan, a iznimno efikasan mehanizam u pretraživanju skupa rješenja, kakav god on bio.

Zaključak

Sustavi za otkrivanje napada nametnuli su se kao neizostavni dio svakog ozbiljnog sigurnosnog sustava. Njihova uloga nije primarna zaštita (uglavnom) već prvenstveno "stražarska" tj. pružanje zaštite ako svi ostali sigurnosni mehanizmi zakažu. Vidjeli smo različite varijante sustava za otkrivanje napada, a kao najbitnije istaknuli smo mrežne i lokalne (obzirom na smještaj) te temeljene na otkrivanju nepravilnosti i temeljene na otkrivanju zlonamjernog korištenja (obzirom na pristup otkrivanju napada). Također smo obradili i standardne tehnike korištene za implementaciju sustava za otkrivanje napada. U moru takvih tehnika nas je posebno interesirala primjena genetskih algoritama. Prikazali smo primjere koji prate "sve faze" u razvoju sustava za otkrivanje napada, od pripreme podataka koje će sustav koristiti, preko samog otkrivanja napada putem genetskih algoritama pa sve do korištenja genetskih algoritama za testiranje sustava. U svakom od proučenih slučajeva genetski algoritmi ponovno su se pokazali kao efikasno sredstvo za pretragu skupa rješenja. Također, sve navedene primjere krasi jednostavnost, čitljivost i prvenstveno prirodnost koja prati genetske algoritme. Prostora za istraživanje ima te se nadam da će zainteresirani čitatelji shvatiti ovaj seminar kao odskočnu dasku ili motivaciju za dublje i temeljitije obrađivanje ove tematike.

Literatura

(a) Melanie Mitchell, *An introduction to genetic algorithms*, MIT Press, 1999

(b) Marin Golub, *Genetski algoritam prvi dio*, skripta, Zagreb, 2004

(c) Leo Budin, Marin Golub, *Operacijski sustavi 2*, skripta, Zagreb, 2007

(d) Ludovic Mé, *GASSATA, a Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis*, http://www.rennes.supelec.fr/rennes/si/equipe/lme/

(e) Sandeep Kumar, *Classification and detection of computer intrusions*, Ph.D. Thesis, 1995, http://www.cerias.purdue.edu/about/history/coast/coast-people.php

(f) Wei Li, *Using genetic algorithm for network intrusion detection*, http://www.security.cse.msstate.edu/docs/Publications/wli/

(g) Pedro A. Diaz-Gomez, Dean F. Hougen, *Improved off-line intrusion detection using a genetic algorithm*, 2005, http://www.cameron.edu/~pdiaz-go/

(h) Dorothy E. Denning, *An intrusion detection model*, http://www.cs.georgetown.edu/~denning/infosec/

(i) Ajith Abraham, Crina Grosan, Carlos Martin-Vide, *Evolutionary design of intrusion detection programs*, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.4087

(j) H. Gunes Kayacik, Malcom I. Heywood, A. Nur Zincir-Heywood, *Evolving buffer overflow attacks with detector feedback*, http://users.cs.dal.ca/~zincir/bildiri/EvoComnet07-gmn.pdf

(k) Aurobindo Sundaram, *An introduction to intrusion detection*, http://www.acm.org/crossroads/xrds2-4/intrus.html