Vizuelizacija upozorenja o otkrivanju mrežnih napada u formi grafa

Stefan Aleksić

Sadržaj – Mrežni napadi mogu imati ogroman negativni uticaj na infrastrukturu jednog informacionog sistema. Standardni pristup za detekciju napada jeste upravo ugradnja sistema za detekciju neautorizovanih pristupa mreži i generisanja upozorenja (*eng. Intrusion Detection System, IDS*). Međutim, ovi sistemi mogu da generišu ogroman broj poruka upozorenja (*eng. alert*) koje centar za bezbednost nije u mogućnosti da analizira, čak i uz pomoć njihove korelacije. Ovaj rad nastoji da kroz vizuelizaciju poruka upozorenja u formi grafa, dobijenog klasterizacijom poruka na osnovu sličnosti korisnički odabranih atributa, analizatoru pomogne pri procesu detektovanja potencijalnih napada na mrežni sistem. Korisnik pre pokretanja programa bi trebalo da navede putanje (URL ili File) do xml log fajlova generisanih od strane IDS-a, kao i mogućnost konfiguracije samog udela koji svaki od atributa poruke upozorenja ima pri računanju njihove sličnosti, čime se konfiguriše sama klasterizacija. Program, uz pomoć *Networkx* i *Matplotlib* biblioteka za programski jezik *Python*, iscrtava formirane grafove, dinamički menjajući njihovu strukturu na osnovu parsiranih podataka. Konfigurabilni *FIFO buffer* je iskorišćen kako bi se ograničio broj prikazanih upozorenja, dok se u pozadini generišu preseci u vidu slika trenutnog stanja grafa sa uparenim csv fajlovima u kojima su naznačeni detalji svake od prikazanih poruka. U radu je pored teorijske osnove iskorišćene za klasterizaciju, opisana i sama implementacija programa, koja može poslužiti kao korisničko uputstvo.

I. Uvod

Mrežna komunikacija je postala glavni način za razmenu informacija u mnogim sferama današnjeg društva, kako u lične svrhe, tako i za prenos važnih informacija unutar i između kompanija pri njihovom poslovanju. Iz ovog razloga, kompanije ulažu u razvijanje svojih informacionih sistema (*eng. Informational Technology, IT systems*), koji za cilj imaju između ostalog i održavanje bezbednosti na mreži preko koje se posluje.

Nažalost, kako se razvijaju metode za očuvanje bezbednosti, tako se na drugoj strani razvijaju metode koje istu narušavaju. Veoma notoran, sajber kriminal, može da podrazumeva i naizgled vrlo bezazlene aktivnosti, poput nerelevantne elektronske pošte, neprikladnih reklama, kao i onih koje zvuče previše dobro da bi bile istinite i sl. Sve od nabrojanog se koristi kao mamac za mnogo ozbiljnije zakonske prekršaje, poput na primer *ransomware*-a (*eng. ransom* – otkup, ucena) odnosno malicioznog softvera ili *malware*-a koji onemogućuje žrtvi da pristupi svom računaru, fajlovima, sistemu, ili mreži, dok ne ispuni ucenu sajber kriminalca. Krađa identiteta, *spoofing*, *phishing* i mnoge druge aktivnosti su sve češće na mreži, što je zastrašujuće s obzirom da je njihova meta prosečan korisnik Interneta.

Tema ovog rada su napadi mnogo ozbiljnije branše, napadi koji se planiraju i bivaju *deploy*-ovani na čitave organizacije, pa čak i vojne infrastrukture. Najpoznatiji napad ovog tipa je *DoS – Denial of Service*, koji podrazumeva da se mreža koja pruža usluge u vidu servisa preoptereti nevalidnim zahtevima, tj. beskorisnim saobraćajem na mreži, kako jednostavno od prevelikog broja zahteva ne bi imala procesorsko vreme, ili *bandwidth* da pruži usluge stvarnom korisniku. Još napredniji, *DDoS – Distributed Denial of Service* podrazumeva da se na distribuiran način, od strane većeg broja „*zaraženih*“ hostova sistem obori kako bi napadač mogao da se infiltrira i prikuplja osetljive informacije iznutra.

Kako bi se ovako kobni ishodi sprečili, mnoge organizacije plaćaju izvrsne programere da upravo obore sistem na najkreativniji način, u svrhu identifikacije slabih tačaka sistema i upotrebljenih obrazaca, koje dalje koriste kao izvor informacija kako bi napade koji bi se potencijalno odigrali u budućnosti prvo mogli da identifikuju, odnosno zaustave pre nego što postanu ozbiljna pretnja.

Cilj ovog rada je upravo da na osnovu prepoznatih napada, za koje je dobro poznato kako izgledaju, odnosno koje obrasce primenjuju i kakva upozorenja IDS sistem za njih generiše, prvo identifikuje odnosno odvoji od ostalog saobraćaja na mreži uz pomoć klasterizacije, a onda tako klasterizovanu grupu upozorenja vizuelno prikaže analizatoru koji održava bezbednost mreže. Na ovaj način analizator ne treba da prolazi kroz detaljne informacije svakog upozorenja koje biva generisano, već samo da definiše atribute, tj. njihov udeo pri računanju sličnosti, na osnovu kojih program u pozadini formira klastere. Tek kada uoči klaster srodnih upozorenja koji potencijalno mogu da predstavljaju jednu od faza napada, analizator može da dejstvuje na odgovarajući način kako bi se napad tu i zaustavio.

U narednom poglavlju će biti opisana teorijska osnova rada, dalje sama implementacija programa, kako bih u četvrtom poglavlju prikazao dobijene rezultate i na kraju izneo sopstvene zaključke vezane za temu.

II. Teoretska osnova

A. IDS mrežna upozorenja

Poruka mrežnog upozorenja (*eng. alert*) $a \in A$ može biti generisana od strane IDS sistema pokrenutog za mrežu ili konkretan mrežni uređaj i služi kao indikacija potencijalno malicioznih aktivnosti na mreži. Svako upozorenje se može posmatrati kao vektor atributa $a = (a_1, a_2, \dots, a_n)$, ovi atributi mogu biti IP adresa i broj porta uređaja koji je poslao zahtev, ili uređaja kome je zahtev upućen, vremenska markica kada je upozorenje generisano, protokol koji se koristi, tip upozorenja itd. Svaka poruka upozorenja takođe sadrži i jedinstveni identifikator (*eng. Unique Identifier – UID*) na osnovu koga se skladišti u bazi, odnosno log fajlu.

$$a = (uid, ts, src\_ip, src\_port, dst\_ip, dst\_port, proto, type)$$

B. IDS meta upozorenje

Napad, preciznije sva upozorenja koja su generisana za istu malicioznu akciju, odnosno jedna korak u napadu i, će za ishod imati skup svih istinski pozitivnih (*eng. true positive*) upozorenja S_i, pa će onda $\hat{S} = \{S_0, S_1, \dots, S_{n-1}\}$ predstavljati skup svih koraka u vidu skupova upozorenja za jedan napad.

Korelacijom je moguće redukovati ove skupove na apstraktna upozorenja, takozvana meta upozorenja, čime bi se smanjila količina redundantnih informacija, jer će jedno meta upozorenje referencirati ceo skup IDS upozorenja za jednu akciju. U ove svrhe se koristi funkcija korelacije, koja klasterizuje skup upozorenja A u skup klastera $\hat{C}$, gde se jedan klaster $Ci \in \hat{C}$ sastoji od svih upozorenja koja predstavljaju jedan korak u napadu.

Meta upozorenje se sastoji od sopstvenog identifikatora *UID-a*, vremenske markice u vidu vremena i datuma kada je kreiran, skupa svih identifikatora upozorenja koje apstrahuje, kao i skupa svih IP adresa žrtve i napadača koje se javljaju kroz upozorenja. Na kraju, meta upozorenju je pridružena i poruka koja opisuje napad. Ovim možemo da zaključimo da je IDS meta upozorenje zapravo skup svih upozorenja koje je generisao IDS kao odgovor na istu akciju.

$$m = (uid, ts, alert\_ids, attackers, victims, message)$$

Napad sačinjen od većeg broja koraka (*eng. multi-step attack*) možemo da predstavimo kao $M_j \subseteq \hat{S}$, pa skup svih takvih napada dalje obeležavamo sa $\widehat{M} = \{M_0, M_1, \dots, M_{m-1}\}$.

C. Klasterizacija upozorenja

Proces klasterizacije upozorenja predstavlja razdvajanje poruka upozorenja u odgovarajuće grupe upozorenja, odnosno u klastere $\hat{C} = \{C_0, C_1, \dots, C_{n-1}\}$, gde svaki klaster $Ci \in \hat{C}$ predstavlja napad i njemu pridružena upozorenja. Klasteri bi trebalo da modeluju stvarne napade $Si \in \hat{S}$, pa bismo u najboljem slučaju imali $\hat{S} = \hat{C}$. U ove svrhe se primenjuju dva zadatka: filtriranje upozorenja i izolacija napada.

Filtriranje upozorenja uklanja prividno tačna (*eng. false positive*) upozorenja, tako da u najboljem slučaju važi:

$$\forall a \in A: \quad \exists Si \in \widehat{S} \ \wedge \ a \in Si \iff \exists Ci \in \hat{C} \ \wedge \ a \in Ci$$

Definicija prividno tačnih upozorenja zavisi od konteksta, odnosno napada za koji se posmatra problem. Ovaj korak podrazumeva da se upozorenja dodele klasteru i samim tim dalje formiraju ulazne podatke za sledeći stepen analize ako i samo ako su prepoznata u skupu $\hat{S}$.

Izolovanje napada zahteva da se svako upozorenje iz skupa A dodeli jednom klasteru $Ci \in \hat{C}$. Skup korelisanih klastera $\hat{C}$ bi trebalo da isprati originalne korake u napadu $Si \subseteq A$, $Si \in \hat{S}$. Ovaj zadatak zavisi od toga da li za cilj imamo visoku homogenost ili heterogenost unutar klastera, odnosno izazov je pronaći korelaciju koja najbolje opisuje realnu situaciju, a optimalna je iz perspektive data mining-a.

D. Dodavanje konteksta

U ovom koraku, svakom klasteru $Ci \in \hat{C}$ se dodeljuje labela $li \in L$ koja sadrži dodatne informacije vezane za klaster, na primer opis klastera, opis tačaka koje su podložne napadu itd. Na ovaj način se olakšava analiza i kasnije uklapanje koraka u kompleksne napade.

E. Povezivanje napada

Kao poslednji korak u procesu korelacije upozorenja, povezivanje napada nastoji da pronađe relacije između klastera u skupu $\hat{C}$. Rezultat su $Ii \in \hat{I}$ koji reflektuju uklapanja iz skupa $\hat{S}$ u skup napada sa većim brojem koraka $\widehat{M}$. Primer ovoga je skeniranje nezaštićenih web server-a u okviru jedne podmreže, koje je dalje praćeno napadom na sam server, odnosno ugradnjom malicioznog softvera i eventualnom preuzimanju kontrole. Kako bi ova dva koraka povezali, koristi se kombinacija sekvencijalnih i kauzalno-zasnovanih mehanizama korelacije.

Ova faza je jako zavisna od samog opisa dodeljenog klasterima, jer se na osnovu toga najbolje mogu odrediti potencijalne veze između grupa upozorenja.

F. Transformacija upozorenja u strukturu grafa

Skup upozorenja A sa svojim atributima se može transformisati u težinski graf $G = (A, E)$ gde je za skup čvorova grafa iskorišćen skup A, a skupom E označeni svi potezi između čvorova, odnosno poruka upozorenja, u obliku (a_i, a_j) čija se težina potega dobija na osnovu vrednosti funkcije sličnosti između para upozorenja a_i i a_j:

$$s = F_{sim}(a_i, a_j), \in [0, 1]$$

Funkcija F_{sim} poredi svih n atributa $(a^0, a^1, \dots, a^{n-1})$ između dva upozorenja respektivno:

$$F_{sim}(a_i, a_j) \ = \ \sum_{k=0}^{n-1} c^k \cdot h^k \left(a_i^k, a_j^k\right)$$

Za svaki od atributa se primenjuje odgovarajuća funkcija poređenja $h^k \in [0, 1]$ čija vrednost biva skalirana odgovarajućim faktorom vektora $c = (c^0, c^1, \dots, c^{n-1})$ za koji važi $\sum c^j = 1$, odnosno funkcija sličnosti dva upozorenja zapravo predstavlja skalarni proizvod vektora c i h, gde vrednosti elemenata vektora h^k dobijamo na osnovu pridruženih funkcija za poređenje specifičnih atributa.

Na osnovu izračunate težine potega s se dalje odlučuje da li će odgovarajući poteg biti prisutan u grafu ili ne. Sličnost između dva čvora (poruka upozorenja) je potrebno da bude veća ili jednaka od minimalnog praga sličnosti τ, za poteg (a_i, a_j) da bi bio deo skupa potega E^{τ} odnosno grafa G^{τ}. Iz ovoga možemo zaključiti da τ kontroliše broj prikazanih potega $|E|$, time što uklanja potege između čvorova koji su najverovatnije nepovezani.

Težina potega zavisi isključivo od funkcije F_{sim} odnosno od vektora c i funkcija poređenja vektora h, dakle ovo će biti parametri konfiguracije kako bi se dobijali grafovi za različite vrste napada.

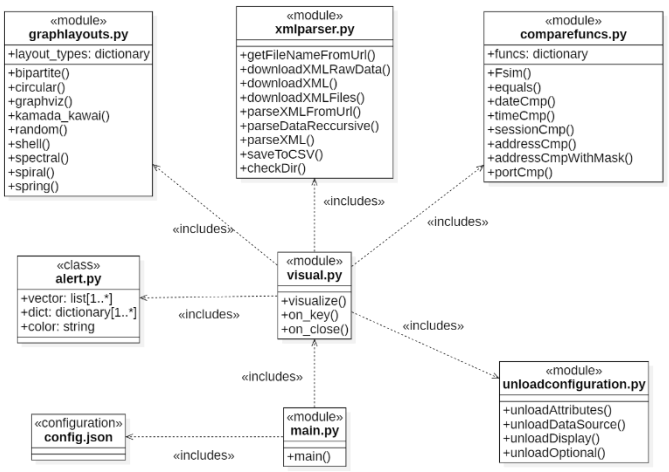
III. Implementacija

Program je u potpunosti napisan u programskom jeziku *Python* i oslanja se na postojanje biblioteka: *matplotlib*,

networkx, *requests*, kao i svih biblioteka koje su neophodne navedenim. Sve one se mogu instalirati uz pomoć *pip* (*Package Installer for Python*) komande:

$$pip\ install\ -r\ requirements.txt$$

Gde je poslednji parametar putanja do txt fajla u okviru kog se nalaze navedene biblioteke sa svojim verzijama.



Slika 1 - Arhitekturni model programa

Opis arhitekturnog modela:

config.json – *json* fajl u okviru kog se mogu podesiti mnogi parametri klasterizacije, prikaza i samog generisanja preseka stanja. To su parametri poput: putanja do izvornih fajlova, odnosno linkova ka istim, maksimalni broj prikazanih čvorova, algoritam koji se koristi za iscrtavanje, vrednost minimalnog praga sličnosti τ kojim se ograničava broj potega između čvorova i najbitnije, atributi koji se koriste pri parsiranju, odnosno pri upoređivanju poruka upozorenja a^k, sa svojim udelom pri računanju konačne težine potega između dva čvora c^ki funkcije koja će se koristiti za njihovo poređenje h^k.

alert.py – Klasa koja čuva atribute poruke upozorenja u formi vektora, odnosno rasute tablice, kao i boju kojom je prikazana na grafu.

graphlayouts.py – Moduo koji sadrži konfiguracije algoritama za iscrtavanje dostupnih biblioteka. Pokazivači na ove funkcije se eksportuju kroz promenljivu *layout_types* tipa *dictionary*, kako bi se olakšalo njihovo pozivanje.

xmlparser.py – Sadrži metode koji se koriste kako bi se učitali podaci neophodni za analizu. Implementirana je funkcija *parseDataReccursive(...)* koja vrši rekurzivno parsiranje podataka iz xml fajla koji je učitan sa sistema, ili skinut sa prosleđenog linka.

comparefuncs.py – eksportuje promenljivu *funcs* tipa *dictionary* koja skladišti sve pokazivače na funkcije koje se koriste za poređenje atributa, kao i same funkcije za poređenje upozorenja.

unloadconfiguration.py – moduo u kome su izdvojene funkcije koje vrše učitavanje konfiguracije i ujedno vrše i proveru iste, odnosno dodeljuju podrazumevane vrednosti ukoliko nisu prosleđene korektne vrednosti u okviru konfiguracionog fajla, obaveštavajući korisnika o tome.

visual.py – glavni modul koji koristi skoro sve do sada navedene module. Zadužen je za samu logiku iscrtavanja prosleđenih podataka kroz konfigurisan prikaz.

main.py – glavni program koji učitava konfiguracioni fajl i poziva *visualize(...)* metod u okviru modula visual.py.

Tok programa je sledeći. Nakon učitavanja konfiguracije, skidaju se log fajlovi sa navednih linkova i zajedno sa putanjama do već postojećih fajlova na sistemu bivaju sukcesivno parsirani. Parsiranje jednog fajla je atomično, odnosno podaci iz jednog fajla će se odjednom učitati u odgovarajuće promenljive i za svaku poruku koja bude bila pronađena, kreirati instanca klase *Alert*, sa odgovarajućim atributima.

Nakon što biva kreiran, objekat tipa *Alert* se upoređuje sa svim prethodno generisanim objektima uz pomoć funkcije $F_{sim}(...)$ u okviru modula *comparefuncs*. Definicija ove funkcije je data u okviru poglavlja teorijske osnove. U pozadini se za svaki par atributa iz vektora atributa u okviru upozorenja pozivaju odgovarajuće funkcije, kojima se porede vrednosti atributa, a procena njihove sličnosti biva skalirana odgovarajućom komponentom vektora c. Svi ovi parametri se mogu konfigurisati u okviru konfiguracionog fajla i njihovom promenom će i klasteri koji se formiraju menjati svoju strukturu.

Težina potega između dva upozorenja koja se upoređuju, dobijena funkcijom $F_{sim}(...)$, se dalje upoređuje sa parametrom τ, kako bi se, ukoliko je veća, zaključilo da su upozorenja dovoljno slična i time formirao poteg između njih. Takođe se kroz poređenje svih prethodno instanciranih objekata koristi prilika da se nađe onaj sa kojim je trenutno obrađivani *Alert* najsličniji, kako bi vizuelno dobili istu boju i eventualno predstavljali jedan klaster.

Broj trenutno prikazanih atributa je ograničen i nakon što broj do sada generisanih čvorova dostigne tu vrednost, čvorovi bivaju prikazani po FIFO (First-In-First-Out) principu, odnosno oni koji su najduže bili deo grafa bivaju izbačeni kako bi se prikazali novi.

Na svaki umnožak broja maksimalno prikazanih čvorova se formira presek, koji u zavisnosti od konfiguracije, podrazumeva generisanje slike trenutnog stanja grafa, koja je kroz vremensku markicu u trenutku generisanja, uparena sa podacima o prikazanim upozorenjima, u vidu csv fajla. Ovim je postignuto da analizator nakon uočavanja klastera upozorenja, može da detaljnije pogleda i informacije o svakom od njih, a onda ustanovi da li se zaista radi o traženom napadu.

Za iscrtavanje grafa se mogu koristiti različiti algoritmi, koji naravno imaju drugačiji prikaz grafa. Podrazumevani, u okviru konfiguracionog fajla je *spring_layout*, koji daje zadovoljavajuće rezultate. Međutim, ukoliko želite bolji prikaz, predlažem odabir *graphviz* algoritma, za koji je neophodno ispratiti instrukcije [1] za instalaciju *pygraphviz*, odnosno *graphviz* biblioteka. Želeo bih da napomenem da je ovo malo složeniji proces i iz tog razloga je ovaj algoritam uključen kao opcioni, odnosno program je sposoban da radi sa njim, ali je na korisniku da instalira neophodne alate.

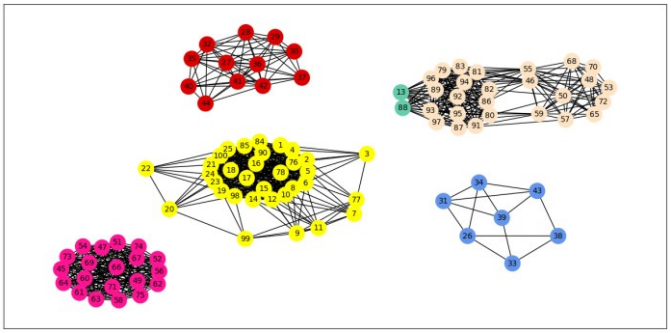
IV. PRIKAZ DOBIJENIH PODATAKA

Za testiranje programa je iskorišćen set podataka (*eng. data set*) generisan od strane agencije za istraživanje i razvoj emergentnih tehnologija za održavanje bezbednosti u vojne svrhe DARPA (Defense Advanced Research Projects Agency). Ovo su stvarni podaci koje je generisao IDS sistem za upozorenje jednog DDoS napada. Napad se sastoji iz dva scenarija, a svaki scenario od po 5 faza.

U prvom scenariju, napadač se još uvek smatra poletarcem, pa je samim tim za njegov napad IDS generisao mnogo više poruka upozorenja. U drugom scenariju ovo nije slučaj, napadač je iskusniji i sa minimalnim brojem poruka upozorenja je uspeo da dođe do svog cilja.

Faze jednog scenarija podrazumevaju:

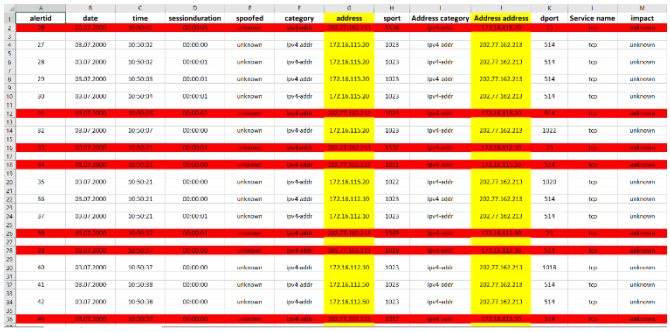
1. *IPsweep* – prikupljanje IP adresa u okviru vazduhoplovne baze koje se nalaze u okviru mreže klase C slanjem ICMP (*Internet Control Message Protocol*) *echo* zahteva i osluškivanja za *echo* odgovore.
2. Pretraga aktivnih IP adresa za *sadmind* demonom koji se izvršava na uređaju na kom je pokrenut Solaris.
3. Upad na sistem, uz pomoć slabosti uređaja na mreži da se odbrane od *sadmind* crva.
4. Instalacija *trojan mstream* DDoS softvera na barem tri mrežna uređaja u okviru mreže vazduhoplovne baze.
5. Pokretanje bujice zahteva koji prouzrokuju opterećenje sistema.



Slika 2 – Stanje grafa

Na slici 2 možemo videti proizvoljno stanje grafa za set podataka iz prvog scenarija između faze 2 i 4. Na grafu, koji je konfigurisan tako da klasterizuje upozorenja prvobitno na osnovu vrednosti atributa njihovih izvornih, odnosno dolaznih IP adresa, sa vrednošću $\tau = 0.7$, i veličinom FIFO buffer-a podešenom na 100, jasno možemo uočiti formirane klastere roze, crvene, žute, plave i taup boje. Ukoliko nam je iz nekog razloga, klaster plave boje sumnjiv, možemo otvoriti csv fajl koji opisuje dati graf.

Unutar csv fajla, slika 3, možemo lako pronaći čvorove sa grafa na osnovu njihovih identifikatora, a dalje uočiti zašto je ovih 7 čvorova klasterizovano. Kao što je i konfigurisano, izvorne IP adrese su identične, dok adrese odredišta se nalaze u jako približnom opsegu, pa samim tim je težina potega između ovih čvorova mnogo viša u odnosu na ostatak grafa, a definitivno viša od minimalnog praga sličnosti 0.7.



alertid	date	time	sessionduration	spoofed	category	address	sport	Address category	Address address	dport	Service name	impact
26	03.07.2000	10:50:01	00:00:00	unknown	ipv4-addr	202.77.162.213	1026	ipv4-addr	172.16.115.50	23	tcp	unknown
27	03.07.2000	10:50:02	00:00:00	unknown	ipv4-addr	172.16.115.20	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
28	03.07.2000	10:50:02	00:00:01	unknown	ipv4-addr	172.16.115.20	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
29	03.07.2000	10:50:03	00:00:01	unknown	ipv4-addr	172.16.115.20	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
30	03.07.2000	10:50:03	00:00:01	unknown	ipv4-addr	172.16.115.20	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
31	03.07.2000	10:50:05	00:00:01	unknown	ipv4-addr	202.77.162.213	1025	ipv4-addr	172.16.115.50	514	tcp	unknown
32	03.07.2000	10:50:07	00:00:00	unknown	ipv4-addr	172.16.115.20	1025	ipv4-addr	202.77.162.213	1022	tcp	unknown
33	03.07.2000	10:50:07	00:00:01	unknown	ipv4-addr	202.77.162.213	1027	ipv4-addr	172.16.115.50	23	tcp	unknown
34	03.07.2000	10:50:07	00:00:00	unknown	ipv4-addr	202.77.162.213	1025	ipv4-addr	172.16.115.20	514	tcp	unknown
35	03.07.2000	10:50:21	00:00:00	unknown	ipv4-addr	172.16.115.20	1022	ipv4-addr	202.77.162.213	1023	tcp	unknown
36	03.07.2000	10:50:21	00:00:00	unknown	ipv4-addr	172.16.112.30	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
37	03.07.2000	10:50:21	00:00:01	unknown	ipv4-addr	172.16.112.30	1023	ipv4-addr	202.77.162.213	514	tcp	unknown
38	03.07.2000	10:50:37	00:00:01	unknown	ipv4-addr	202.77.162.213	1025	ipv4-addr	172.16.112.50	23	tcp	unknown
39	03.07.2000	10:50:37	00:00:00	unknown	ipv4-addr	202.77.162.213	1029	ipv4-addr	172.16.112.50	514	tcp	unknown
40	03.07.2000	10:50:37	00:00:00	unknown	ipv4-addr	172.16.112.30	1025	ipv4-addr	202.77.162.213	1018	tcp	unknown
41	03.07.2000	10:50:38	00:00:00	unknown	ipv4-addr	172.16.112.50	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
42	03.07.2000	10:50:38	00:00:00	unknown	ipv4-addr	172.16.112.50	1025	ipv4-addr	202.77.162.213	514	tcp	unknown
43	03.07.2000	10:50:39	00:00:00	unknown	ipv4-addr	202.77.162.213	1031	ipv4-addr	172.16.112.50	514	tcp	unknown

Slika 3 – Detalji prikazanih čvorova

Za ostale klastere, koji imaju mnogo veći broj čvorova je ista situacija. Dakle definitivno su upozorenja povezana, što je i očekivano, s obzirom da program prikazuje podatke za jedan već odigran DDoS napad.

V. ZAKLJUČAK

U ovom radu je implementiran program za vizuelnu reprezentaciju poruka upozorenja koje generišu IDS sistemi u formi grafa. Program je testiran sa dostupnim setom podataka, za koji je, uz odgovarajuću konfiguraciju uspeo da klasterizuje poruke upozorenja i time vizuelno identifikuje potencijalni korak u napadu na sistem. U radu je objašnjena teorijska osnova na kojoj je bazirana implementacija, kao i opis same arhitekture programa, čime su date i smernice šta i kako je moguće konfigurisati da bi se dobili željeni rezultati.

ZAHVALNICA

Želeo bih da se zahvalim profesoru doktoru Vladimiru Ćiriću, kao i diplomiranom inženjeru Nađi Gavrilović, pre svega što su mi pružili priliku pisanja ovog rada, kao i za samo mentorstvo pri njegovoj izradi.

LITERATURA

[1] "Pygraphviz," [Online]. Available: https://pygraphviz.github.io/documentation/stable/install.html.

[2] M. F. Steffen Haass, Security Monitoring and Alert Correlation for Network Intrusion Detection, Mainz, 2020.

[3] "Denial-of-service-attack," [Online]. Available: https://en.wikipedia.org/wiki/Denial-of-service_attack.

[4] "Intrusion detection system," [Online]. Available: https://en.wikipedia.org/wiki/Intrusion_detection_system.

[5] "Matplotlib official documentation," [Online]. Available: https://matplotlib.org/.

[6] "NetworkX official documentation," [Online]. Available: https://networkx.org/.

[7] "Visualize Graphs in Python," [Online]. Available: https://www.geeksforgeeks.org/visualize-graphs-in-python/.

[8] DARPA, "DARPA Intrusion Detection Evaluation - Lincoln Laboratory Scenario (DD0S) 2.0.2," 2000. [Online]. Available: https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_2.0.2.html.

[9] DARPA, "DARPA Intrusion Detection Evaluation - Lincoln Laboratory Scenario (DDoS) 1.0," 2000. [Online]. Available: https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html.