UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET

Boban Banjević

Analiza softverskih rješenja za detekciju i prevenciju bezbjedonosnih prijetnji na aplikativnom nivou

MASTER RAD

Podgorica, 2023. godine

PODACI I INFORMACIJE O MAGISTRANDU

Ime i prezime:	Boban Banjević
Datum i mjesto rođenja:	23.02.1999. godine, Podgorica, Crna Gora
Naziv završenog osnovnog studijskog programa i godina završetka studija:	Studijski program Primijenjeno računarstvo, Elektrotehnički fakultet, 2020. godine.

INFORMACIJE O MASTER RADU

Naziv postdiplomskih studija:	Postdiplomske master studije primijenjenog računarstva
Naslov rada:	Analiza softverskih rješenja za detekciju i prevenciju bezbjedonosnih prijetnji na aplikativnom nivou
Fakultet na kojem je rad odbranjen:	Elektrotehnički fakultet

UDK, OCJENA I ODBRANA MASTER RADA

Datum prijave master rada:	06.04.2022. godine
Datum sjednice Vijeća na kojoj je prihvaćena tema:	15.09.2022. godine
Komisija za ocjenu teme i podobnosti magistranda:	Prof. dr Slobodan Đukanović, predsjednik Prof. dr Nikola Žarić, mentor Prof. dr Miloš Brajović, član
Mentor:	Prof. dr Nikola Žarić, ETF Podgorica
Komisija za ocjenu rada:	Prof. dr Slobodan Đukanović, predsjednik Prof. dr Nikola Žarić, mentor Prof. dr Miloš Brajović, član
Komisija za odbranu rada:	Prof. dr Slobodan Đukanović, predsjednik Prof. dr Nikola Žarić, mentor Prof. dr Miloš Brajović, član
Datum odbrane:	27.12.2023. godine

Ime i prezime autora: Boban Banjević, BApp

ETIČKA IZJAVA

U skladu sa članom 22 Zakona o akademskom integritetu i članom 18 Pravila studiranja na master studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je master rad pod naslovom

"Analiza softverskih rješenja za detekciju i prevenciju bezbjedonosnih prijetnji na aplikativnom nivou"

moje originalno djelo.

Podnosilac izjave,

Boban Banjević

Boban Banjević, BApp

U Podgorici, dana 15.11.2023. godine

PREDGOVOR

Ovaj rad posvećujem mojim roditeljima koji su me školovali, mojoj tetki, bratu i sestrama i svima ostalima koji su mi pružali podršku i pomoć prilikom moga sveukupnog školovanja. Posebnu zahvalnost želim da iskažem mom mentoru Prof. Dr Nikoli Žariću, koji mi je pomogao prilikom izbora teme, koji mi je davao kvalitetne savjete i usmjeravao prilikom vršenja istraživanja i pisanja rada. Veliku zahvalnost želim takođe da izrazim svim svojim kolegama – prijateljima iz institucije u kojoj radim, Erste banka Crne Gore, koji su mi dali inicijalnu inspiraciju i neophodne informacije za realizaciju ovog projekta.

Boban Banjević, BApp

IZVOD RADA

Sve veći rast i razvoj informacionih tehnologija i njihova primjena u raznim poslovnim i privatnim sferama života, ali i integracija istih sa povjerljivim i osjetljivim podacima, predstavlja ogromnu metu podležnu raznim vrstama napada od strane agresora koji imaju za cilj da ukradu ili oštete te podatke. Sa razvojem velikog broja bezbjedonosnih prijetnji mora da raste i broj ideja koje bi doprinijele njihovoj detekciji i sprečavanju. Sedmi nivo OSI modela, odnosno aplikativni nivo, predstavlja prvu udarnu tačku jednog sistema zbog njegove direktne izloženosti krajnjim korisnicima.

U teorijskom dijelu se prolazi kroz bezbjedonosne prijetnje, klasifikaciju i taksonomiju bezbjedonosnih prijetnji radi njihovog boljeg razumijevanja i prepoznavanja, ali i neke od najčešćih praksi pripreme i prijevremene prevencije, kao i oporavka od istih. Zatim se govori o gotovim softverskim rješenjima i njihovoj primjeni, a na kraju se u praktičnom dijelu rada primjenjuju mehanizmi detekcije i odbrane o kojima se pričalo u teorijskom okviru i na kreativan način, uz kombinaciju softverskih rješenja i skripti detektuje i sprečava jedna vrsta DDoS napada.

Istraživanjem je dokazano da korišćenjem raznih softverskih rješenja, pretežno "open-source" tipa, u kombinaciji sa skriptama i ugrađenim alatima koji dolaze uz sistem, a pogotovo uz neophodno planiranje arhitekture sistema, moguće je odbraniti se od raznih bezbjedonosnih prijetnji na aplikativnom nivou.

Izvedeno istraživanje prikazuje upotrebu "open-source" softverskih rješenja koje male i srednje kompanije mogu da koriste da sačuvaju podatke svojih korisnika. Ova rješenja mogu da omoguće kompanijama da umanje troškove, a da pri tom ostanu konkurentna na tržištu. Kao doprinos u oblasti istraživanja, ovaj rad može omogućiti osnov za dalje istaživanje u polju zaštite aplikacija i konkretno podataka korisnika.

Ključne riječi: open-source, aplikativni nivo, reverse proxy, centralizacija, kompetitivnost

ABSTRACT

The increasing growth and development of information technologies and their application in various business and private spheres of life, as well as their integration with confidential and sensitive data, represents a huge target subject to various types of attacks by aggressors who aim to steal or damage this data. With the development of a large number of security threats, the number of ideas that would contribute to their detection and prevention must also grow. The seventh level of the OSI model, i.e. the application level, represents the first impact point of a system due to its direct exposure to end users.

In the theoretical part, we go through security threats, the classification and taxonomy of security threats for their better understanding and recognition, but also some of the most common practices of preparation and early prevention, as well as recovery from the same. Next we discuss ready-made software solutions and their application, and following in the practical part of the work, the detection and defense mechanisms discussed in the theoretical framework are applied in a creative way, with the combination of software solutions and scripts, to detect and prevent one type of DDoS attack.

Research has proven that by using various software solutions, mostly "open source", in combination with scripts and built-in tools that come with the system, and especially with the necessary planning of the system architecture, it is possible to defend against various security threats at the application level.

The conducted research shows the use of "open source" software solutions that small and medium-sized companies can use to save their users' data. These solutions can enable companies to reduce costs while remaining competitive in the market. As a contribution to the field of research, this paper can provide a basis for further research in the field of application protection, or more specifically user data.

Key words: open-source, application level, reverse proxy, centralization, competitivity

Sadržaj

1. Uvod

Bezbjednost podataka i informacija je jedan od ključnih izazova današnjice. Detekcija i prevencija upada na aplikativnom nivou teži ka tome da spriječi pokušaje krađe bitnih podataka ili kodova od strane napadača ili grupe napadača. Da bi se olakšao proces i da bi se detektovale različite vrste upada, koriste se posebna softverska rješenja za detekciju upada i ista ili različita rješenja za njihovu prevenciju.

Zaštita na nivou aplikacije se može odnositi na zaštitu aplikacija na računaru, aplikacija na pretraživaču, mobilnih aplikacija ili aplikacija na cloud-u. Napadač cilja operativni sistem ili aplikaciju i traži njihove ranjivosti, koje mu omogućavaju da zaobiđe kontrolu pristupa i pristupi podacima u okviru željene aplikacije. Zato se koriste razni softverski alati za zaštitu koji imaju glavne funkcije testiranja i/ili uklanjanja prijetnji. Dosta rješenja se specijalizuje za jednu od ove dvije aktivnosti, ali postoje i one koje se specijalizuju za obije funkcije.

Predmet istraživanja ovog rada je fokusiran na bezbjednost podataka, odnosno samu bezbjednost na aplikativnom nivou. Istraživanjem su obuhvaćeni postojeći napadi, rekreiranje određene vrste DDoS napada na aplikativnom nivou, analiza alata za zaštitu i upoređivanje različitih "Open source" (otvoreni kod) alata.

Imajući u vidu da su se tokom godina, prilikom razvijanja novih tehnologija, razvijale razne vrste prijetnji i upada koji mogu da nanesu štetu ili ukradu veliki broj privatnih podataka, predmet istraživanja je najprije izučavanje postojećih vrsta upada, načina kako funkcionišu, koji segment aplikacije ciljaju, koju slabost traže i slično. Jedan dio je posvećen i prikazu kako zaštita funkcioniše u pravom poslovnom okruženju. Sledeći korak u istraživanju je izučavanje kako koji alat funkcioniše i koje specifikacije sadrži, odnosno šta nudi u pogledu zaštite. Testiranje performansi rješenja za zaštitu aplikacije, podrazumijeva rekreiranje određene vrste napada, te se i ovi mehanizmi moraju izučiti, kako bi se pravilno koristili.

Sa razvojem tehnologija, kao i sve češćim procesom digitalizacije podataka od strane pojedinaca, kompanija, sve su češći upadi u cilju krađe osjetljivih i povjerljivih podataka ili kodova. Takođe, u slučaju da se nađu slabosti koje upadač može iskoristiti, treba ih ispraviti, odnosno zakrpiti. Cilj je da se podigne nivo znanja o upadima i mogućnosti gubitka, odnosno krađe

povjerljivih podataka kompanija, ili njenih korisnika. Dosta alata koji već postoje i koji su namijenjeni za detekciju, zaštitu ili oboje istovremeno, mogu da se iskoriste kako bi se olakšao sami proces zaštite.

Cilj ovog rada je i upoznavanje sa postojećim alatima koji mogu da doprinesu detekciji i prevenciji, kao i poređenje komercijalnih alata u odnosu na "open source" alate, odnosno da se izvrši analiza ponašanja softvera za zaštitu u određenim situacijama rekreiranih upada. Ali pored alata koji služe za zaštitu, da bi se bolje razumjelo šta ti alati rade, odnosno od čega ti alati tačno štite, ili šta tačno detektuju, moraju se upoznati i sredstva, tj. napadi koje upadači koriste da bi zaobišli nivo postojeće protekcije i pristupili privatnim podacima korisnika.

Shodno tome jedan od ciljeva je da se rekreira jedna vrsta bezbjedonosne prijetnje kako bi se upoznali sa vrstama upada, što ti upadi tačno ciljaju, koje slabosti traže i iskorišćavaju, način na koji dolaze do hardvera ili aplikacije korisnika i nivo opasnosti koje isti predstavljaju. Jedan od primjera upada gdje su česti problemi u zaštiti su *web* sajtovi. Veliki udio u tome problemu imaju interakcije između velikog broja komponenti dinamičkih *web* sajtova, koje čine zaštitu na nivou aplikacija veoma teškim izazovom. To je jedan od primarnih razloga zašto je korišćenje raznih posebnih alata za zaštitu na aplikativnom nivou velika neophodnost.

Ovakvi primjeri stvaraju još jedan veoma bitan cilj, a to je upoznavanje programera sa greškama u aplikacijama koje prave, ili ukazivanje na moguće probleme prilikom pisanja koda, na koje moraju da obrate pažnju. Iako u odgovornim kompanijama svaka aplikacija prolazi kroz dodatne provjere koda i ostale propratne prakse koje te kompanije imaju, dosta aplikacija je napravljeno od strane "solo-developera" ili “free lancer-a“, koje na kraju koriste krajnji korisnici. U takve aplikacije korisnici unose privatne podatke, adrese, povezuju bankovne račune i vrše određene transakcije, koje mogu da dovedu do "data leaks" (curenja podataka). Zbog toga je ovaj rad jedan vid pokušaja podizanja svijesti programera koji razvijaju aplikacije, ali i korisnika koji će na kraju da ih koriste.

Hipoteze koje su dokazane u ovom master radu su:

1. Besplatni "open source" softveri za detekciju i prevenciju na aplikativnom nivou su jednako ili približno efikasni u detektovanju i uklanjanju potencijalnih opasnosti kao i profesionalni softveri.

2. Korišćenje softvera za detekciju i prevenciju bezbjedonosnih prijetnji smanjuje troškove ulaganja u opremu i ljudstvo i omogućava kompetitivnost malih preduzeća uz sigurnost po realnim i pristupačnim cijenama.

Upoređivanjem trenutnih tržišnih cijena kompanija koje se bave zaštitom sa cijenom softvera za zaštitu i pojedinačnih plata ljudi koji se bave zaštitom, pokušano je dokazati da i male kompanije mogu da opstanu na tržištu, bez potrebe za velikim timom stručnjaka. Odnosno nema potrebe za kreiranjem novih privatnih softverskih alata ili korišćenja alata drugih kompanija koje se bave zaštitom, a koje naplaćuju velike cijene koriščenja tih alata kao i lica koja su osposobljena za upravljanjem istim, kada postoje već gotova rješenja koje pojedinci ili manji timovi mogu da koriste da osiguraju kompaniju i njene podatke po mnogo pristupačnijoj cijeni. Takođe, dokazivanjem efikasnosti programa koji se koriste za praktični dio rada, dodatno može doprinijeti dokazivanju konkurentnosti manjih kompanija, naravno uz dodatni trošak koji kompanije snose za osposobljavanje lica koja bi se bavila zaštitom, njihovo samo osposobljavanje, kao i vrijeme utrošeno za taj proces obuke.

Ovo istraživanje bi kao rezultat trebalo da pokaže performanse primjene softvera za detekciju i prevenciju upada u određenim rekreiranim situacijama upada, kao i poređenje između komercijalnih i nekomercijalnih softvera u pogledu kvaliteta i uspješnosti obavljanja predvidjenih zadataka. Očekivano je da se ovim istraživanjem pokaže uticaj upotrebe softvera za detekciju i prevenciju upada na poboljšanje produktivnosti i kvaliteta rada u odnosu na period kada se takvi alati nisu koristili, ali i da se pokaže razlika u kvalitetu obavljanja zadataka u odnosu na nekomercijalne softvere istoga tipa.

Primjena softvera za detekciju i prevenciju bi trebala da pomogne u borbi protiv "cyber" kriminala, odnosno da se spriječe razne potencijalne vrste upada i mogućnosti krađe bitnih privatnih podataka neke kompanije ili pojedinaca. Takođe, dodatna pomoć je i u tome što male kompanije koje budu koristile ove vrste softvera, mogu da osiguraju svoju imovinu, svoje račune, svoje korisnike i još mnogo toga, a da opet smanje potencijalne troškove, jer plaćanje posebnih kompanija koje se bave zaštitom podataka, a mogu biti veoma skupe, je finansijski neodrživo. Ovaj rad bi mogao da posluži onima koji žele da se bave zaštitom na aplikativnom nivou, ili uopšteno cyber zaštitom.

U drugom poglavlju ćemo govoriti šta je to zaštita na aplikativnom nivou, zašto je bitan pojam zaštite. Proći ćemo kroz sve nivoe OSI modela i upoznati se kratko sa ostalim nivoima. Potom ćemo sagledati bezbjedonosne prijetnje na aplikativnom nivou i kratko se upoznati sa njima, kao i sa načinima odbrane koji mogu da se primijene na više različitih napada.

U narednom poglavlju ćemo se detaljnije upoznati sa bezbjedonosnim prijetnjama, njihovom klasifikacijom i klasifikacionim modelima poput trodimenzionalnog, piramidnog, hibridnog i drugih, u cilju lakše detekcije i zaštite u budućnosti. Takođe ćemo proći kroz veoma bitan pojam taksonomije, koji je još jedan vid podjele bezbjedonosnih prijetnji. Nakon svega toga, proći ćemo i kroz neke od bezbjedonosnih prijetnji na aplikativnom nivou, kao što su DDoS napadi i njihova podjela, potom SQL Injection, Cross-Site Scripting i na kraju Cross-Site Request Forgery.

Četvrto poglavlje govori o detekciji bezbjedonosnih prijetnji na aplikativnom nivou i za svaku od prijetnji pomenutih u prethodnom poglavlju daje načine kako mogu da se identifikuju i postavlja podlogu za naredno poglavlje koje objašnjava kako da se izvrši prevencija ovih prijetnji.

Poglavlje pet prolazi kroz sami pojam zaštite od bezbjedonosnih prijetnji. U istom se opisuju četri načina prevencije detektovanih bezbjedonosnih prijetnji, od korišćenja raznih firewall logika koji mogu da filtriraju prijetnje, zatim korišćenje Security *Web* Gateway-a (Zaštitni mrežni prolaz) koji ima mogućnost višestruke validacije prijetnji, korisćenje Runtime application self-protection-a (Samostalna zaštita rada aplikacije) ili RASP-a koji je integrisan u aplikaciji ili naprednog firewall-a. U poglavlju se zatim govori o preventivnoj zaštiti vršenjem penetracionog testiranja u cilju saznanja slabosti koje aplikacija ima. Poglavlje se završava sa pojavom softverskih rješenja kao olakšanja u zaštiti od bezbjedonosnih prijetnji.

U poglavlju šest se izvršava jedan od napada na aplikativnom nivou (Slowloris DDoS), a kao odgovor na opasnost, napad se detektuje i sprečava dodavanjem IP adrese računara na crnu listu i trajnom zabranom pristupa. U šestom poglavlju se koristi veliki dio informacija o kojima smo pisali u prethodnim poglavljima, da bi se krajnji projekat doveo do kraja.

Poglavlja sedam i osam sadrže diskusiju i analizu, kao i zaključak, gdje se sumira cijeli rad, kao i utisci koje smo stekli kroz istraživanja i pisanje istog.

2. Zaštita na aplikativnom nivou

Zaštita podataka bilo koje vrste aplikacija, nebitno da li se radi o malim ili velikim, korporacijskim ili ličnim, internim ili eksternim, bankovnim ili nekim drugim aplikacijama, sajtovima, cloud-u (oblaku) ili bila koja druga vrsta kreacije koja zahtijeva da korisnik ili vlasnik unese svoje podatke, svoju šifru ili svoju sliku, a naravno da je povezana na *web* (mrežu), mora da ima neki vid zaštite. Svako voli da posjeduje neki vid sigurnosti ili zaštite, pogotovo na Internetu, gdje je teško pronaći počinioce onih djela koje ugrožavaju privatnost i bezbjednost korisnika. Interni programi kompanija često posjeduju polise i privatne podatke klijenata, podatke o trenutnom i budućem toku novca i dosta sličnih podataka koje kompanije možda žele da zadrže za sebe. Jedne od najčešćih meta pokušaja pristupa podacima, jesu bankovne aplikacije, odnosno mobilne aplikacije banki preko kojih korisnici pristupaju uvidu o svojim računima, transakcijama i slično. Prilikom pristupa sajtovima koji nisu zaštićeni, odnosno koji ne posjeduje "Secure Sockets Layer" (Zaštitni sloj priključaka) SSL internet protokol, ili bilo koji drugi, korisnici unose svoje podatke sa kartice ili bankovne podatke, koje omogućavaju počiniocima koji kradu podatke da istim i pristupe. Jedan od veoma čestih razloga može biti slaba šifra korisnika, ponavljanje šifre, ne-ažuriranje šifre i ne zaključavanje računara sa podignutim sistemom ili sistemima [1]. Broj takvih napada na bankovne račune korisnika se udvostručio u odnosu na 2021. godinu skoro duplo, odnosno na skoro dvadeset miliona upada [2]. Rast sistema i mreže se zasniva na anonimnosti, što propratno znači i poteškoće prilikom praćenja napadača, a sama kompleksnost sistema i broj protokola i aplikacija čini potpuno uklanjanje ranjivosti, skoro pa nemogućim. Napade najčešće trpe banke, aplikacije za odbrambenu industriju, procesi konsolidacije kreditnih kartica, zdravstvena industrija, distributeri sadržaja i slično [3]. Zaštita podataka označava da osoba, ili tim, koji su kreirali aplikaciju, osiguraju istu od potencijalnih upada, odnosno od potencijalnih bezbjedonosnih prijetnji, koji mogu da ugroze i izlože privatne podatke korisnika. Međutim, oblast zaštite od rizika je dostigla toliki nivo da čak ni profesionalci ne mogu da ostanu u koraku kako sa postojećim rizicima, tako ni sa postojećim načinima zaštite od istih [4]. Postavlja se pitanje da li su ranjivosti gusto zbijene ili rasprostranjene? Ako su rasprostranjene, to onda znači da rješavanje jedne ranjivosti nas dovodi na korak od rješavanja svih ranjivosti, ili su naprotiv gusto zbijene, što znači da je rješavanje jedne ranjivosti neprimjetno u širokom prostranstvu drugih. Ovakav problem

nazivamo zagonetkom, da li je isplativo ulagati velika sredstva u rješavanje ranjivosti ili je njihovo rješavanje neiscrpljivi bunar novih problema [6].

2.1 Koji sve nivoi zaštite postoje?

Sa obzirom da je tema master rada sami aplikativni nivo, nećemo puno govoriti o drugim nivoima, ali da bi razumjeli samu važnost zaštite na aplikativnom nivou, smatramo da je neophodno da budemo upoznati, makar na kratko i sa drugim nivoima koji postoje. Postoji ukupno sedam nivoa, prema "The Open Systems Interconnection" (Međusobna povezanost otvorenih sistema) ili OSI modelu, što se može vidjeti na slici 1. OSI model je takođe prihvaćen od strane "The International Organization for Standardization" (Internacionalna organizacija za standardizaciju) ili ISO, kao internacionalni standard. Iako današnji Internet nije baziran na OSI modelu, već TCP/IP modelu i dalje se koristi OSI model. Sledeća podjela gleda tako da je broj jedan najviši nivo, pa potom ide ka manjem.

Sedam nivoa OSI modela [5]:

1. Application Layer (Aplikativni Nivo) – Aplikativni nivo je nivo koji se koristi od strane programa namijenjen za krajnje korisnike. Sadrži protokole koji omogućavaju komunikaciju, odnosno primanje i slanje podataka i informacija, kao i njihovo predstavljanje krajnjim korisnicima;

2. Presentation Layer (Prezentacioni Nivo) - Prezentacioni nivo priprema podatke za aplikativni nivo, a takođe podatke od aplikativnog nivoa priprema za nivo sesije. Ovaj nivo takođe služi za definiciju kako podaci treba da budu kriptovani i kompresovani;

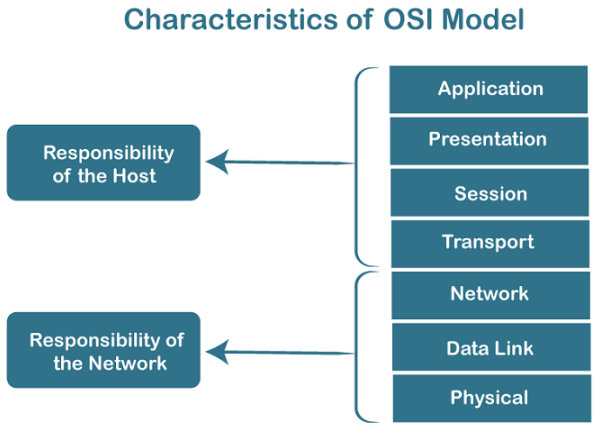
3. Session Layer (Nivo Sesije) – Nivo sesije je zadužen za otvaranje, odnosno kreiranje komunikacionih kanala između uređaja, zvanih "sesije". Glavni zadatak koji ovaj nivo ima, pored kreiranja sesija, je da osigura da sesija ostane otvorena tokom komunikacije i da je na samom kraju zatvori;

4. Transport Layer (Nivo Transporta) – Ovaj nivo je zadužen da prilikom daljeg slanja podataka primljenih preko nivoa sesije, iste prvo pretvori u segmente (manje djelove). Ako se podaci šalju do primaoca (korisnika), transportni nivo je zadužen da segmente spoji u cijelinu, koja se šalje brzinom koja odgovara brzini konekcije korisnika;

5. Network Layer (Nivo Mreže) – Nivo mreže ima posao da segmente koje primi pretvori u manje mrežne pakete i omogući njihovo prosljeđivanje najboljom rutom kroz mrežu do elementa, a na strani korisnika da pakete spoji u jednu cijelinu;

6. Data Link Layer (Nivo Telekomunikacije) – Ovaj nivo uspostavlja konekciju između dva fizički povezana elementa na mreži, zatim pakete pretvara u "frames" ili okvire i njih šalje do cilja. Sastoji se od "Logical Link Control" (Kontrola logičnom vezom) ili LLC i "Media Access Control" (Kontrola pristupa mediju) ili MAC. LLC je zadužen da identifikuje mrežne protokole, sinhronizuje okvire i slično. MAC je jedinstvena adresa uređaja koji šalje ili prima podatke;

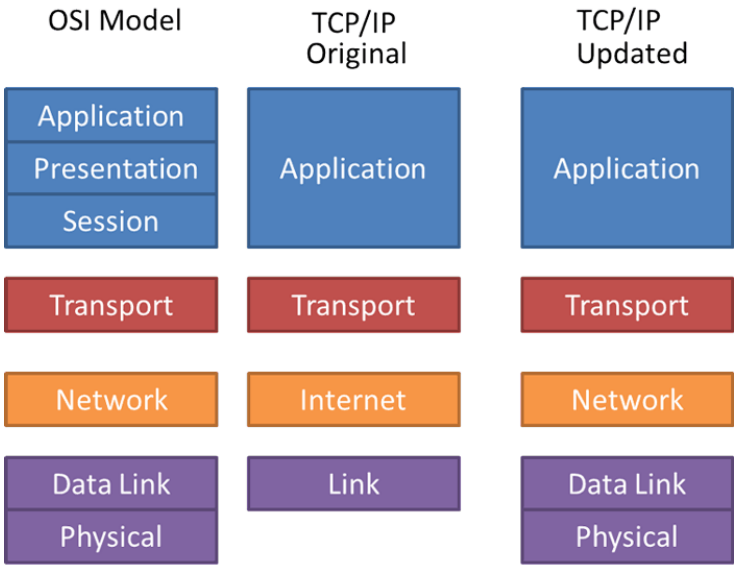
7. Physical Layer (Fizički Nivo) – Nivo koji je zadužen za fizičku ili bežičnu vezu između mrežnih elemenata. Odgovorna je za prenos podataka, definisanih kao jedinice i nule, a pri čemu i kontroliše njihov odnos bita;



Slika 1: Sedam nivoa OSI modela (https://www.javatpoint.com/osi-model)

Na slici 2 možemo vidjeti razliku između OSI modela i TCP/IP modela. TCP/IP model je jednostavniji. Velika je razlika u broju nivoa. Kod OSI modela vidimo aplikativni, prezentacioni i nivo sesije, dok kod TCP/IP modela to sve čini jedan nivo aplikacije. Ostali nivoi su poprilično slični.

Prva četri nivoa OSI modela (aplikativni, prezentacioni, nivo sesije, transportni) smatramo da su odgovornost domaćina, dok ostala tri (mrežni, telekomunikacioni, fizički) nivoa smatramo da su odgovornost mreže.



Slika 2: OSI model nasprem TCP/IP modela (https://www.computernetworkingnotes.com/ccna-study-guide/similarities-and-differences-between-osi-and-tcp-ip-model.html)

2.2 Bezbjedonosne prijetnje i načini odbrane

Ova sekcija je samo uvod i ne govori detaljno o vrstama bezbjedonosnih prijetnji koji postoje, kao i načinima odbrane od istih. Pomenute su samo one prijetnje za koje smatramo da su bitne za ovaj master rad. To takođe ne znači da ne smatramo da je bitna zaštita i od ostalih, ali s' obzirom na veličinu oblasti, pokušavamo da suzimo listu na one prijetnje koje su najučestalije.

Neki od najčešćih bezbjedonosnih prijetnji na aplikativnom nivou su "Distributed denial of service" (Distribuirani prekid usluge) ili DDoS napadi, SQL injections (SQL upad), Cross-site

scripting (skriptovanje), Cross-site request forgery (lažiranje zahtjeva), Slowloris napadi i slično. Slowloris napad je jedan od veoma interesantnih vrsta bezbjedonosne prijetnje, jer je veoma teško za tradicionalne sisteme detekcije da detektuju ovu vrstu bezbjedonosne prijetnje, zbog načina na koji se on izvršava i zbog toga smo ga stavili kao posebnu cijelinu i ako se radi o vrsti DDoS napada, a o njemu ćemo govoriti malo više kasnije. U svakom slučaju, sve ove prijetnje imaju zajednički cilj, a to je da iskoriste slabosti koje aplikacija ima. Kao što postoje prijetnje, tako postoje i načini da se iste uklone. Neki su specifični za određene vrste prijetnji, dok su neki zajednički za više vrsta. Nakon što kažemo po nešto o gore navedenim bezbjedonosnim prijetnjama, proći ćemo kroz par načina odbrane od istih.

1. Distributed denial of service – DDoS je još jedna veoma interesantna vrsta prijetnje, ili napada da budemo precizniji. Razlog zašto je interesantan je što može da utiče dvostrano, odnosno može da utiče i na napadača i na žrtvu. Ali što je još interesantno kod njega jeste što omogućava napadaču da iskoristi infrastrukturu žrtve i preko nje pokrene "Botnet-based" napad, odnosno napad zasnovan na mrežnim botovima, na druge kompanije, preko kompanije koja je originalno žrtva napada. Šta je napad zasnovan na mrežnim botovima tačno? To je mreža uređaja koji su inficirani, a koje napadač kontroliše preko servera putem komandi i koja omogućava između ostalog i rušenje onlajn mreže njenim preopterećenjem. Neke od vrsta DDoS napada na aplikativnom nivou su Slowloris, Slow Post (Sporo postavljanje), Slow Read (Sporo čitanje), HTTP ili HTTPS Flooding (HTTPS poplave), Low and Slow Attack (Niski i spori napad), Large Payload POST (Veliki teret POST), Mimicked User Browsing (Mimikovana korisnička pretraga).

2. SQL Injections – sastoji se od unošenja podataka u "input" polje ili ulazno polje neke aplikacije koja sadrži bazu koja koristi "Query" (upit/upitni) jezik. Podaci koji se unose u polje bi imali sintaksu upita, što bi dozvolilo izvršavanje administratorskih operacija modifikovanje same baze, odnosno brisanje, ažuriranje ili bilo koji drugi "REST" zahtjev. REST je skraćenica od Representational State Transfer (Transfer reprezentativnog stanja). Neke od opasnosti koje SQL injection predstavlja jesu da omogućavaju napadaču da

mijenja postojeće podatke u bazi, pristupi svojim bankovnim računima, izmijeni ih, pristupi tuđim bankovnim računima, izbriše bazu i još mnogo toga. Najveći problem je što je veoma lako izvršiti SQL injection.

3. Cross-site scripting – XSS napadi su vrsta upada, gdje se maliciozne skripte ubacaju u sajtove koji izgledaju kao od povjerenja. XSS manipuliše sajt tako da prilikom učitavanja vraće maliciozni JavaScript korisnicima. Nakon što se kod koji je postavio napadač, iskoristivši slabosti sajta, izvrši u pretraživaču korisnika, napadač dobija potpuni pristup aplikaciji i svim korisnikovim podacima. Vrste XSS-a su Reflected XSS (Reflektovani), Stored XSS (Skladišten), DOM-based XSS (Baziran na DOM-u).

4. Cross-site request forgery – CSRF napad koji omogućava napadaču da djelimično zaobiđe "the same origin policy" (Polisa istog porijekla), pravilo koje onemogućava da neki drugi sajt ili link nekog drugog sajta ugrađenog u sajt na koji se korisnik trenutno nalazi utiče na isti. Primjer ovoga napada bi bio da korisnik prilikom klika na link pošalje zahtjev za brisanjem svoga naloga na trenutni sajt. To se izvršava tako što podaci koji se na primjer čuvaju u kolačićima na pretraživaču iskoriste da se pošalje zahtjev za brisanjem datog naloga, a inače se kolačići šalju sajtu svaki put kada se neki zahtjev pošalje. Na isti način napadač može da izvrši promjenu email-a, šifre ili da pošalje neke podatke kao što su bankovni, naravno ako su povezani sa nalogom. Tri uslova koja moraju da se ispune su akcija, sesija zasnovana na kolačićima i ne postojanje parametara sa posebnim zahtjevom [7].

5. Slowloris – vrsta DDoS napada gdje se koriste parcijalne HTTP zahtjeve da bi se kreirala veza između korisnika i *web* servera i koja se drži otvorenom koliko god to bilo neophodno. Ne zahtijeva veliku količinu podataka za transfer podataka preko Interneta u datom vremenu, a opet uspijeva da drastično usporava metu. Veoma je spor i metodičan, jer zahtjevi koji se šalju se nikad ne izvrše, a to podstiče server da pokušava da uspostavi konekciju i time održava

komunikaciju i konekciju, dok je ne optereti i kompletno ne spriječi sve zahtjeve prema serveru sa datim portom. Drugim riječima on okupira sve threads (niti), do tačke gdje ne može više da se uspostavi ni jedna konekcija.

U sledećem poglavlju će se više govoriti o samim upadima. O pojedinim se govori više, a o pojedinim manje, ali je analiza potpunija i sami problemi više razumljiviji. Sada će biti pomenute neke od vrsta zaštite od gore navedenih bezbjedonosnih prijetnji. U poglavlju gdje se govori o zaštiti, odnosno prevenciji od bezbjedonosnih prijetnji, ćemo više analizirati samu temu, od pripreme do prevencije istih. Pod nekim od najčešćih načina zaštite od bezbjedonosnih prijetnji spadaju:

1. Application-level firewall logic (Fajervol logika na aplikativnom nivou) – najpoznatiji koncept zaštite, koji određuje koji podaci mogu da prođu, a definisan je nekim unaprijed postavljenim pravilima. Generalno govoreći (tradicionalno) firewall jeste graničnik mreža (network) ili podmreža (subnet) i ima fizičku formu, odnosno jeste hardverski uređaj. Ali firewall o kojem ćemo mi govoriti je na aplikativnom nivou i on kao tradicionalni firewall filtrira koji zahtjevi mogu dalje proći ili ne, takođe prema unaprijed definisanim pravilima. Tema koju ne možemo zaobići jeste WAF ili *Web* Application Firewall (Mrežni aplikativni fajervol), ako govorimo o firewall-ima na aplikativnom nivou, tj. WAF je programsko rješenje unutar posebnog mrežnog uređaja, koje štiti od raznih upada. Najpraktičnije rješenje je korišćenje WAF-a ugrađenog u aplikaciji. Funkcionišu tako što se koriste razni filteri sa logikom što je prihvatljivo, a što ne. Takođe firewall koji se veoma često koristi jeste Database Firewall (Firewall Baze Podataka), o njemu ćemo pričati nešto više u poglavlju četiri.

2. The Security *Web* Gateway – SWG ima za cilj da zaštiti korisnike da pristupe sajtovima koji su zaraženi ili posjeduju sadržaj koji je zabranjen unutar gateway-a. Ovaj koncept je dosta sličan Application-level firewall logici i često se pogrešno smatra da su ista stvar. Razlika je što firewall štiti od upada koji su

inicirani od strane napadača. SWG se koristi unutar kompanija, kada korisnici pokušavaju da pristupe eksternim sajtovima. Provjeravaju se parametri, kolačići i Uniform Resource Locator (Uniformni lokator resursa) ili URL link. Ako URL ne odgovara ni jednom iz security policy (zaštitne polise), onda se korisniku kao odgovor vraća status greška. Ako je URL dobar provjeravaju se dalje parametri i kolačići, odnosno njihova imena. Ovaj proces je mnogo složeniji nego što je predstavljeno ovdje, ali ćemo ga detaljnije proći u poglavlju zaštite.

3. Runtime application self-protection – RASP služi da nadgleda i zaštiti pojedinačnu aplikaciju na nivou organizacije. On nadgleda ulaze, izlaze, kao i samo ponašanje aplikacije, korišćenjem introspekcije. Jedna stvar koja je veoma važna jeste da mora biti pokrenut na istom uređaju kao i sama aplikacija. RASP i *Web* application & Application Programming Interface (API) protection (Zaštita mrežne aplikacije i aplikativnog programskog interfejsa) ili WAAP se pretežno koriste za cloud protekciju. On ima mogućnost detekcije promjena na aplikaciji, što mu omogućava identifikovanje i zaštitu od same bezbjedonosne prijetnje. Velika prednost kod korišćenja RASP-a jeste što bukvalno nadgleda što aplikacija radi i time olakšava sami rad, jer nema potrebe za bilo kakvom modifikacijom.

4. *Web* application & API protection – WAAP se smatra evolucijom aplikacionog firewall-a. Proširuje njegov opseg kao i dubinu zaštite, ali on nije firewall, već veoma visoko specijalizovana alatka, koja služi za zaštitu aplikacija i API-ja. Stoji na spoljnoj ivici mreže (outer edge) na javnoj strani *web* aplikacije i analizira sav saobraćaj, pri čemu se samo bazira na aplikativni nivo. Prednosti koje WAAP ima jeste da je prilagodljiv, odnosno stalno uči i veoma je automatizovan.

5. Load balancer (Balanser opterećenja) i Reversed proxy (Obrnuti posrednik) – Load balancer ima za cilj da distribuira mrežni ili aplikacioni saobraćaj između

broja dostupnih servera. Oni omogućavaju povećanje kapaciteta trenutnih korisnika i samim time omoguće siguran rad aplikacije. Na nivou sedam load balancer se koncentriše na sadržaje samih poruka, pri čemu može da terminiše mrežni saobraćaj i pročita sadržaj. Potom kreira novu TCP konekciju ka novom serveru po izboru ili samo ponovo iskoristi postojeću. Reverse proxy može da vrši sve što i load balancer čak i više, u smislu da pored optimizacije, odnosno balansiranja on dodatno može da vrši protekciju i da čuva identitete samog servera. Ujedno je posrednik koji može da donosi odluke, a pri tom ako se dobro konfiguriše, sprečava direktnu komunikaciju sa serverom, odnosno sva komunikacija do servera ide preko reverse proxy-ja.

Na slici 3 se može vidjeti slikoviti primjer kako izgleda komunikacija aplikacije i servera i eventualno baze na kraju, kao i zaštita aplikacije putem firewall-a koji se nalazi između klijenata i samog servera.

Slika 3: Firewall aplikacije (https://bunny.net/academy/security/what-is-*web*-application-firewall-WAF-and-how-it-works/)

Gore navedeni primjeri prevencije od bezbjedonosnih prijetnji su jedni od najkorišćenijih sistema prevencije, što ne znači da su uvijek i uspješni. Razlog je što se stalno prave nove bezbjedonosne prijetnje ili se ažuriraju već postojeće. Kako se kreiraju nove prijetnje, tako se i kreiraju nova rješenja, ili unapređuju već postojeća. Ovo je neprestalni ciklus i to je razlog zašto

su neophodni novi automatizovani sistemi, odnosno nova softverska rješenja koja imaju sposobnost učenja i automatizacije procesa.

2.3 Cloud infrastruktura i cloud aplikacije

Cloud infrastruktura je skup fizičkih i virtualnih djelova, odnosno hardvera i softvera, neophodnih da bi se vršio "cloud computing" (računarstvo u oblaku), proces koji se odnosi na bilo koju vrstu zahtjeva IT resursa preko interneta. Ova vrsta tehnologije omogućava veliku uštedu, jer kompanija nema potrebu da kupuje fizičke centre podataka (data centers), nema potrebu da ih održava, kao ni potrebu da izdvaja prostor u kojem bi isti bili smješteni [8]. Sva komputaciona moć, skladišta, baze podataka, njihova virtualizacija i slično se sada nalaze na udaljenom serveru za koje se plaća mjesečna ili godišnja pretplata firmama kao što su AWS, Microsoft Azure, Google Cloud Platform i mnoge druge. Da bi nešto mogli da smatramo "cloud rešenjem" mora da posjeduje pet ključnih karakteristika. Mora da bude samouslužno po zahtjevu, da ima širok mrežni pristup, da omogućava pooling (udruživanje resursa), ubrzanu elastičnost i izmjerenu uslugu. Ove karakteristike označavaju da su svi resursi dostupni dvadeset četiri sata, sedam dana u nedelji klijentima, da su izdijeljeni u komponente koje ne zavise jedna od drugih i da sistemi posjeduju sposobnost da sami optimizuju upotrebu resursa. Korisnicima su cloud usluge dostupne u četiri modela (deployment models) i to "javni cloud", "privatni cloud", "hibridni cloud" i "cloud zajednice". Javni cloud se koristi od strane svih korisnika, koji žele da koriste hardverske i softverske elemente na primjer za razvoja aplikacije, testiranja i slično, a pri tom su spremni da plaćaju mjesečnu pretplatu. Privatni cloud se koristi od strane jedne kompanije, a istu održava sama kompanija ili druge kompanije koje se bave time. Ova vrsta cloud-a je definitivno i skuplja, mada u pogledu zaštite i rješavanja problema daleko ispred ostalih modela. Hibridni model je mješavina privatnog i javnog cloud-a. Kompanija koristi javni cloud kada treba da poveća drastično kapacitet unutar privatnog clouda, ovo se pogotovo izvršava tokom sezonskih posjeta aplikacijama. Cloud zajednica je vrsta cloud modela koji omogućava dijeljenje resursa između kompanija. Primjer korišćenja ove vrste clouda je postojanje više različitih univerziteta na istom kampusu. Oni pritom kreiraju zajednicu, a pristup podacima je ograničen samo na članove date zajednice [9]. Aplikacija koja se bazira na cloud-u je softver koji izvršava sve svoje logičke procese i skladištenje između dva različita sistema, (klijentska i serverska strana). Postoje tri vrste ovih

aplikacija, tj. tri vrste cloud servisnih modela [10], “Softver kao usluga” (SaaS), “Platforma kao usluga” (PaaS) i “Infrastruktura kao usluga” (IaaS).

Prvo ćemo opisati IaaS model. IaaS model omogućava klijentima najviše kontrole nad svojim resursima. Klijent zakupljuje preko Interneta infrastrukturu (mrežu, servere, skladišta i slično), a istoj mogu da pristupe preko API ili administratorskog panela, odnosno interfejsa. Dodatno opterećenje u ovom okruženju spada na administratore koji će održavati IaaS, pogotovo jer su sve konekcije na daljinu (remote).

PaaS model je vrsta modela koja omogućava fleksibilnu platformu za razvoj i podizanje aplikacija, dostavljena od strane trećeg lica na cloud-u (provajder). U njega spadaju alatke za razvoj, middleware, operativni sistemi, alatke za upravljanje bazama i slično. Sjajna prednost PaaS-a je što omogućava kolaboraciju većeg broja programera i automatsko postavljanje aplikacija po njihovom završetku. PaaS sadrži iste opasnosti kao i IaaS model, pri čemu je taj broj mnogo veći.

SaaS je model kod kojeg cloud provajder vrši hosting softverskog rešenja koji je dostupan klijentima preko interneta. Na udaljenom serveru se čuva jedna kopija koda, koja je identična za sve klijente. Ako se desi promjena koda, recimo novo ažuriranje, ta promjena će se izvršiti za sve krajnje korisnike. Ovaj softver može da se integriše u bilo čiju aplikaciju upotrebom API. SaaS model omogućava da se lakše održava kod, jer se ne moraju vršiti promjene na više različih instanci, već je dovoljno izvršiti promjene na jednom mjestu, što dodatno omogućava veću sigurnost, jer je kod isti za sve. Tri glavna rizika po ovu vrstu modela su [11]:

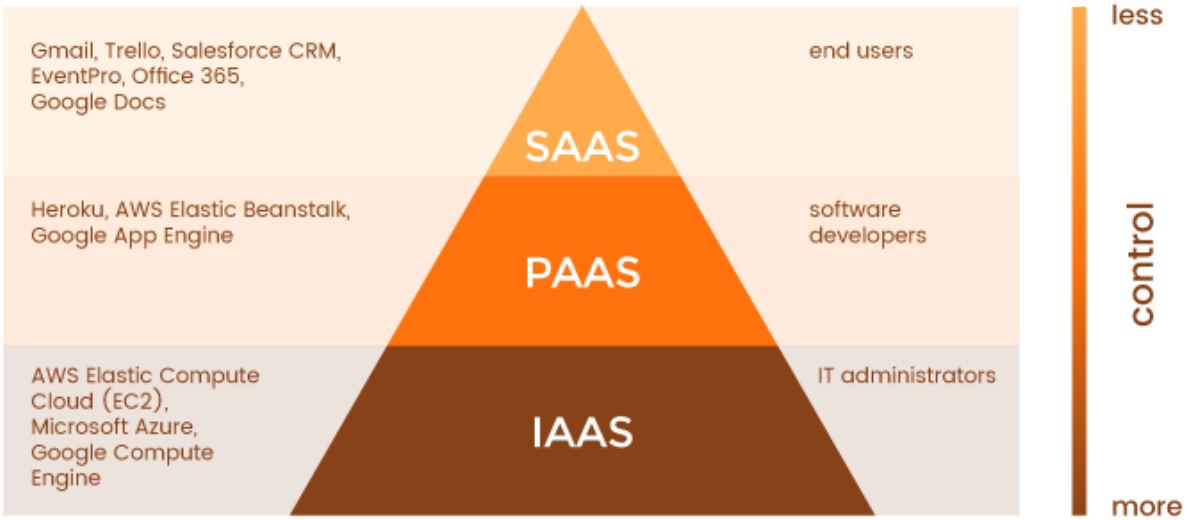
1. Vlasnički formati – provajder može da sakuplja i prikazuje podatke u formatu koji je jedinstven za njega, što smanjuje portabilnost aplikacije prodavca.

2. Virtualizacija – Predstavlja sami način rada cloud-a. Predstavlja provajdere koji sadrže centre podataka, na koji se kreiraju razna virtualna okruženja, kojima pristupamo preko API. Dva bitna pojma za virtualizaciju su Virtualna mašina (VM) i Hipervizor. VM je softverski definisan kompjuter sa svojim operativnim sistemom koji se pokreće na jednom računaru i taj računar može da ima više VM. Hipervizor je softver koji reguliše i prati više VM na računaru i pazi da svaka VM dobija potrebne resurse. Rizik se javlja, pogotovo u

SaaS modelu gdje ogroman broj ljudi dijeli resurse, u hipervizoru koji kao meta napada može da prouzrokuje pad svih VM.

3. Sigurnost *web* aplikacija – S' obzirom na to da se aplikacijama pristupa preko pretraživača, bilo koja slabost u sigurnosti *web* aplikacije može izazvati veliki broj problema.

Na slici 4 se mogu vidjeti cloud service modeli. Figura na slici je u obliku piramide, na kojoj se na samom dnu nalazi IaaS model, potom PaaS i na kraju SaaS model. Sa desne strane vidimo nivo kontrole po modelu, pri čemu što idemo na veći nivo to je količina kontrole manja i na samom kraju su krajnji 'end users' proizvodi, kao što su Gmail, Trello, Office 365 i druge više ili manje poznate aplikacije.



Slika 4: Cloud servisni modeli (https://rubygarage.org/blog/iaas-vs-paas-vs-saas)

Danas se dosta često koriste sve tri vrste aplikacija u kombinaciji, ali to sve zavisi od vrste zahtjeva i potreba koje odgovaraju klijentima, ali kao što se vidi iz prethodne slike i od nivoa kontrole koji je menadžment kompanija spreman da izgubi.

3. Bezbjedonosne prijetnje i vrste bezbjedonosnih prijetnji na aplikativnom nivou

Kao što smo ranije naveli, postoji više raznih vrsta bezbjedonosnih prijetnji na aplikativnom nivou. Sve one imaju zajednički cilj da pronađu i iskoriste slabosti, kako bi pristupili privatnim podacima korisnika ili kompanija, dok pojedine čak imaju i za cilj da samo nanesu štetu. Još jedna zajednička stvar koju sve ove bezbjedonosne prijetnje posjeduju jeste da mora da postoji lice iza kreacije i izvršenog napada. Ovo uključuje slučaj i kada se radi o "white ethical hacking" (bijelo etničko hakovanje), odnosno bijeli šešir i "black unethical hacking" (crno ne-etničko hakovanje) ili crni šešir. Razlika je u polju u kojem ova lica djeluju, odnosno da li se radi o hakerima koji su plaćeni da izvrše upad i testiraju aplikaciju ili sistem od bezbjedonosnih prijetnji naspram crnih šešira koji to rade ilegalno i imaju za cilj da zloupotrijebe podatke. O ovoj temi više govorimo u poglavlju namijenjenom za protekciju i identifikaciju. Tema poglavlja tri jesu vrste bezbjedonosnih prijetnji, njihovo izvršavanje, njihove mete, njihovi djelovi i potrebni uslovi da bi se isti izvršili. S' obzirom na to da dosta ovih programa koristi posebne i različite vrste programskih jezika, biblioteke, i slično, nećemo se toliko bazirati na kodu i njegovom pojašnjenju u slučajevima gdje se kod bude javljao, već na sam rad, odnosno izvršenje bezbjedonosne prijetnje.

3.1 Klasifikacija bezbjedonosnih prijetnji

Bezbjedonosne prijetnje možemo klasifikovati na više načina, ali dvije glavne metode za klasifikaciju su klasifikacija prema tehnici napada i prema tehnici uticaja koju ostvaruje prijetnja [12].

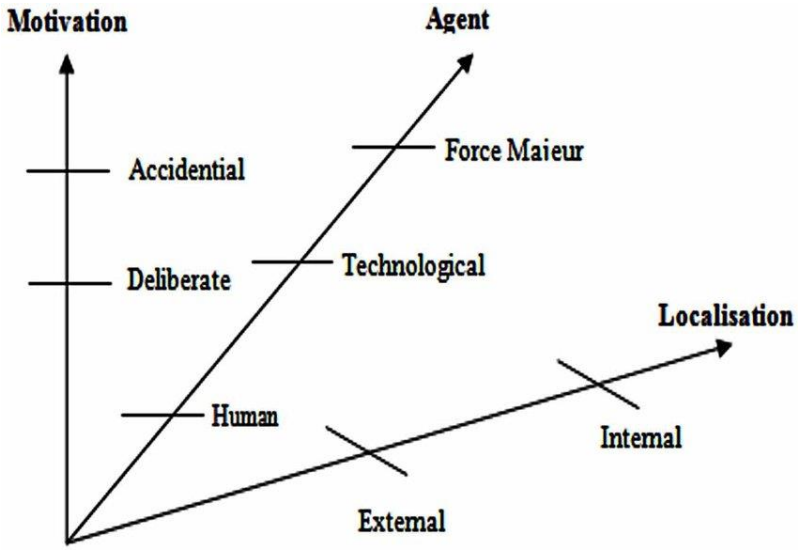
Metoda klasifikacije prema tehnici napada ima više metodologija koje se mogu primijeniti, na primjer trodimenzionalni model koji dijeli oblast prijetnji u više takvih podoblasti, što se može vidjeti na slici 5. Potom postoji hibridni model koji se drugačije naziva C3 ili kubni klasifikacioni model bezbjedonosnih prijetnji informacionih sistema (Slika 6). Sledeći model je piramidni klasifikacioni model informacionih bezbjedonosnih modela (Slika 7).

Trodimenzionalni model se zasniva na tri ortagonalne dimenzije:

1. Motivacija – ona predstavlja razlog kreacije prijetnje i dijeli se na dvije klase, namjerno izazvana meta i slučajna meta;

2. Lokalizacija – predstavlja porijeklo same prijetnje, da li se radi o internom ili eksternom porijeklu;

3. Agent – predstavlja samoga aktera koji izaziva prijetnju, a dijeli se na ljudski, tehnološki ili neku veću silu, na koju nemamo uticaj, odnosno neki drugi spoljni factor;



Slika 5: Trodimenzionalni model (https://www.researchgate.net/figure/The-three-orthogonal-dimensional-model_fig2_313241139)

Hibridni model se sastoji od:

1. Učestalost bezbjedonosnih prijetnji – kao što mu samo ime govori, prikazuje učestalost bezbjedonosne prijetnje;

2. Oblast djelovanja bezbjedonosnih prijetnji – predstavlja prostor (oblast) u kome bezbjedonosna prijetnja djeluje. U našem slučaju to je aplikativni nivo;

3. Izvor bezbjedonosnih prijetnji – daje tipove porijekla bezbjedonosne prijetnje;

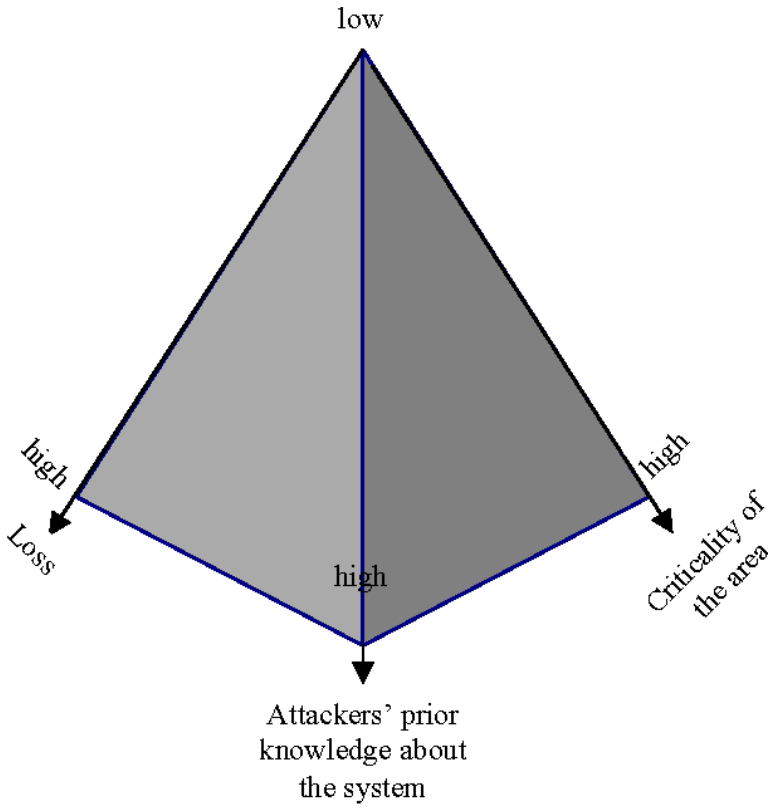
Slika 6: Hibridni model

Piramidni model se sastoji od tri činioca:

1. Prethodno znanje korisnika o sistemu – predstavlja koliko napadač zna o sistemu u oblastima hardvera, softvera, zaposlenih i slično. U našem slučaju ovo se odnosi samo na aplikaciju sa bazom;

2. Kritičnost područja – odnosi se na kritikalnost djelova aplikacije, koji mogu biti uslovljeni bezbjedonosnom prijetnjom;

3. Gubitak – svi gubici koji mogu proizaći od uspješnog izvršenja same bezbjedonosne prijetnje;



Slika 7: Piramidni klasifikacioni model (https://www.semanticscholar.org/paper/Information-Security-Threats-Classification-Pyramid-Alhabeeb-Almuhaideb/860fa25a5c28dd3698517729222bbf37f27ff6f4)

Pod klasifikacionim metodama zasnovanim prema tehnici uticaja koju ostvaruje prijetnja, takođe imamo par različitih metodologija. Jedna od njih je STRIDE model, koji može da se primijeni i za aplikacije između ostalog. STRIDE je veoma dobar model za klasifikaciju i karakterizaciju bezbjedonosnih prijetnji prema cilju i razlogu. Svaka od bezbjedonosnih prijetnji spada u jednu od ovih kategorija prema STRIDE-u i prvo slovo svake od ovih kategorija kreira i akronim STRIDE (Spoofing (Stažiranje), Tampering (Miješanje), Repudiation (Odbijanje), Information Disclosure (Otkrivanje podataka), Denial of Service (Prekid usluga), Elevation of Privilege (Povećanje privilegija)). Koristi se da bi se ušlo u um napadača, ali je problem što ne sadrži sve vrste prijetnji.

Postoje i drugi modeli klasifikacije bezbjedonosnih prijetnji kao što su ISO model, NIMS model i mnogi drugi, mada smatramo da je ovo dovoljno, jer ne želimo preduboku da zalazimo u samu klasifikaciju. Još jedan model koji ćemo pomenuti, ujedno i najbitniji model, jeste hibridni.

Većina klasifikacionih modela ne sadrži sve prijetnje ili kriterijume da izvrši klasifikaciju, a sama klasifikacija je veoma bitna, da bi svi uključeni bili spremni na moguće opasnosti i gubitke od istih, bilo da se radi o organizaciji kao cjelini ili pojedincima. Svi modeli koje smo do sada naveli, ne sadrže sve mogućnosti ili principe da uspješno potpuno izvrše klasifikaciju, zato se koristi model koji uključuje malo od svake. Taj model se naziva Multi-dimenzijalni model (Slika 8). Cilj modela je da se svi kriterijumi, gore navedeni, spoje u jedan model, odnosno da se kriterijumi od važnosti iskoriste na nivou cjeline i da se prikaže njihov impakt.

Ovaj model treba da sadrži sledeće kriterijume:

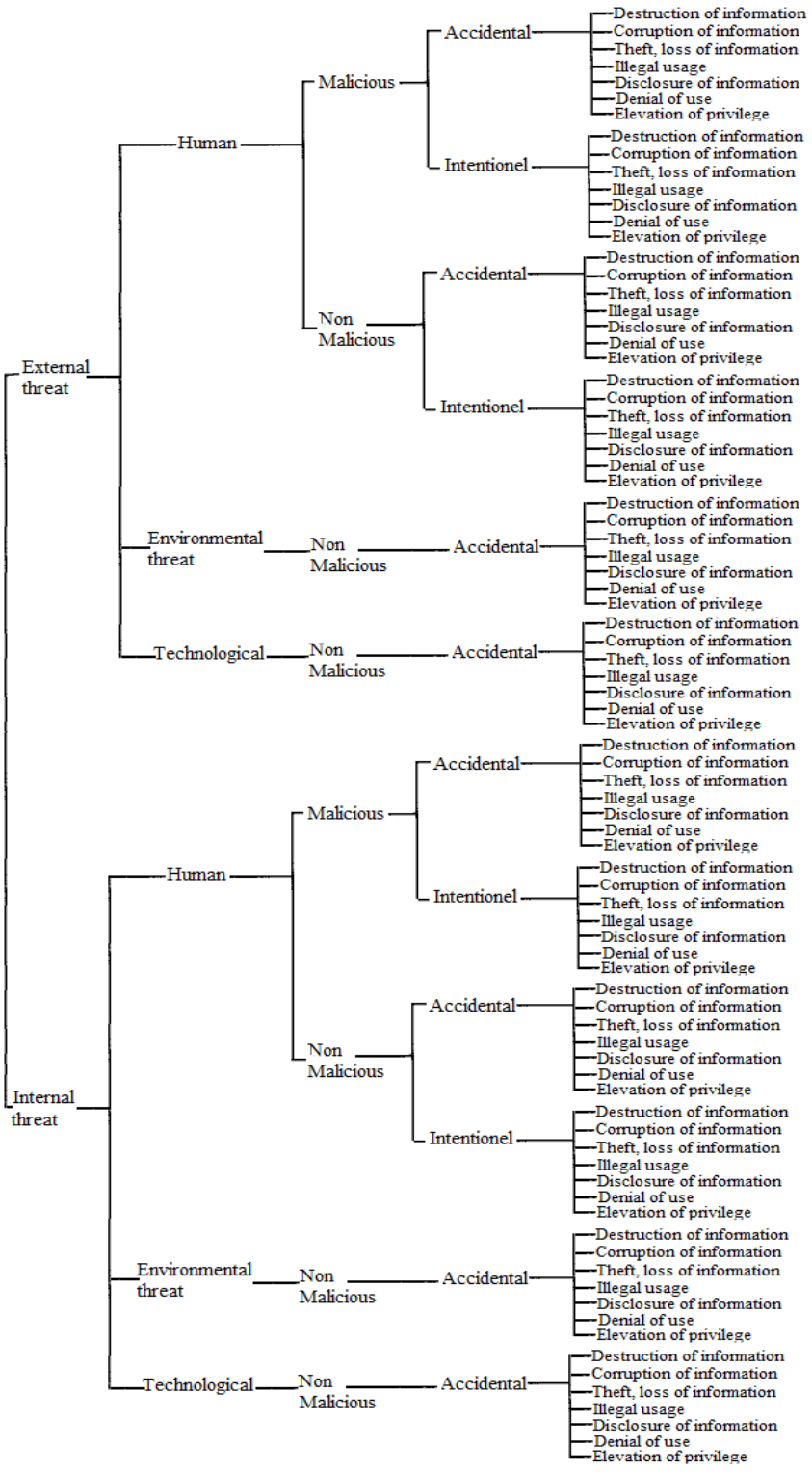
1. Izvor bezbjedonosne prijetnje – porijeklo prijetnje (eksterno ili interno);

2. Agente bezbjedonosne prijetnje – aktere (ljudske, tehnološke i viša sila);

3. Motiv bezbjedonosne prijetnje – cilj napadača (maliciozan ili ne maliciozan);

4. Namjera bezbjedonosne prijetnje – namjera napadača može biti namjerna ili slučajna. Ovo je veoma bitno u slučaju kada pokušavamo da preciziramo vrstu napada i kako je došlo do njega, da bi ga mogli rekreirati;

5. Impakt bezbjedonosne prijetnje – posljedica proizašla od izvršenja bezbjedonosne prijetnje. U primjeru koji ćemo dolje navesti impakt ili štete koje mogu da proizađu od izvršenja bezbjedonosne prijetnje jesu uništenje informacija, korupcija informacija, krađa ili gubitak informacija, ilegalno korišćenje, objelodanjivanje informacija, zabrana pristupa i krajnje, ne i najmanje bitno, uklanjanje privilegija;



Slika 8: Multi-dimenzijonalni model (https://www.researchgate.net/figure/The-multi-dimensions-threats-classification-model_fig1_315714820)

U ovom primjeru možemo jasno vidjeti kriterijum multi-dimenzijalnog modela. Izvor bezbjedonosne prijetnje može biti interni ili eksterni. Pod internim spadaju sve prijetnje koje proizilaze unutar kompanije, a za nju su krivi sami zaposleni, nepažnjom ili namjerno. Eksterni izvori su oni izvori koji dolaze iz spoljnog svijeta, preko mreže. Agenti bezbjedonosne prijetnje su, u oba slučaja internog ili eksternog izvora, ljudski, tehnološki ili prirodni. Ljudski agenti su ljudi (hakeri), tehnološki agenti su bilo koja vrsta fizičke povrede tehnologije ili krađe iste i na kraju pod prirodnim spadaju sve štete izazvane vremenskim nepogodama. Motivi mogu biti maliciozni, odnosno da imaju za cilj da ukradu podatke, nasilno pristupe istim ili naprave štetu. Takođe mogu biti nemaliciozni, kada postoje loše polise osiguranja, koje izazivaju greške. Namjera bezbjedonosne prijetnje može biti slučajna kada se slučajno modifikuje kod (aplikacija), ili namjerna, gdje se namjerno mijenja aplikacija u cilju pristupa podacima koji ne pripadaju napadaču. Na samom kraju imamo impakt bezbjedonosne prijetnje, koje smo već pomenuli gore.

Svaka prijetnja treba da ima neki vid klasifikacije, da bi znali kako da se odnosimo prema njoj, odnosno odakle da krenemo sa njenom prevencijom, naravno i prije toga sa njenom identifikacijom.

3.2 Bezbjedonosne prijetnje

Do sada u uvodnom poglavlju smo pomenuli neke od češćih bezbjedonosnih prijetnji koji se izvode na aplikativnom nivou, a o kojima ćemo pričati više u ovom potpoglavlju. Pod tim spadaju DDoS napadi, SQL injections, Cross-site scripting, Cross-site request forgery i mnoge druge manje specifične, ne i manje bitne, bezbjedonosne prijetnje.

Postoje razne varijacije istih napada, ali u zavisnoti od načina djelovanja, kao i ciljanje mete, možemo ih lako klasifikovati (Pr. DDoS napad, Cross-site request forgery i slično).

Prije nego krenemo da ih dijelimo i objašnjavamo, moramo preći kroz jedan veoma bitan pojam, a to je “Taksonomija bezbjedonosnih prijetnji.

3.2.1 Taksonomija bezbjedonosnih prijetnji

Taksonomija je pojam koji služi za fasilitaciju bezbjedonosnih prijetnji prema njihovom razumijevanju i poređenju.

Da odmah razjasnimo da ovaj pojam iako zvuči sličan klasifikaciji, nije ista stvar. Klasifikacija bezbjedonosnih prijetnji služi samo za grupisanje više objekata, u ovom slučaju bezbjedonosnih prijetnji, dok taksonomija bezbjedonosnih prijetnji opisuje odnose između istih. Ovaj pojam je mnogo bitan za detekciju i prevenciju bezbjedonosnih prijetnji. Razlog zašto se nalazi ovdje kao dio ovoga poglavlja je da bi mogli razumjeti bezbjedonosnu prijetnju, shodno tome moramo biti upoznati sa onim što ih veže ili razlikuje. Postoji mnogo vrsta predloženih taksnomija, mada nećemo zalaziti u njihovo poređenje, jer to nije cilj ovog master rada i smatramo da je to dovoljno velika oblast da može biti tema master rada ili nekog drugog naučnog istraživanja, mi ćemo objasniti samo jednu vrstu kao primjer, koja se zove AVOIDIT [13].

Da bi se razvila uspješna taksonomija moraju se ispuniti određeni uslovi, a to su:

1. Prihvaćenost – mora da bude izgrađena na prethodnim primjerima koji su bili takođe prihvaćeni;

2. Međusobna isključivost – svaka bezbjedonosna prijetnja može da se klasifikuje samo jednom;

3. Shvatljivost – jasne i koncizne informacije;

4. Kompletnost – upotpunjenost kategorija;

5. Nedvosmislenost – jasne klase sa jasnim razumijevanjem koja prijetnja pripada kojoj klasi;

6. Ponovljivost – klasifikacija klasa treba da je ponovljiva;

7. Jasna definisanost uslova – jasno definisane kategorije sa jasno definisanim uslovima koji se slažu sa terminima zajednice zaštite;

8. Korisnost – mogućnost da izučavanja budu jasna i korisna;

Sa time na umu opisaćemo AVOIDIT metodu za taksonomiju bezbjedonosnih prijetnji. AVOIDIT naziv je akronim nastao iz više kategorija ili potkategorija koji spadaju pod ovom taksonomijom. AV stoji za "Attack Vector" (Vektor napada), OI stoji za "Operational Impact" (Operacioni impakt), D stoji za "Defense" (Odbrana), I stoji za "Informational Impact" (Informacioni impakt) i T stoji za "Target" (Meta). Svaka od ovih kategorija sadrži veliki broj drugih potkategorija u koje ne zalazimo, a na slici 9 se mogu vidjeti sve podjele do detalja. Mi ćemo samo objasniti glavne kategorije, jer tema master rada nije taksnomija, ali ju je neophodno pomenuti.

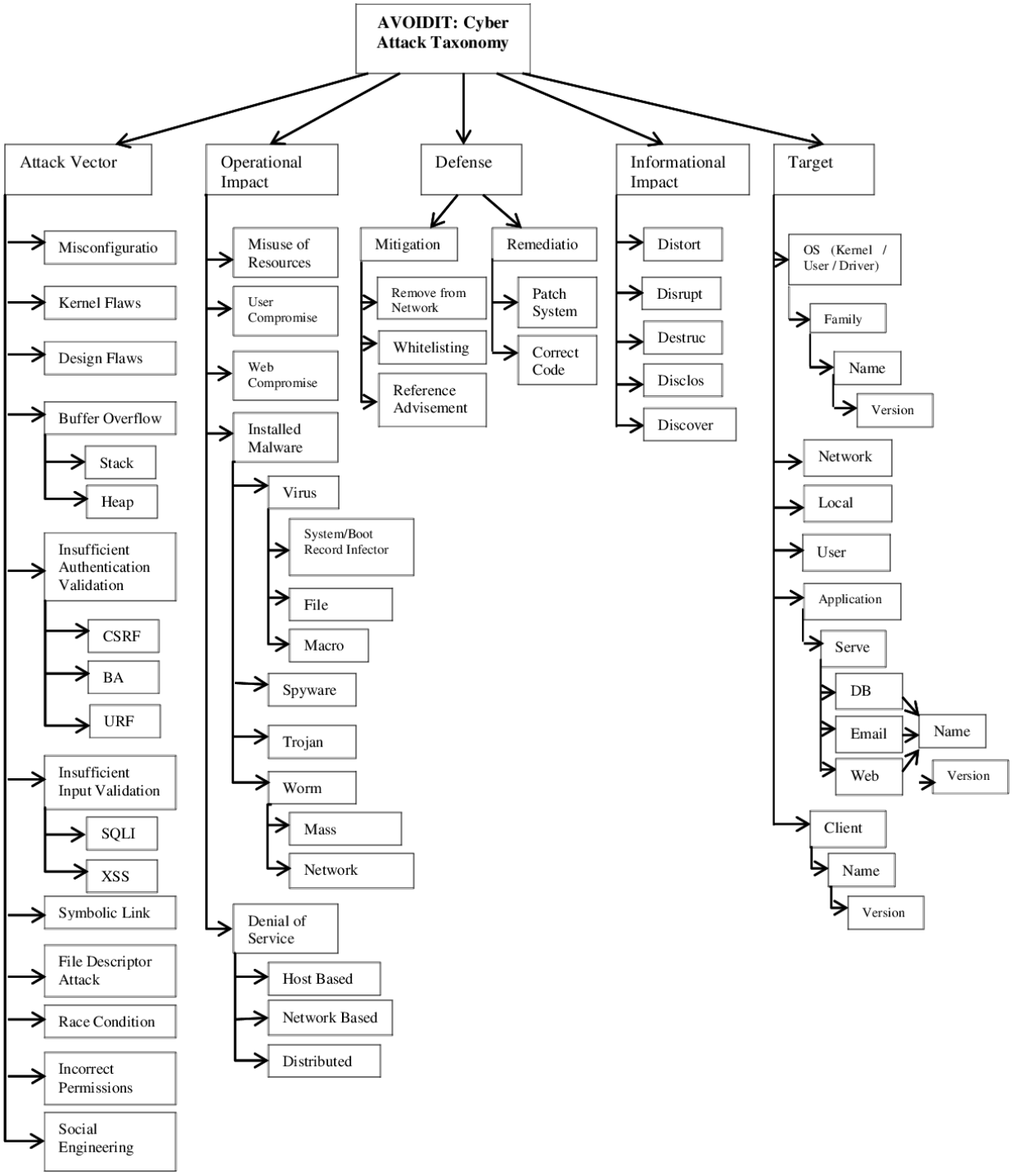
Klasifikacija vektorom napada se naziva tako jer ona predstavlja putanju koju prijetnja mora da prođe da bi se izvršio napad na domaćina. Pod ovom kategorijom takođe spadaju slabosti ili ranjivosti koje se iskoriste da bi se prijetnja realizovala, tj. da bi se prijetnja uspješno izvršila.

Klasifikacija operacionim uticajem predstavlja sposobnost počinioca bezbjedonosne prijetnje da sakupi i iskoristi informacije, koje se nalaze na visokom nivou, a poznate su ekspertima ili onima sa nižim znanjem i iskoristi ih za svoje potrebe.

Klasifikacija odbranom se koristi za ono što joj samo ime kaže. To je skup strategija koje mogu da se primijene da se napadnuta strana zaštiti od napadača, odnosno služi za uklanjanje i saniranje nanijete štete. Veoma bitna kategorija, posebno na nivou zaštite pre i post napada.

Klasifikacija informacionim uticajem označava sposobnost zaštite osjetljivih informacija, pojedinca ili grupe, kojima te informacije pripadaju. Cilj je da se zaštiti od krađe, disrupcije, distorzije i uništenja osjetljivih podataka.

Klasifikacija metom napada podrazumijeva postojanje velikog broja ciljanih meta, koje mogu da omoguće lakše prodiranje ili propadanje odbrane, strane koja pokušava da se odbrani. To označava da svaki sledeći napad može da bude i krajnji, u smislu da se odbrana ostavlja bez znanja o prodornosti sledećeg napada.



Slika 9: AVOIDIT (https://www.semanticscholar.org/paper/AVOIDIT%3A-A-Cyber-Attack-Taxonomy-Simmons-Ellis/4a58f71bb0646755ac732b29da8a55b1dd5c0122)

3.2.2 Bezbjedonosne prijetnje – vrste i podjele

U narednim dijelu ćemo proći kroz par primjera bezbjedonosnih prijetnji. Ovi primjeri uključuju Distributed denial of service napad ili DDoS, SQL injection, Cross-site scripting ili XSS i Cross-site request forgery ili CSRF.

3.2.2.1 Distributed denial of service

Prva vrsta bezbjedonosne prijetnje o kojoj ćemo detaljnije govoriti jeste DDoS napad. Način na koji DDoS napad funkcioniše jeste takav da napadač iskoristi slabosti na računaru korisnika i na taj način dobije pristup. Napadač da bi dobio pristup uređaju korisnika mora da instalira virus koji pokreće neku skriptu ili kreira neki pozadinski ulaz, a ponekad dodijeli napadaču i prava administratorskog nivoa. Veoma često napač uspostavlja i komunikaciju na udaljeni server, "Command & Control" (Zapovijedaj i Kontroliši) ili C&C, kako bi mogao da u određeno vrijeme izda napad na željenu metu. Mreža računara, koje nazivamo zombijima ili botovima, istovremeno pokreće napad (preopterećenje) na neku mrežu ili *web* server i time ga uspori i na kraju dovede do njegovog pada. Ali DDoS napadi koji se izvršavaju na aplikativnom nivou, ciljaju samu aplikaciju, odnosno njenu sposobnost da dostavi sadržaj krajnjim korisnicima. Ovo ostvaruje tako što iskorištava slabosti koje ta aplikacija posjeduje. Ova vrsta bezbjedonosne prijetnje je dizajnirana tako da napada specifične aplikacije koje mogu biti *web* serveri, ali su na meti takođe i razne aplikacije kao što su "Session Initiation Protocol" (Protokol inicijacije sesije) glasovni serveri ili SIP i "Border Gateway Protocol" (Protokol granice prolaza) ili BGP i druge. Ovakva vrsta DDoS napada se primarno izvršava korišćenjem diskretnih pametnih klijenata. Kada kažemo diskretni inteligentni uređaji, pretežno mislim na uređaje koji ne mogu biti "spoofed" ili prevareni, kao što su "Internet of Things" (Internet stvari) ili IoT uređaji. DDoS napadi na nivou aplikacije ili aplikativnom nivou se mogu detektovati korišćenjem analize toka zasnovane na zaštiti, ali je takođe neophodno da se koristi analiza ponašanja ili nešto što se naziva "deep packet analysis" (analiza paketa po dubini). Više o ovome ćemo govoriti u sledećim poglavljima. Sa rastom broja IoT uređaja sve su češći napadi, a i kompleksniji, na aplikativnom nivou. Broj napada na IoT uređaje je veoma učestal i do sada je zabilježen veliki broj upada, koji koriste ranjivosti i kao metu imaju rutere preko nezaštićenih ili slabo zaštićenih IoT uređaja. Primjer napada na rutere

preko slabo zaštićenih IoT uređaja je “VPNFilter malware“, odnosno virus koji ima za cilj da zarazi ruter i prikupi podatke sa istog [14]. IoT je tehnologija koja se može koristiti u svim poljima da pomogne ljudima, ali i da ih ukloni od opasnosti. IoT počinje da se koristi kako u medicini, tako i u vojnom naoružanju (za detekciju, zaštitu, pa čak i vođenje rata na daljinu). Ali na trenutnom nivou postoji dosta nesigurnosti što se tiče zaštite vezane za pametne uređaje koji su vezani na ovaj način. Ovo se odnosi i na hardverski, simulacioni (testiranje upada u simulacionom okruženju) i softverski pristup. Ali definitivno pristup od tri koji bi se mogao smatrati najpropustljivijim i najvećim krivcem jeste softverski pristup, jer se većina grešaka ili slabosti koje se najčešće koriste kriju u samom kodu koji je pisao pojedinac ili tim [15]. Veliki problem predstavlja sposobnost učenja napadača, odnosno prilagodljivost i odlučnost. Svaki neuspjeli napad omogućava napadaču da nauči zašto napad nije uspio i da ga izmijeni u područijima koji su bili odgovorni za neuspjeh. Zbog tih stalnih promjena, ne postoji šablon po kojem bi se uklanjale prijetnje, već one stalno napreduju i mijenjaju se. Zbog toga je neophodno stalno prilagođavanje softvera tim promjenama. Proces odbrane i kreiranja novih DDoS napada je često kružan, u smislu da kreiranje novih napada izaziva i kreiranje nove odbrane. Pod nekim od čestih DDoS napada na aplikativnom nivou spadaju Slowloris [16], Slow Post [17], Slow Read [18], Low and Slow Attack [19], Large Payload POST i Mimicked User Browsing [20], HTTP(/s) Flooding [21].

- Slowloris napad je bezbjedonosna prijetnja na aplikativnom nivou. To je vrsta DDoS napada koja koristi parcijalne HTTP zahtjeve da drži konekciju otvorenom trajno, odnosno onoliko koliko je to potrebno da se uspori aplikacija i sruši server. Već smo rekli da nije neophodan veliki protok, odnosno dovoljan je mali protok da bi se održala konekcija otvorenom. Ova vrsta DDoS napada je veoma spora i metodična, HTTP zahtjevi koji se šalju su “beskonačni“, odnosno nikad se ne izvršavaju do kraja. Cilj je da se portovi (jedan po jedan) namijenjeni za konekciju sa serverom prenatrpaju zahtjevima, do te mjere gdje bilo koji stvarni zahtjev više ne može da prođe, odnosno gdje on biva odbijen. Tradicionalni sistemi za detekciju, ili sistemi za detekciju i prevenciju nisu mnogo uspješni u detektovanju ovih vrsta napada. Ovu vrstu napada smatramo upornim napadom, jer čak i kada je neki port oboren (time out), to neće spiječiti napad da pokuša da ostvari konekciju sve dok cijeli server ne bude oboren. Da stvar bude gora , Slowloris DDoS napad može da šalje različite header-e (zaglavlja) domaćina, u slučajevima kada su logovi

skladišteni odvojeno. Takođe, ima mogućnost da spriječi pojavu obavještenja da nešto nije uredu i time ostane nedetektovan.

- Slow Post napad je vrsta DDoS napada koja ima za cilj da uspori komunikaciju sa serverom do toga nivoa gdje neće moći da se ostvari legitimna konekcija. Napad funkcioniše tako što se šalje pravi HTTP POST zahtjev sa header-om na *web* server. Čak je i veličina tijela poruke koja se šalje tačno specificirana. Pa kako napad uspijeva da nanese štetu i dovede do pada konekcije? Tijelo poruke se šalje brzinom koja je nevjerovatno spora, čak i do nekoliko bajta po sekundi. Imajući u vidu da poruka poštuje sve protokole i ne krši nikakva pravila, tj. rad sa porukom se izvršava normalno, serveru ne preostaje ništa osim da prati sva propisana pravila. Sada kada se ovo multiplikuje ogroman broj puta, to dovodi ne do usporavanja, već i do samog pada svih konekcija i nemogućnosti uspostavljanja istih. Ova vrsta DDoS napada je okarakterisana slanjem HTTP POST zahtjeva sa header-om koji namjenski cilja *web* servere koji su bazirani na thread-u (niti), za razliku od onih koje su bazirani na procesu. *Web* server baziran na niti označava da se sa svakim zahtjevom kreira nova nit. Pri tom se pri ovoj vrsti DDoS napada podaci šalju ekstremno sporo, držeći se na donjoj granici da se sesija, odnosno server ne isključi. To označava da server čeka dodatne podatke koji se veoma sporo šalju POST zahtjevom, a pravi korisnik ne može istom da pristupi. Takođe ova vrsta napada ne zahtijeva veliki protok, kao ni alokaciju velikog broja sredstava, što znači da ne treba puno da bi se napad izvršio, kao ni da napad izgleda kao obični saobraćaj.

- Slow Read DDoS napad je prijetnja gdje napadač šalje potrebne HTTP zahtjeve prema serveru, ali za razliku od Slow post DDoS napada, klijent čita podatke veoma malom brzinom (ako ih uopšte čita), pa čak i do jednog bajta u datom trenutku. To omogućava napadaču da spriječi server da prekine konekciju, jer se komunikacija i dalje održava, ali veoma malom brzinom. Napadač šalje nešto što nazivamo nultim prozorom. Nulti prozor je odgovor koji označava da je protok podataka trenutno nemoguće izvršiti i da se sačeka da se napravi još prostora . To server interpretira kao da korisnik još uvjek čita primljeni odgovor, što propratno znači da

korisnik i server i dalje komuniciraju i da server treba da održi komunikaciju. Isto kao i kod Slow post-a ovo polako crpi sve resurse servera, dok ne dovede do toga da se ne može uspostaviti ni jedna legitimna komunikacija. Jedna stvar koja karakteriše ovu vrstu DDoS napada je mala veličina TCP Receive (TCP primaoca) prozora, kao i crpljenje napadačevog TCP Receive bafera, što izaziva veoma nisku brzinu protoka. Isto kao i sa Slow postom, ne treba mu velika brzina i velika alocirana sredstva, pa ga je veoma lako izvesti, a veoma teško detektovati.

- Low and Slow Attack ili drugim imenom poznat kao slow-rate (spori napad), je DDoS napad koji na prvi pogled izgleda kao potpuno legitiman spori saobraćaj. Slow-rate napad je posebna vrsta napada koju nazivamo "stanje iscrpljenosti DDoS napad" i koji cilja same aplikacije, ali i resurse servera sa kojim komunicira. Stanje iscrpljenosti DDoS napadi ciljaju da obore servise ili serversku infrastrukturu, koja ima za cilj da dostavi razne sadržaje korisnicima, koji te usluge koriste. Napadi koji ovdje spadaju su SYN Flood (Poplava), SSL/TLS Exhaustion (Premor), DNS Query Flooding (Poplava upita) i slično. To su napadi koji izazivaju velika opterećenja na samu veličinu Transmission Control Protocol (Protokol kontrole prenosa) ili TCP mašine stanja, a krajni cilj im je firewall, "Elastic Load Balancer" (Elastični Balanser Opterećenja) ili ELB i slično. Slow-rate napadi su veoma često ciljani ka HTTP-u (zbog njihovog kratkog životnog trajanja), ali znaju često i biti usmjereni ka TCP sesijama (dugog su životnog trajanja). Prilikom mjerenja i posmatranja, kada se tipične akcije odvijaju mnogo duže nego što to inače traje, onda postoji velika šansa da je napad izvršen ili se izvršava. Ovo je lako detektovati, ako ga ciljano provjeravamo. U većini slučajeva i njega je mnogo teško detektovati, jer izgleda kao običan povećani saobraćaj. Ovaj napad je učestaliji, pogotovo sa velikim povećanjem ili proliferacijom pametnih uređaja i lakoćom izvršavanja samog napada. Stvara uslove idealne za izvršenje DDoS napada, korišćenjem velike mreže botova.

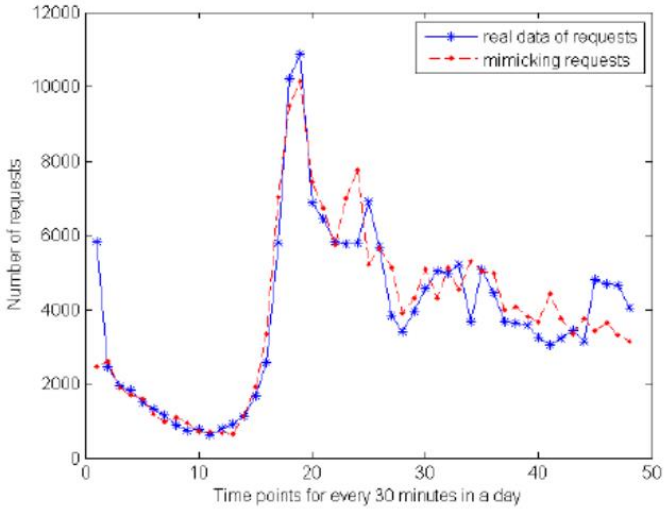
- Large Payload POST je bezbjedonosna prijetnja, odnosno DDoS napad, koji pripada posebnoj vrsti HTTP DDoS napada, gdje napadač ciljano napada XML

kodiranje, koje koriste *web* serveri. *Web* server prima, ili bolje rečeno serveru se šalju strukturisani podaci koji su kodirani u XML formatu. XML kodirani podaci su podaci koji su konvertovani iz "Unicode" karaktera u binarni format, a koje procesor kodira u deklarisani tip kodiranja prema atributu koje sadrže ("encoding"). Server pokušava da dekodira ove kodirane XML podatke, ali pri tome procesu koristi ogromnu ili pretjeranu količinu memorije. Ovo na kraju dovodi do toga da sami sistem padne, a sa njim i sve usluge. Drugi naziv za ovu vrstu DDoS napada je "Oversized Payload Attack", odnosno napad sa prekomjernim teretom. Ova vrsta napada se dešava kada *web* serveri koriste "Document Object Model" (Objektni Model Dokumenta) ili DOM parser (raščlanjivač), koji se koristi da kreira memorijsku reprezentaciju "Simple Object Access Protocol" (Pristupni Protokol Jednostavnog Objekta) ili SOAP poruke.

DOM je hijerarhijski prikaz strukture neke *web* stranice, kako je vidi pretraživač. SOAP omogućava komunikaciju između aplikacija, putem HTTP-a , koje se nalaze na različitim uređajima, sa raznim tehnologijama ili programskim jezicima. SOAP se sastoji od koverte koja identifikuje XML dokument kao SOAP poruku, zaglavlja, tijela i krive koja je skup grešaka. "Extensible Markup Language" ili XML stoji za prošireni jezik za označavanje.

Tu nastaje cijeli problem, jer veličina podataka SOAP poruke može da se duplira, što više sama veličina poruke može biti čak trideset puta veća. Ovo naravno dovodi do opterećenja memorije. Da stvar bude gora, ova vrsta DDoS napada može da sadrži više varijacija. Ovi paketi ili sadržaj ogromne količine može da bude sadržan u zaglavlju SOAP poruke, u tijelu SOAP poruke, kao i u koverti poruke. Ako se nalazi u koverti, to znači da mora da bude sadržan van zaglavlja ili tijela SOAP-a. Razlog zašto je ovaj napad opasan, ali ne i lak za izvesti, je to što napadač mora biti upoznat sa krajnjim tačkama *web* servisa koji cilja, kao i sa pristupom istog. Ako je napadač ispunjava gore navedene uslove, onda je upad veoma jednostavan za izvesti.

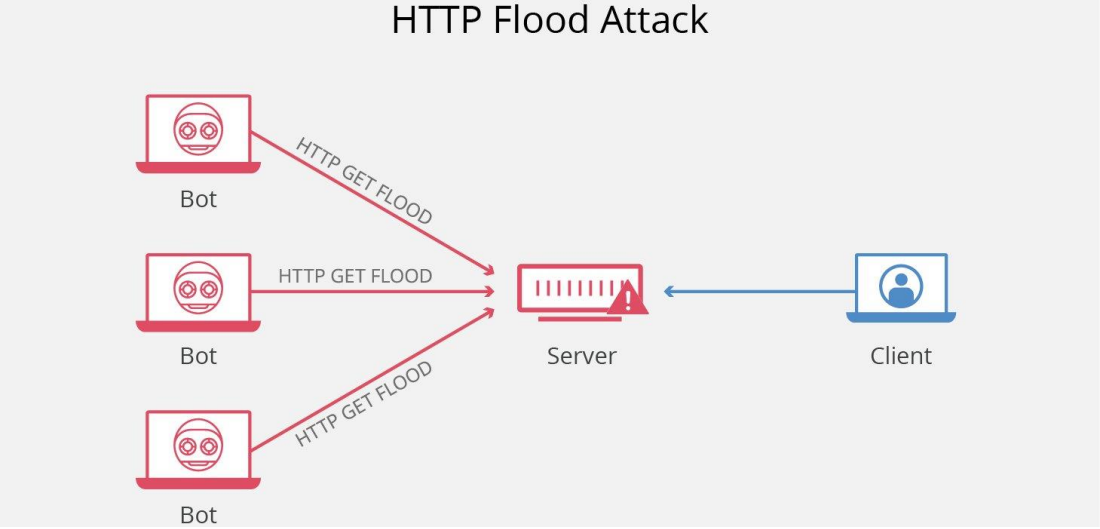
- Mimicked User Browsing je DDoS napad koji koristi mrežu zombija ili mrežnih zombija, koji imitiraju legitimne korisnike, koji pokušavaju da dobiju pristup nekom *web* sajtu. Kao što već znamo ovi mrežni botovi, naravno u velikoj količini će eventualno dovesti do pada tog *web* sajta, ili do toga da saobraćaj neće moći da se odvija. Kao i u svim prethodnim varijacijama ili vrstama DDoS napada, zbog imitiranja pravog saobraćaja, veoma je teško identifikovati da se radi o DDoS napadu. Zbog rasta broja mrežnih botova, čak do toga nivoa, da daleko preovladavaju količinu pravih zahtjeva, sva će komunikacija biti zaustavljana. Postoje razni razlozi za ovakvu vrstu upada, od finansijskih, političkih, testiranja (u cilju saznavanja slabosti), ili bilo koje druge maliciozne namjere. Način na koji se ovaj napad izvršava jeste tako što korisnik kreira bota, kojeg možemo nazvati nulti bot. Ovaj bot pristupa zaraženom uređaju i na njemu instalira drugog bota, koji opet nastavlja proces praveći tako mrežu botova. Host koji izvršava ove napade sadrži komandni i kontrolni (C&C) server ili servere da komunicira sa botovima i skuplja podatke od istih. Da bi izbjegao da bude uhvaćen napadač često mijenja URL svoga C&C servera (nedjeljno). Veličina tih mreža botova može biti čak u milionima. Na grafu 1 možemo vidjeti da je broj legitimnih zahtjeva (plava boja), kao i zahtjeva generisanih od strane mrežnih botova (crvena boja) prema servisu u periodu od deset dana na svakih trideset minuta od kreiranja mreže botova skoro podjednak. Kao što možemo vidjeti podaci su u pojedinim tačkama vremena skoro identični, a u pojedinim dovoljno blizu originalu. Ovo je rezultat istraživanja sa konferencije "Computer Communications Workshops, 2011 IEEE Conference", gdje su autori istraživačkog rada [22] koristili algoritam koji su kreirali, da dokažu da je veoma teško identifikovati mrežu botova, zbog njihove sličnosti sa pravim zahtjevima i generisanju saobraćaja.



Graf 1: Poređenje legitimnih zahtjeva i zahtjeva mrežnih botova [22]

- HTTP(s) Flooding DDoS napad je vrsta napada koja koristi "legitimne" HTTP GET ili HTTP POST zahtjeve, koje koristi za napad na neku aplikaciju. Ova vrsta DDoS napada se dosta često oslanja na grupu kompjutera koji su povezani preko Interneta. Napadač je do ovih uređaja stigao korišćenjem raznih virusa, kao što su malware ili Trojanac poznatiji kao Trojanski konj. Kako funkcionišu HTTP Flooding DDoS napadi? Riječ "Flooding" ili "Flood" se prevodi kao poplavljivanje ili poplava. Prilikom ovih napada server ili aplikacija, koji su u ovom slučaju ciljani, alociraju većinu svojih resursa u pokušaju da odgovore na mnogobrojne zahtjeve upućene ka njima. Zato se koristi riječ poplava, jer napadač pokušava da poplavi ili preplavi ciljani server ili aplikaciju sa što je više moguće zahtjeva koji su veoma intezivni. Što su zahtjevi intezivniji to je bolje. Pošto smo pomenuli da ova vrsta DDoS napada šalje i POST i GET HTTP zahtjeve, razlikuju se načini na koji se oni koriste. HTTP GET zahtjevi se koriste zbog njihove lakoće kreiranja, za mrežu botova koji se oslanjaju na njihovu sposobnost uveličavanja. Nasuprot HTTP GET zahtjevu, HTTP POST zahtjev se više upotrebljava za serversku stranu, jer se pretežno koriste za veoma kompleksno serversko renderovanje, odnosno stvaranje procesa. Obzirom da ne koriste nikakve tehnike refleksije, nikakve

izmijenjene pakete, koriste standardne URL zahtjeve i takođe ne zahtijevaju veliki protok, možemo reći da su teško identifikovani i veoma neprimjetni. Imaju svoju metu, za koju su planski i napravljeni. Komplikovanije HTTP floods tehnike čak pokušavaju da imitiraju ljudsku interakciju sa aplikacijom. Na slici 10 se može vidjeti primjer gdje klijent pokušava da pristupi serveru, ali zbog velikog broja zahtjeva od strane mreže botova koji šalju HTTP GET zahtjeve, server nije responzivan. Sva komunikacija sa serverom u oba pravca je automatski obustavljena, jer je cilj DDoS Flood napada bio da se sruši server. Na primjeru ispod se može vidjeti uspješno izvršen napad.



Slika 10: HTTP GET DDoS Flood (https://www.cloudflare.com/en-gb/learning/ddos/http-flood-ddos-attack/)

DDoS napadi su najčešći vid napada na aplikativnom nivou. Postoje razni načini detekcije i ublažavanja nanijete štete nakon što se izvrši ovakav vid napada, a o njima ćemo više pričati u narednim poglavljima. Nivo štete koja je nanijeta nakon DDoS napada može biti veći ili manji, naravno to zavisi od vida štete koja je nanijeta, da li su podaci trajno oštećeni u napadu, da li je prekinut neki bitni proces legitimnih korisnika, da li je nanijeta veća ili manja šteta i da li je moguće povratiti sve nakon takvog napada. Jedna stvar je sigurna, ova vrsta napada je maliciozna i nanijeta šteta može biti stvarna. Ova vrsta napada će stalno napredovati i zato je veoma neophodno da se prati i konstantno nadograđuju sistemi koji služe za detekciju ili prevenciju protiv istih.

3.2.2.2 SQL Injection

Sledeća vrsta bezbjedonosne prijetnje jeste SQL Injection. SQL Injection je vrsta bezbjedonosne prijetnje koja cilja slabosti "Structured Query Language" (Struktuirani jezik upita) ili SQL-a. SQL je vrsta veoma jednostavnog jezika, koji se koristi da bi se vršile operacije nad relacionim bazama. Relacione baze su kolekcije informacija koje se koriste kako bi se skladištili i organizovali podaci u jednu ili više tabela, odnosno relacija sa kolonama i redovima. Drugim riječima relaciona baza podataka je baza bazirana na relacionom modelu podataka. Izrazi koje koristi SQL jezik su veoma jednostavni (SELECT, INSERT, CREATE, DELETE). Izraz SELECT je najčešće korišćen izraz od svih i on se najviše koristi kada se izvršava SQL Injection. Korišćenjem ovih komandi i njihovim miješanjem mogu se dodati nove tabele u relaciji, što i predstavlja veliku prednost korišćenja relacionih modela, razdvajanje podataka u različite tabele prema određenim uslovima ili atributima, a pritom da ti podaci ostanu povezani.

SQL Injection može da se izvede iz različitih razloga, ali jedna od glavnih klasifikacija izvođenja napada se može vidjeti u tabeli 2.

Tabela 2: Razlozi za SQL Injection [23]

To obtain undisclosed information about the database or its content
To alter the database
To gain privileged access to applications
To limit application functionality

Razlozi za izvršavanje SQL Injection-a prema tabeli iznad mogu biti dobijanje informacija iz baza, izmjena baza, dobijanje administratorskih prava i ograničavanje funkcija aplikacije.

Takođe moramo proći kroz neke od najčešće korišćenih karaktera, koji se koriste za SQL Injection. Apostrof (') ili znaci navoda (") se često koriste, jer riječi, odnosno vrijednosti koje se unose, a ograđeni su njima, predstavljaju tekstualnu vrijednost u samim upitima. Drugi primjer

znaka koji se često koristi jeste tačka-zarez (;) i on označava kraj upita i početak drugog. Takođe može da se koristi i komentar blok (/* */), kod kojeg sav tekst koji se nalazi između dva asteriska, predstavlja komentar. On ima sposobnost da cijeli dio koda napravi "beskorisnim". Poslednji, ali i jedan od najbitnijih je sami asterisk (*), koji ima značenje "SVE". On označava da upit treba da vrati sve instance iz baze ili baza traženih komandom. Česti načini korišćenja SQL Injection komandi, jeste pristup informacijama, na način koji nije bio predvidjen logikom aplikacije. Potom zaobilaženje autentifikacione kapije u samoj aplikaciji, čime se dobija veća kontrola, odnosno sposobnost da se administratorska prava dignu na mnogo veći nivo. Poslednji način obuhvata izvršavanje SQL Injection-a na slijepo, praćenjem ponašanja same aplikacije. Postoje i drugi načini pristupa podacima korišćenjem SQL Injection-a, koji neće biti ovdje navedeni, ali koji su podjednako opasni i naravno ilegalni. U daljem dijelu rada ćemo pojasniti ove načine izvršavanja SQL Injection-a [23].

- Sakupljanje podataka i izvlačenje šeme (UNION upit) – prvi korak svakog napada je planiranje, odnosno sakupljanje informacija, informacija poput konfiguracija sistema, ulaznih tačaka, i slično, u cilju identifikovanja mete. Ovo tehnički ne smatramo da je SQL Injection, ali veoma neophodan korak za svakog napadača koji želi da izvrši SQL Injection, pogotovo sa malim razlikama u DBMS, odnosno "Data Base Management Systems" (Sistemima za upravljanje bazama). Jedan od načina bi bio da za vrijednosti ulaznog polja unesemo string "wrong" u polje za korisničko ime, koji će nam odmah vratiti grešku iz koje možemo saznati da se radi o MySQL bazi na primer, iako dostina kvalitetno urađenih aplikacija neće u logu direktno ispisati poruku greške, vrijedno je bilo probati i možda saznati jednu više informaciju. UNION operator unutar upita u kombinaciji sa SELECT izrazom omogućava konkatenaciju ili spajanje više tabela u jednu veliku. Sa time dobijamo UNION SQL Injection, koji omogućava da pored standardnog SQL upita, koji će vratiti rezultat jedne tabele, izvedemo više SQL upita i vežemo na osnovni.

  SELECT a, b FROM table1 UNION SELECT c, d FROM table2

  Da bi UNION upit funkcionisao, neophodno je da se ispune svi uslovi:

1. Svaki pojedinačni upit mora kao rezultat da ima isti broj kolona;

2. Podaci, odnosno njihovi tipovi moraju biti kompatibilni i to mora da važi za svaki upit;

Da bi UNION SQL Injection funkcionisao, neophodno je da se ispune dva uslova:

1. Mora se saznati broj kolona koji će se vratiti kao odgovor od strane originalnog upita;

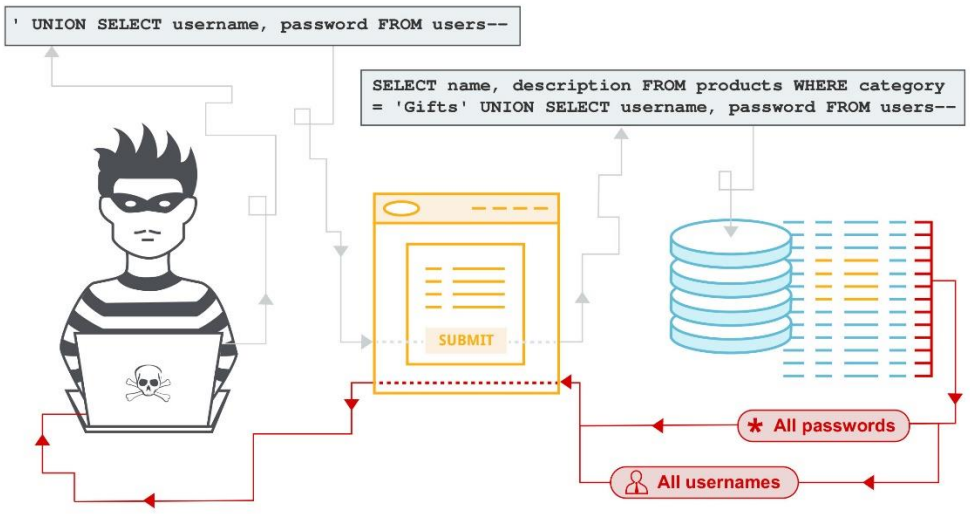
2. Mora se saznati koje kolone, koje se vraćaju kao odogovor na originalni upit su kompatibilne sa tipom podataka, da bi sadržale rezultate dodatih upita;

Da bi saznali broj kolona koji će se vratiti kao odgovor od strane originalnog upita postoji par načina da se to izvrši. Prvi način bi bio korišćenje niza ili serije ORDER BY klauzule ili uslova korišćenjem indeksa kao parametra. Ako broj premaši granicu postavljenu u uslovu vratiće grešku, koja može da bude u HTTP obliku, generička greška ili bilo koji drugi oblik koji će nam omogućiti da saznamo broj kolona.

Drugi način obuhvata takođe seriju, ali UNION operatora sa SELECT izrazom na NULL vrijednost koja se svaki put razlikuje. Ako greška kaže da operator mora da ima isti broj izraza ili neka generička greška, onda smo saznali da se broj kolona ne podudara. Ako bi se broj kolona podudarao, vratile bi se dodatne kolone sa NULL vrijednostima.

Da saznamo koje kolone koje su kompatibilne sa datim tipom podatka i mogu da sadrže rezultate dodatih upita, možemo da izvršimo seriju UNION operatora sa SELECT izrazom i recimo nekim stringom, koji ćemo da stavimo između apostrofa. Na svakom upitu, naravno nakon što odredimo broj kolona, stavićemo operator na različito mjesto, da bi vidjeli da li je svaka kolona kompatibilna sa datim

podatkom. Ako nije javiće se neka generička ili specifična greška, kao na primjer pogrešan tip podataka. Onda ćemo saznati koje još tipove sadrži. Ako nema greške onda to znači da smo otkrili tačan tip, u ovom slučaju string. Na slici 11 se može vidjeti kako izgleda UNION SQL Injection.



Slika 11: UNION SQL Injection (https://kratikal.com/blog/sql-injection-attack-a-major-application-security-threat/)

- Povećanje privilegija i dobijanje pristupa – baze podataka, kao što već znamo, veoma često se koriste za skladištenje privatnih podataka za pristup aplikaciji, kao što su šifra i korisničko ime ili email. Kada neki korisnik pokuša da pristupi sajtu, korišćenjem svojih kredencijala, ti podaci se upoređuju sa podacima koji se nalaze u bazi. Naravno ti podaci se kroz posebne sisteme kriptuju određenom funkcijom za hešovanje. Trenutno se najviše koriste SHA-256 ili SHA-512. Prilikom pristupa vrši se dekripcija i ako se podaci iz baze i novo-unijeti poklapaju, korisniku se omogućava da pristupi svom nalogu. Na ovaj način sistem zna da li korisnik ima pravo pristupa kroz kapiju za autentifikaciju neke aplikacije. Naravno, ovo se sve vrši u sklopu sa tokenima za autentifikaciju, gdje se kreira sesija korisnika koji se uloginuje, ali to je sasvim drugačija tema od ove i više se tiče polja autentifikacije u saradnji sa autorizacijom. Jedan od manje poznatih vrsta potpunog zanemarivanja

autentifikacione forme jeste naravno ako se upit u SQL, koji se koriste za autentifikaciju, oslanja samo na potrebu da bude tipa Boolean, odnosno da bude istinit (true). Tautologija je pojam, ako se radi o Boolean logici, koji je uvijek istinit. Dovoljno je da se stavi OR (ili) izraz u sklopu upita, i stavi bilo koja tačna logika, na primjer 1=1, i to čini cijeli upit tačan. Kod OR izraza dovoljno je da jedna strana bude tačna. Još jedan operator koji ide dobro uz ovu vrstu upita je WHERE. Naravno, za napadača je neophodno da zna i na kojem se DBMS-u baza pokrenula, da bi koristio prave karaktere koji korespondiraju datom modelu ili verziji. Na slici broj 12 se može vidjeti kako to izgleda na primjeru.

Slika 12: SQL Injection tautologija na nivou UI

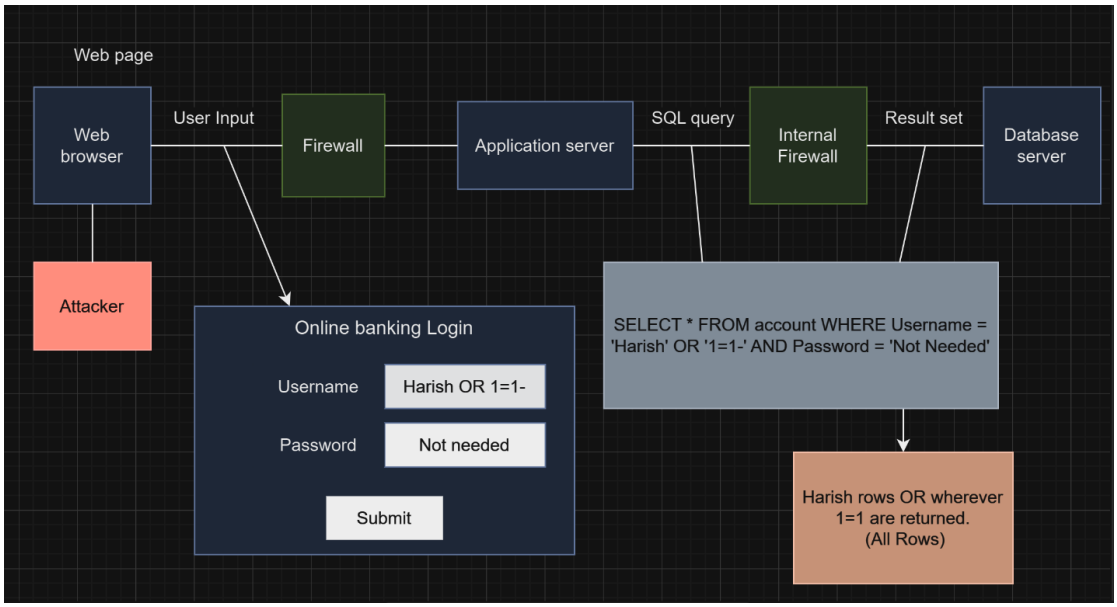
- SQL Injection na slijepo – vrsta SQL Injection-a kod kojeg se napad vrši procesom zaključivanja. Cijeli proces se oslanja na logičko zaključivanje i kroz mnogobrojne pokušaje u nadi da će se saznati bilo kakve informacije. Razlika između ovog načina i načina ”povećanja privilegija i dobijanje pristupa” je što ”povećanja privilegija i dobijanje pristupa” stalno vraće tačan odgovor, dok kod ovog načina mi možemo samo da naslutimo da će da vrati tačan odgovor. Naravno napadač ne zna da li je aplikacija ranjiva na ovaj vid bezbjedonosne prijetnje, zato postoji par načina da sazna te informacije. Jedan od najboljih načina je da se iskoristi zakašnjenje prilikom izvršavanja upita, potencijalno korišćenjem WAITFOR operatora, koji omogućava da se odredi kada će se upit izvršiti (do 24 časa). Jedan od najčešće korišćenih SQL upada na slijepo, jeste korišćenjem logičkih izraza kao i kod prethodnog slučaja SQL Injection-a. Ovo se naziva SQL Injection na slijepo

zasnovan na Boolean-u. Trik je da koristimo oboje tautologiju i kontradikciju. Prvo koristimo tautologiju i stavimo nekik izraz koji će uvijek biti tačan, kao na primjer (‘ OR ‘5=5’ --), a potom neki koji će biti netačan i dodamo AND, kao na primjer (‘ AND ‘3+3’ --). Cilj je da vidimo kako izgledaju odgovori kod oboje tačno i netačnog upita. Na ovaj način koristimo poređenje da saznamo neke od potrebnih informacija. Takođe, korišćenjem SUBSTRING() metode, možemo da saznamo “potencijalne” pozicije karaktera na određenim pozicijama i time rekonstruišemo informacije koje nedostaju. Za ovo se koriste skripte, jer bi cijeli proces trajao nevjerovatno dugo.

Drugi način koji možemo da koristimo je SQL upad na slijepo, zasnovan na vremenu. Ponekad informacije koje napadač dobije procesom zaključivanja nisu dovoljne, ili se ne razlikuju dovoljno, zbog čega funkcije koje ćemo dalje pomenuti se veoma često koriste da bi se ostvarili rezultati, koji se inače teško ostvaruju.

Funkcije koje se često koriste su SLEEP (vrijeme u sekundama), kreira vremensko zakašnjenje od onoliko sekundi koliko mi proslijedimo kao parameter funkcije. Takođe se koriste i funkcije WAITFOR DELAY() i WAITFOR TIME(), ovo naravno zavisi od toga koja se vrsta SQL koristi (MySQL, MSSQL, …). Verifikovanjem vremenske razlike između dostavljanja i izvršavanja upita, napadač može da potvrdi da li funkcije stvarno funkcionišu. Naravno postoji i mogućnost kombinacije procesa zaključivanja i procesa zasnovanog na vremenu.

Poslednja, ali i ne manje bitna metoda u sklopu SQL Injection-a na slijepo jeste dijeljenje i balansiranje. Dva upita mogu biti ista, a da se ne poklapaju potpuno u samom dijelu koda. Kako su dva upita ista, a imaju različitu sintaksu. Recimo da imamo upit gdje ispisujemo ime iz tabele, gdje je ID identičan petici. Drugi način bi bio da se ovo napiše gdje je ID identičan trojci plus dvojci, što takođe daje pet. Ovo se isto tako može raditi i sa stringovima, odnosno stringovi se mogu nadovezivati. Takođe možemo dodati i zagrade gdje možemo da dodamo upit, na primjer pet možemo da dobijemo ‘3 + (SELECT 1+1)’. Tu nastaje cijeli problem, jer se između tih zagrada može naći dio koda koji može da izvrši upad.



Slika 13: SQL Injection tautologija na nivou *web* aplikacije (Dehariya Harish, Shukla Piyush, Ahirwar Manish, "A Survey on Detection and Prevention Techniques for SQL Injection Attacks", 2016.11.08., do: 10.5815/ijwmt.2016.06.08)

Napadač unese pogrešne podatke u korisničko ime, a u našem slučaju tautološke podatke, za koje smo rekli da su uvijek tačni. Unijeto je ime OR slučaj koji je uvijek tačan, u ovom slučaju '1=1'. Potom se ova vrijednost gore izvrši i napadač dobije autorizaciona prava za izvršenje upita. Server od aplikacije proslijedi ovaj upit do baznog servera, a potom bazni server generiše sve rezultate koje bi inače legitimni korisnik dobio za potpuno legitimno pristupanje nalogu. Pristup podacima može biti čak i veći ako je napadač dobio pristup administratorskom nalogu, koji mu omogućava da izvrši elevaciju prava na adminstratorsko.

3.2.2.3 Cross-Site Scripting

Nakon SQL Injection-a, slijedi bezbjedonosna prijetnja na aplikativnom nivou koju nazivamo Cross-Site Scripting ili skraćeno XSS. XSS je vrsta prijetnje kod koga se maliciozne skripte ubacaju na sajtove koji izgledaju vjerodostojno. Ova vrsta upada koristi nedostatke u ulazima korisnika, preko izlaza koje odgovor generiše, bez njihove prethodne validacije. Skripta

koja se šalje korisnikovom pretraživaču će biti izvršena, jer *web* pretraživač nema način da zna da skripta sadrži maliciozni kod. Sada ta skripta može da pristupi kolačićima, tokenima sesije, kao i bilo kojoj drugoj informaciji koju čuva pretraživač, a vezana je za tu *web* aplikaciju. Takođe, bitno je navesti, da su ove skripte u stanju da izvrše kompletnu promjenu neke HTML stranice. Da bi se izvršio XSS napad, moraju da se ispune neki uslovi. Podaci uđu na *web* aplikaciju kroz neki izvor kojem se pretežno ne može vjerovati, tipično kroz neki zahtjev na *web*-u. Ovi podaci se šalju korisniku sa ostalim dinamičkim sadržajem, koji naravno ne provjerava da li sadrži bilo kakav maliciozni sadržaj, može biti u formi segmenta JaveScript-a, a HTML, ili bilo koji drugi kod, može biti dio njega. Neke od čestih akcija koje se izvršavaju sa ovim malicioznim sadržajem mogu biti transfer podataka iz kolačića, sesije, preusmjeravanje na neki drugi sajt koji je u vlasništvu napadača ili bilo koja druga aktivnost koristeći uređaj žrtve.

XSS napad možemo da podijelimo u tri kategorije. XSS napad može biti reflektivan, skladišten i baziran na DOM-u.

Reflektivna vrsta XSS napada je ona koja je reflektovana od *web* servera, to su odgovori servera, kao što je poruka da postoji neka greška, rezultat pretrage, ili bilo koji HTTP GET zahtjev prema serveru, na koji server može da da povratni odgovor. Ovaj XSS napad do žrtve stiže preko mail poruke, ili na neki drugi *web* sajt. Kada korisnik pretražuje neki sajt koji je zaražen, posjećuje sadržaj na koji je preusmjeren ili klikne na neku formu, kod u pozadini koji je postavio napadač, a koji ima za cilj da ukrade podatke, se izvršava koristeći slabosti (na primjer neregulisano ulazno polje), što daje napadaču pristup kolačiću sesije. Ova vrsta XSS napada se takođe zove "ne-uporan" XSS napad. U nastavku se može vidjeti primjer reflektivne vrste XSS napada.

1. Napadač provjerava da li je sajt ranjiv tako što izvrši sledeći upit prilikom autentifikacije u input polje:

<script type='text/javascript'>alert('masterRad');</script>

2. Alert notifikacija prikaže string masterRad.

3. Stranica potom prikaže sledeću grešku:

"<script type='text/javascript'>alert('masterRad');</script > not found."

4. URL link stranice će u datom trenutku da bude:

> http://masterrad.com?q=<script type=”text/javascript”>alert(‘masterRad’);
> </script>

5. Napadač sada zna da je sajt ranjiv i kreiraće svoj URL link koji izvršava maliciozni napad, na primjer:

> http://masterrad.com?q=news<\script%20src=”http://xssbobanbanjevic.com/authstealucg.js”

6. Napadač pošalje ovaj link na neki forum ostalim korisnicima, odnosno posjetiocima i neko od njih klikne.

7. Napadač dobija pristup svim njihovim kolačićima, tj. njihovim sesijama.

Skladišna vrsta XSS napada ili “uporan” XSS napad, je napad koji se trajno skladišti na server mete, na lokacijama kao što su: baze, forumi, komentari i slično. Kada se ti podaci zahtijevaju sa njihovom isporukom stiže i maliciozna skripta. Jedan oblik skladišnog XSS napada je Slijepi XSS. Slijepi XSS je vrsta XSS-a gdje je maliciozna skripta sačuvana na serveru i kada korisnik pokuša da pristupi podacima, na primjer formi, a ona se učita u backend-u aplikacije, maliciozni kod se aktivira.

Poslednji tip XSS-a je onaj baziran na DOM-u, a drugačije se naziva “tip nula”. On se izvršava kao posljedica promjene samog DOM okruženja. Ovo se dešava od strane originalne skripte na klijentskoj strani. Zahtjev se u ovom slučaju ne mijenja, ali se kod na stranici izvršava neočekivano, odnosno drugačije zbog modifikacija u okruženju DOM-a.

3.2.2.4 Cross-Site Request Forgery

Na kraju od bezbjedonosnih prijetnji na aplikativnom nivou koje smo mi smatrali jednim od češćih, ostao je još Cross-Site Request Forgery ili CSRF. CSRF je vrsta bezbjedonosne prijetnje gdje napadač “prevari” korisnika da izvrši neku vrstu akcije, koju nije htio da izvrši, na aplikaciji na kojoj je korisnik autentifikovan. Primjer ovog napada bi bio da napadač pošalje poruku ili link

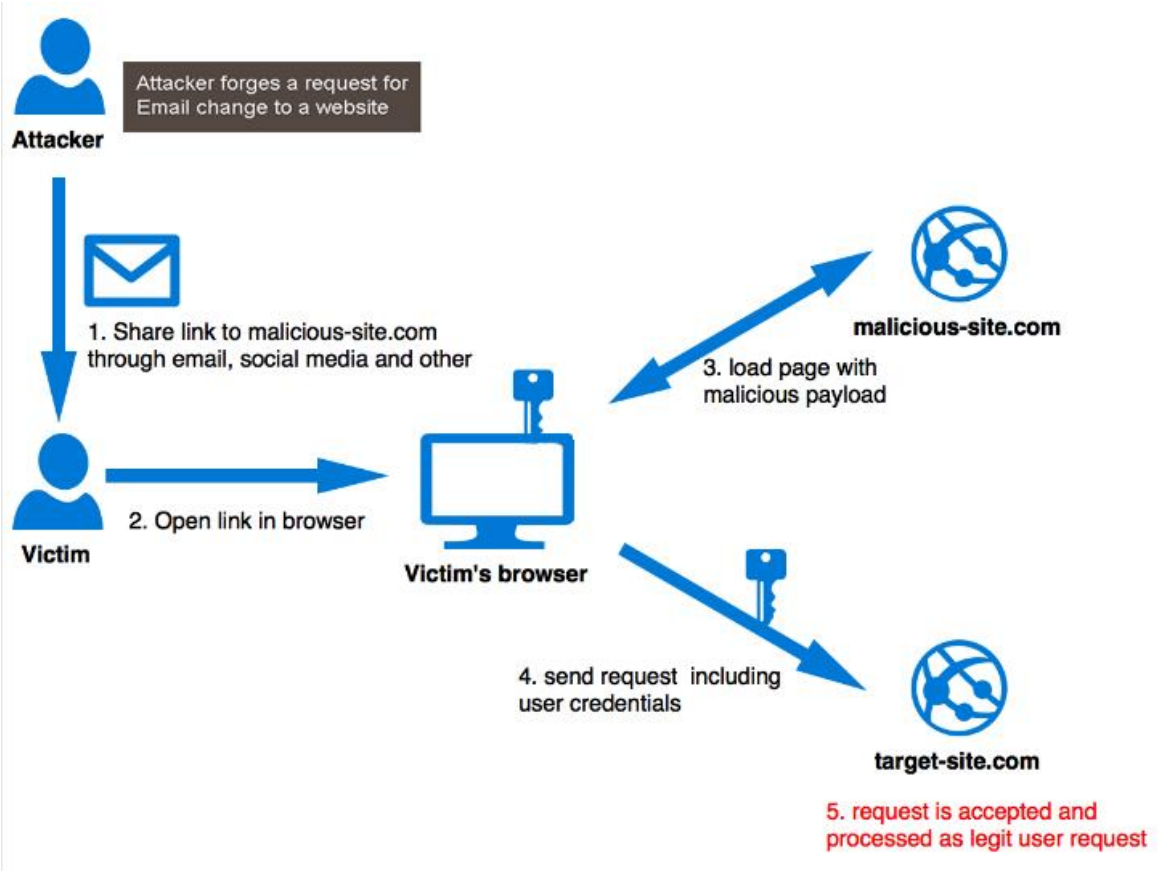
na nekoj aplikaciji ili sajtu, na primjer u poruci. Korisnik koji trenutno ima pristup svom nalogu klikne na tu poruku i on ga odvede na taj eksterni sajt. Ali korisnik nije svjesan da je upravo izbrisao svoj nalog, jer iza te poruke se krio zahtjev koji kaže da se korisnik sa datim ID-jem (identifikacija trenutno autentifikovanog korisnika) izbriše. Svakim klikom se izvršava neki request (zahtjev), a u kodu se ovaj zahtjev izvršio prije nego što je korisnik odveden na novi sajt. Ove akcije mogu biti i prenos sredstava, promjena email-a ako se radi o administratorskom nalogu, cijeli sajt je u opasnosti. Slika broj 14 prikazuje primjer kako ovo tačno funkcioniše.

Posebna vrsta ove bezbjedonosne prijetnje jeste login CSRF. Cilj napadača je da se korisnik koji nije autentifikovan, autentifikuje na nalog koji kontroliše taj napadač, a veliki bonus za napadača je ako korisnik počne da unosi svoje lične podatke, kreditne podatke, ili bilo koje druge privatne podatke. To omogućava napadaču da pristupi tom nalogu sa svim podacima korisnika.

CSRF je napad koji možemo da izvršimo korišćenjem GET ili POST HTTP zahtjeva. Dovoljno je da se prevari korisnik da klikne na neki link, gdje će izvršiti GET zahtjev na neku specifičnu URL adresu. Ovaj link mora da sadrži neki naziv koji će privući pažnju korisniku, a opet da ne bude previse očigledan. Ali zanimljiviji način da se izvrši CSRF napad jeste slanjem POST zahtjeva. Primjer korišćenja POST zahtjeva bi bio korišćenje forme, koju korisnik treba da dostavi. Kada korisnik unese podatke i klikne “Pošalji” dugme, zajedno sa ovim podacima bi se poslali i autorizacioni kolačići, isto kao i kod GET zahtjeva. Sada novi problem koji tu nastaje jeste kako da se prevari korisnik da preda tu formu. Jedna stvar koju treba razjasniti jeste da stranica sa ovom formom ostavlja korisnika autentifikovanog, sa svim njegovim podacima. Kada korisnik posjeti formu, podaci već mogu biti unijeti, pa više nema potrebe da korisnik unosi bilo kakve podatke, sami podaci su već unijeti, podaci kao što su količina novca koja se šalje, računa na koji se šalje i slično. Poslednji problem koji treba da se riješi je kako da se sakrije poruka od korisnika da je određena količina novca poslata, odnosno da je bilo kakva transakcija izvršena. Ovo se može lako postići sakrivanjem HTML iframe-a. Dovoljno bi bilo da postavimo vrijednost “display” atributa na nepostojeću i korisnik ne bi dobijao bilo koju poruku, odnosno obavještenje.

Pored gore navedenih bezbjedonosnih prijetnji na aplikativnom nivou, postoji dosta drugih kao što su parametarski napad (promjena parametara u poljima forme), prisiljeno pretraživanje (generisanje i testiranje – sistematsko pretraživanje svih mogućih kandidata bez obzira da li

zadovoljavaju uslove), otimanje sesije (metoda potajnog preuzimanja sesije korisnika, krađom njegovog ID-ja i pretvaranja da je taj autorizovani korisnik) i mnogi drugi. Svaki od njih predstavlja ozbiljnu prijetnju, neki veću, neki manju. Mi smo prošli kroz neke od najčešćih, a u sledećem poglavlju ćemo proći kroz njihovu identifikaciju, a potom i njihovu prevenciju.



Slika 14: CSRF napad (https://medium.com/@ashifm4/protection-from-cross-site-request-forgery-csrf-9cf4f542e268)

4. Detekcija upada, pre i post detekcija

Prvi korak u prevenciji i otklanjanju bezbjedonosnih prijetnji i početku procesa oporavka od istih jeste identifikacija samih prijetnji.Važno je da se ne samo identifikuje prijetnja, već i da se izvrži evaluacija rizika, odnosno kakvi su gubici i koliki je nivo rizika. Bitno je identifikovati bezbjedonosnu prijetnju i njen izvor, razumjeti samu prijetnju i naravno klasifikovati je. Klasifikacija je veoma bitna, jer pomaže da se lakše identifikuje o kojoj se prijetnji radi. U poglavlju dva smo već prošli kroz klasifikaciju i vrste podjele klasifikacije, tako da se nećemo mnogo obazirati na samu klasifikaciju u ovom poglavlju. Ako odredimo ne samo izvor već i oblast koja je pogođena bezbjedonosnom prijetnjom, moći ćemo da odredimo o kojoj se prijetnji radi.

U ovom poglavlju ćemo proći kroz detekciju pojedinih prijetnji koje smo naveli kao najčešće bezbjedonosne prijetnje, odnosno kroz načine detekcije. Neke je lakše detektovati od drugih, dok je kod nekih sami proces detektovanja teži. Naravno da se ova vrsta detektovanja ne radi često ručno, odnosno koriste se posebni softveri da bi se izvršila detekcija u realnom vremenu i odstranila. Mi nećemo trenutno govoriti o toj vrsti softvera već ćemo analizirati načine na koje to softveri rade, ili načine na koje pojedinci mogu da izvrše isti proces kao softver.

4.1 Detekcija “Distributed denial of service“ napada

Prva od tri bezbjedonosne prijetnje za koje ćemo objasniti kako da se izvrši detekcija jeste DDoS napad. Meta DDoS napada na nivou sedam OSI modela je *web* server. Ovaj napad ima za cilj da ga obori koristeći potencijalno neki od HTTP protokola koji pripadaju nivou sedam. Jedan od najočiglednijih simptoma DDoS napada na neki sajt ili neku aplikaciju jeste kada aplikacija slučajno uspori ili se na primjer obori. Naravno, ako se obori *web* server, to onda znači da je već prekasno, ali je jedan od osnovnih znakova da je DDoS napad izvršen, ali ne i da je završen. Kada kažemo prekasno, mislimo na činjenicu da se sajtu ili aplikaciji ne može pristupiti, jer je *web* server pao, što je i cilj napada, ali cilj napada je da ostane oboren što duže može. Jedini način da se napad detektuje jeste u periodu dok se on još izvršava, a naš cilj je da ga detektujemo i spriječimo prije nego uspije da obori aplikaciju. Ako sajt ili aplikacija nisu aktivni ili su usporeni, to onda ne mora

po automatizmu da znači da je izvršen DDoS napad već takođe može biti neki drugi problem kao što je velika posjećenost (sezonska posjeta sajtovima), akcija ili promocija neke usluge ili proizvoda, neki hardverski problem, ili nešto drugo. Korišćenjem nekog alata za praćenje saobraćaja ka aplikaciji ili serveru možemo da vidimo da li se dešava neki od sledećih problema:

- Možemo da pratimo da li je došlo do povećanja saobraćaja od strane korisnika (posjetioca, odnosno klijenata) koji dijele neku zajedničku osobinu, na primjer zajednički potpis. To mogu da budu isti ili slični pretraživači (verzije pretraživača), zajedničku geolokaciju, isti tip uređaja, i slično;

- Velika količina saobraćaja koja proizilazi sa jedne zajedničke IP adrese, ili sa zajedničkog opsega IP adresa;

- Nagli veliki rast, to jest nagli skok u količini zahtjeva, na jednu od stranica aplikacije ili *web* sajta, koji ne može biti objašnjen;

- Posebni vremenski šabloni ili obrasci. Česta ponavljanja u vremenskim periodima naglog skoka posjeta (saobraćaja), na primjer u određenom vremenu (u 15:00 sati svakog trećeg dana);

Mora se napomenuti da se danas koriste i softverska rješenja koja koriste Artificial Intelligence (vještačku inteligenciju) ili AI i mašinsko učenje, da analiziraju šablone i razlikuju DDoS napade od običnog saobraćaja ka sajtu ili aplikaciji, upoređivanjem sa detaljnom bazom podataka koju sadrže. AI se takođe koristi u raznim poljima kao što su medicina, edukacija, građevina i sve više vojska (navođenje raketa, navođenje tenkova, pilotnih i bespilotnih letjelica, exoskeletoni, cyber zaštita i slično), koji takođe moraju biti zaštićeni od spoljnih faktora aplikativnih napada.

4.2 Detekcija “SQL Injection” napada

Sledeća u nizu detekcija, jeste detekcija SQL Injection bezbjedonosnih prijetnji. Način da se spriječi, odnosno da se detektuje pokušaj izvršavanja SQL Injection napada jeste pomoću IDS-a ili “Intrusion Detection System” (Sistema za detekciju upada). Naravno IDS možemo da koristimo za bilo koju vrstu bezbjedonosne prijetnje. Ali prvo da objasnimo što je to IDS.

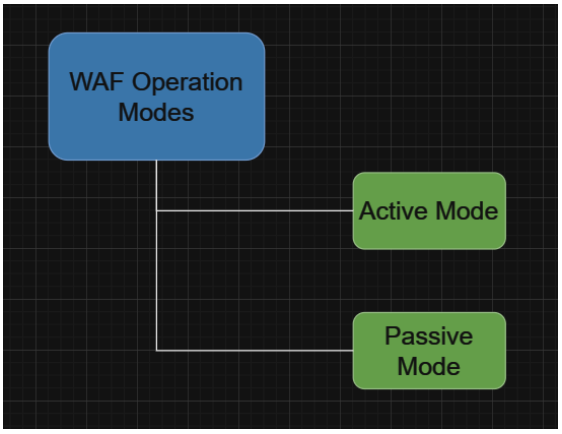
IDS je sistem koji nadgleda mrežni saobraćaj i pokušava da nađe bilo koju vrstu sumnjive aktivnosti i naravno alarmira korisnika i/ili administratora o takvim aktivnostima. Svaka aktivnost koja ima za cilj da izvrši povredu, a bude detektovana, alarmira administraciju, ili se sakuplja korišćenjem sistema za upravljanje zaštitnim informacija i događajima (SIEM). IDS je sistem koji se često koristi, ali “problem” sa IDS sistemom jeste što on proces analize i detekcije obavlja na nivou tri OSI, odnosno mrežnom nivou. Metode koje IDS tipično koristi su metoda “signature” (potpisa) i metodu zasnovanu na abnormalnostima. Za prvu metodu je neophodno upisivanje u bazu svakog potpisa elementa sistema koji šalje poruke i na taj način se zna da li paket pripada elementu. Ovaj proces radi, ali je memorijski veoma zahtjevan i spor. Druga metoda gleda abnormalnosti u mreži i ne zahtijeva čuvanje podataka u bazi, zbog čega je mnogo brža i memorijski efikasnija [24]. Nama je potreban aplikativni nivo ili nivo sedam. Zato se koristi jedan od pet modela IDS sistema, koji je specijalno namijenjen za aplikativni nivo, odnosno “Application Protocol-based Intrusion Detection System” (APIDS) ili sistem za detekciju upada zasnovan na aplikativnim protokolima. Koji protokoli spadaju pod kojim nivoom i koji su protokoli koji nas interesuju, odnosno koji su protokoli pod nivoom sedam OSI tabele, možemo vidjeti na tabeli 3.

APIDS je sistem, odnosno agent koji se nalazi unutar grupe servera. On ima sposobnost da identifikuje neku bezbjedonosnu prijetnju nadgledanjem, to jest interpretacijom komunikacije koja se odvija preko protokola na nivou aplikacije. Što se tiče SQL Injection-a, APIDS bi nadgledao SQL protokole određene middleware-u (posredni softver), odnosno pratio bi komunikaciju sa bazom podataka na *web* serveru i čim bi pronašao, neke sumnjive zahtjeve, obavijestio bi administraciju.

Tabela 3: Protokoli svakog nivoa

Nivo	Protokoli
Aplikativni	NTP, POP, SMTP, SNTP, SSH, NFS, IMAP, HTTP, LDAP, DNS, TELNET, SIP, SDP
Prezentacioni	NCP, LPP, JPEG, MIPI, MPEG
Sesije	ZIP, NETBIOS, SDP, SMPP, RPC, PPTP
Transportni	TCP, UDP, RDP
Mrežni	DDP, EGP, EIGRP, ICMP, IGMP, IPV4, IPV6, IPX
Telekomunikacijski	ATM, CDP, CAN, Ethernet, Frame Relay, HDLC
Fizički	Blueetooth, DSL, ISDN, 802.11 Wi-Fi, 10 BASE-T

Za detekciju SQL Injection-a se koristi *Web* Application Firewall ili WAF o kojem ćemo više govoriti u narednom poglavlju. WAF se koristi kao zaštita od većine bezbjedonosnih prijetnji na nivou sedam OSI tabele, ali postoji mogućnost da se modifikuje, tako da postane jedan vid IDS-a na aplikativnom nivou, i može samo da alarmira administraciju da postoji neki vid bezbjedonosne prijetnje, to jest u našem slučaju neki oblik promjena ili nasilnog pokušaja pristupa bazi podataka. Kako tačno to funkcioniše? WAF treba postaviti na pasivni mod (Slika 15), što mu omogućava da nadgleda zahtjeve aplikacije i ako naiđe na bilo kakav neobični, sumnjivi zahtjev, da automatski obavijesti administraciju. To propratno omogućava administratorima da djeluju blagovremeno prema notifikacijama (obavještenjima) koje su pristigle.



Slika 15: WAF modovi

4.3 Detekcija “Cross-Site Request Forgery“ napada

Poslednja bezbjedonosna prijetnja koju trebamo detektovati je Cross-Site Request Forgery ili CSRF. CSRF je veo ma teško detektovati, zato je bolji način samo testiranja aplikacije i to na jedan od dva načina. Možemo ih testirati ručno i korišćenjem neke od automatizovanih alatki za CSRF testiranje. Neke od tih alatki mogu biti Bright, OWASP ZAP, CSRF Tester i mnoge druge. Nas ovdje samo interesuje ručno testiranje ove vrste bezbjedonosne prijetnje i u temu automatizovanih alatki ćemo zalaziti vrlo malo. Moramo napomenuti da ako korisnici baš žele da izvrše detekciju i ne samo da testiraju aplikaciju ili sajt, to mogu da urade korišćenjem automatizovanih skenera (softvera koji su namijeni za to) ili korišćenjem tokena namijenjenih za zaštitu (Cross-Site Forgery Prevention Cheat Sheet).

Ručno testiranje se vrši tako što za početak moramo da saznamo da li je trenutna sesija sigurna. Ako je informacija za upravljanjem sesijom dostupna na strani klijenta, odnosno korisnika u pretraživaču, to onda znači da je cijela aplikacija ranjiva, a već znamo da pomoću HTTP GET ili HTTP POST zahtjeva možemo da pristupimo svim vidljivim nezaštićenim podacima na klijentskoj strani. Ako se podaci o sesiji čuvaju u kolačićima, onda će korisnik definitivno biti ranjiv. Podatke o sesiji treba čuvati u session storage (skladište sesija), jer traju koliko i pretraživač, za razliku od ostalih.

Takođe treba koristiti dodatnu zaštitu prilikom dizajniranja i kreiranja aplikacija ili sajtova, kao što su anti-CSRF tokeni, da se isti dodaju na svaku formu ili bilo koju akciju koja može da promijeni stanja. Tokeni su unikatni i nepredvidljivi, skoro je nemoguće za napadača da ih otkrije. Svaki dalji zahtjev koji korisnik kreira mora da sadrži date tokene, jer u protivnom će zahtjev biti odbijen. Korisniku se prilikom slanja zahtjeva šalju tokeni, odnosno par tokena (jedan je skriven) od strane servera u JSON ili HTML formatu odgovora, koje je najbolje skladišti u sesiji. Prilikom slanja forme, server će da provjeri ta dva tokena i da sazna da li se slažu prema kriptografskom mehanizmu. Ako se slažu zahtjev je prihvaćen u protivnom je odbijen i javiće se greška.

Još jedan od načina je automatsko slanje jednog nasumičnog tokena u kolačić klijentu, čiji će kod da pročita kolačić i u svaki zahtjev da doda ručno zaglavlje koje će da sadrži dati token iz kolačića. Naravno postoje i drugi načini, ali ovo su neki od najčešćih.

5. Prevencija upada, pre i post prevencija

Do sada smo opisali razne bezbjedonosne prijetnje, koje mogu da se jave na aplikativnom nivou. Prošli smo kroz klasifikacije bezbjedonosnih prijetnji, njihovu taksonomiju, potom smo pogledali načine na koje je moguće detektovati samu prijetnju, i na koji način možemo testirati aplikaciju na pojedine ranjivosti.

Sada je na redu da nakon detektovanja prijetnje vidimo kako je moguće izvršiti prevenciju bezbjedonosnih prijetnji, odnosno kako je moguće pripremiti se za predstojeće potencijalne napade i kako da ublažimo posljedice uspješno izvršenog napada.

5.1 Firewall logika

Jedan od načina sprečavanja bezbjedonosnih prijetnji, koji se široko koristi, jeste firewall logika na aplikativnom nivou. Firewall logika na aplikativnom nivou radi kao obični firewall na principu filtriranja zahtjeva, gdje zahtjevi koji izgledaju kao prijetnja po neku aplikacijsku komponentu ne mogu da prođu dalje. Prvi firewall koji ćemo analizirati na aplikativnom nivou je *Web* Application Firewall ili WAF skraćeno, a potom firewall baze podataka.

WAF je vrsta firewall-a, često u vidu softvera ili u sklopu nekog uređaja, koja ima za cilj da zaštiti *web* aplikaciju. WAF zasnovan na softveru je vid WAF-a koji se nalazi ugrađen u aplikaciji ili serveru i on ne mijenja infrastrukturu *web*-a koji okružuje aplikaciju. Drugi način je korišćenje WAF u sklopu nekog uređaja. On jeste bolji u vidu da ne alocira prostor na *web* serveru i time ne okupira njegove resurse. Zbog toga je bolji u pogledu funkcionalnosti i performansi. Ali on je takođe kompleksniji za implementirati i suprotno od WAF-a zasnovanog na softveru on se pretežno koristi jer se može mijenjati *web* infrastruktura koja opkružuje aplikaciju. Prije smo naveli da se kod firewall-a na aplikativnom nivou koriste filteri koji kontrolišu koji zahtjevi mogu proći, a koji ne. Postoje različiti načini implementacije filtera, odnosno postoje različiti *web* filteri:

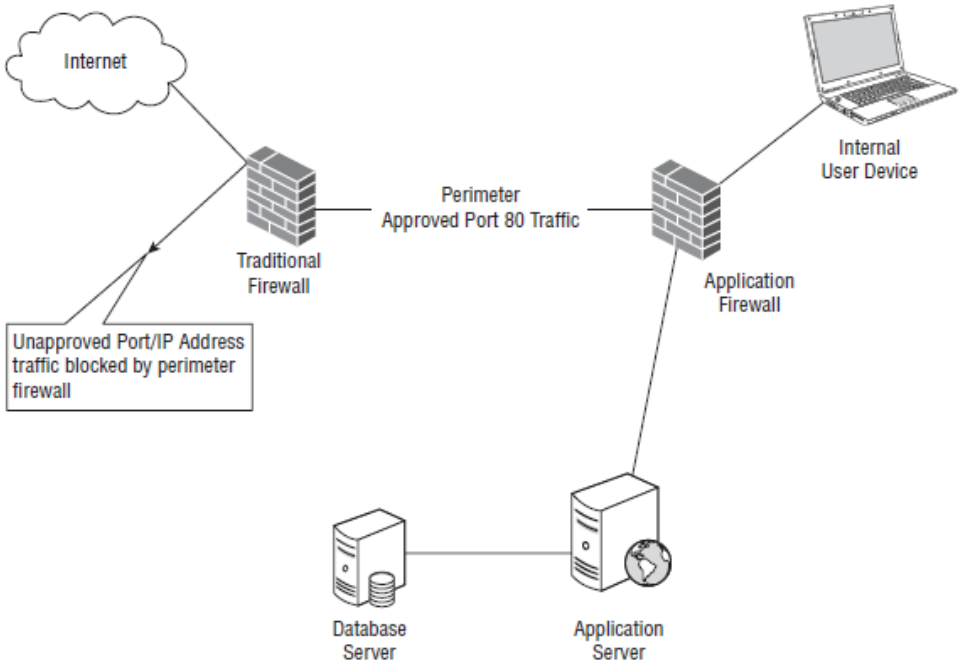
- Filteri *web* servera – je vrsta filtera koja se instalira naknadno, odnosno kao posebna odvojena komponenta, a ima za cilj da procijeni koje komponente treba i mogu da prođu do *web* servera. Funkcioniše na principu prihvaćanja informacija i njihove

validacije. Ako je informacija validna propušta je do *web* servera, u protivnom je odbija.

- Aplikativni filter – je vrsta filtera koja takođe kao filteri *web* servera mogu da se implementiraju kao zasebne komponente, Oni se implementiraju u istom programskom jeziku kao ostatak aplikacije, a zbog njihove modularnosti, često se koriste od strane developera. Oni takođe mogu da se dodaju na lakši način i kao dio koda u vidu regex pravila (regularnih izraza). Ova pravila se pišu na način na koji imaju sposobnost kontrole podataka koji dolaze u aplikaciju. Ovaj tip kontrole može biti strog koliko je developerima potrebno, do najmanjih tipova podataka, kontrole integera (cijeli brojevi), stringova (tekstovi) i tako dalje. Naravno i ovaj način zaštite nije perfektan, jer često sadrži neke propuste i zato se koriste razne gotove biblioteke napravljene od strane velikih timova. Naravno opcija dodavanja modula, koju smo gore naveli takođe je jedna od opcija koja se mnogo često, čak i više koristi.

- Filteri *web* servisa - šta su *web* servisi tačno? Postoje različite definicije *web* servisa, kao i različita značenja istog tog pojma. Ali definicija koja je nama potrebna jeste, *web* servisi su softveri dostupni preko interneta i koji koriste standardizovane XML sisteme, odnosno sisteme za komunikaciju i sve XML odgovore. Filteri za *web* servise su filteri koji služe za kontrolu tih servisnih *web* poruka. Ovo se postiže filtriranjem poruka koje stižu, kao što su SQL Injection upadi (pokušaji).

Naravno WAF može koristiti i druge modove koje smo naveli ili se može koristiti na druge načine. Primjer bi bio pasivni mod koji samo vrši notifikaciju. Mora se navesti da se WAF ne koristi sam već u kombinaciji sa mrežnim firewall-om, što se može vidjeti na slici 16.



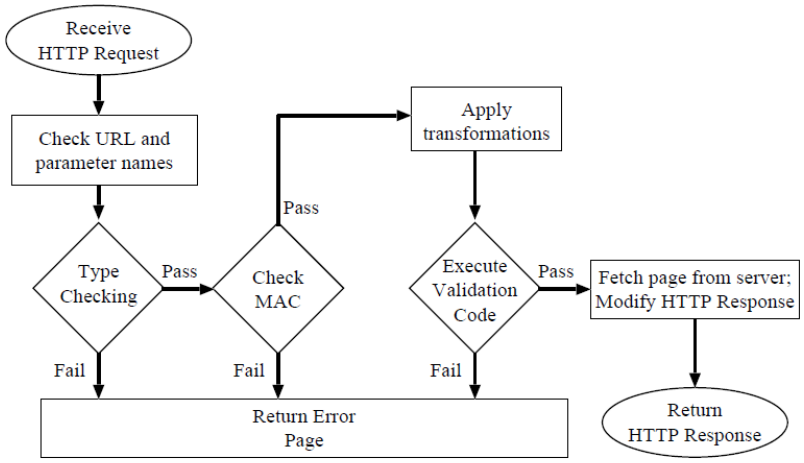
Slika 16: WAP u korelacija sa mrežnim firewall-om [11]

Sada ćemo da govorimo o drugoj vrsti firewall-a, odnosno firewall namijenjen za bazu podataka koji se naziva firewall baze podataka. Firewall baze podataka služi kao posrednik, a lociran je između aplikacije i baze podataka. Njegov posao je da posmatra koji se sve upiti šalju prema bazi podataka. Tradicionalno slanje upita do baze se obavlja na direktnoj osnovi aplikacija – baza, dok korišćenjem firewall-a baze podataka upiti se sada šalju do samog firewall-a, umjesto do baze podataka. Kada upit stigne, firewall provjerava da li postoji kakav sumnjiv ili neobičan upit, kao na primjer tautologija ili neki ilegalni karakter. Kada analizira upit, firewall onda vrši procjenu da li upit može da se šalje dalje do baze ili se kompletno odbija. Korišćenje firewall-a baze podataka nije dovoljno da se baza zaštiti, odnosno aplikacija, neophodno je takođe pored hašovanja šifri i primjena kriptografije na podatke skladištene na aplikaciji. To označava da ni jedan podatak, koji je na primjer kopiran, nema mogućnost direktnog pristupa. Kriptografija podatka označava da je isti šifrovan ili očuvan i da mu se mora direktno pristupiti poznavanjem vrste kodiranja koja je korišćena, što propratno znači da bi se kod provalio i podatku pristupilo treba velika količina vremena, kao i velika komputaciona moć. Ovo se odnosi i na starije kriptografske algoritme, kao što je “Data Encryption Standard” (Standard enkripcije podataka) ili

DES, isto tako i na modernije algoritme kao što je “Advanced Encryption Standard” (Standard napredne enkripcije) ili AES. Još jedna stvar koja ide dobro u kombinaciji sa svime što smo do sada naveli jeste DAM ili “Database Activity Monitoring“ (Nadgledanje aktivnosti baze podataka). DAM je rješenje koje je slično WAF-u i vrši nadgledanje baze podataka, a takođe ima mogućnost alarmiranja u slučaju malicioznih aktivnosti, kao i mogućnost njihovih uklanjanja.

5.2 Security Gateway

Sledeća vrsta prevencije bezbjedonosne prijetnje je korišćenjem Security Gateway-a. Security Gateway ima za cilj sprječavanje pristupa linkovima koji nisu sadržani u gateway-u ili pristup sajtovima koji mogu biti zaraženi. On se pretežno koristi na nivou kompanije, odnosno kada se pristupa eksternoj mreži, to jest aplikacijama i sajtovima u spoljnoj mreži. Na slici 17 možemo vidjeti kako tačno funkcioniše Security Gateway, odnosno koji se sve procesi odvijaju u njemu.



Slika 17: Security Gateway [25]

Kada se primi neki HTTP zahtjev Security Gateway mora da izdvoji zaglavlje, a iz zaglavlja URL da bi izabrao potrebna pravila za validaciju i naravno primijenio sve promjene. Kao što smo gore naveli, ako se izdvojeni URL ne slaže ni sa jednim drugim linkom u zaštitnoj polisi, taj zahtjev se neće dalje poslati prema serveru (javiće se greška), ili u protivnom će biti proslijeđen,

naravno poslije par dodatnih koraka provjere. Razlog zašto je URL, odnosno zahtjev odbijen, jeste što je zaštitna polisa sinhronizovana sa aplikacijom, što je naravno čuva sigurnom. Ako dati URL nema propratnu "Security Policy Definition Language" (Definicioni jezik polise osiguranja) ili SPDL definiciju, to onda znači da neće biti prihvaćena kao legitimna, što znači da će cijeli zahtjev biti odbačen kao nelegitiman, ili maliciozan i poruka sa greškom će biti poslata korisniku. Nakon što je URL potvrđen kao validan, to jest da sadrži potrebne SPDL definicije koje su korespondentne, sledeći korak je da se provjere imena svih parametara i kolačića koji su proslijeđeni u tom zahtjevu. U ovom slučaju zahtjev može biti odbijen ako neki od parametara nedostaje ili se ne nalazi u SDPL polisi, ili ako neki kolačići nisu deklarisani u SDPL-u. Ako makar jedan od gore navedenih uslova nije ispunjen, generisaće se greška koja se vraća korisniku, a zahtjev biva odbijen. Sledeći korak u nizu jeste validacija tipa, odnosno provjera tipa i dužine. Korak nakon toga jeste provjera MAC-a koju ćemo objasniti u nastavku. Ako su svi uslovi ispunjeni, dalje se primjenjuju promjene i izvršava validacioni kod. To onda označava da je zahtjev uspješno validisan i šalje se dalje do *web* servera. Sada *web* server kao odgovor vraća traženu stranicu.

MAC stoji za "Message Authentication Codes" (Kodovi autentifikacije poruka) i označava vid autentifikacije URL-a. Ovo je vid koda koji generiše Security Gateway i SPDL naredjuje da ga pojedini URL moraju sadržati, odnosno da moraju sadržati podatke koje imaju MAC. Služe da spriječe korisnike da vrše bilo kakve promjene nad podacima na strani klijenta. A sami podaci se provjeravaju da li sadrže MAC kada se šalju klijetnu, koje je generisao Security Gateway prilikom slanja zahtjeva *web* serveru. Još jedan korak koji nismo provjerili je izvršenje validacionog koda. Svaka HTML forma sadrži JavaScript kod koji se provjerava na strani klijenta, a koju je tu dodao Security Gateway. Taj se kod dodaje na način što Security Gateway provjerava formu, to jest destinaciju URL forme, zatim da li se podaci šalju kao GET ili POST zahtjev, kreira se JavaScript validacioni program, koji se dodaje kao akcija na "Submit" dugme.

Sve ovo što smo do sada pomenuli je veoma bitno, jer se komunikacija između klijentske strane aplikacije i *web* servera odvija dvostrano. Kada su svi podaci provjereni i poslati, stigli do *web* servera, server mora da pošalje odgovor pravom korisniku, koji ima već validisan zahtjev.

5.3 Runtime Application Self-Protection

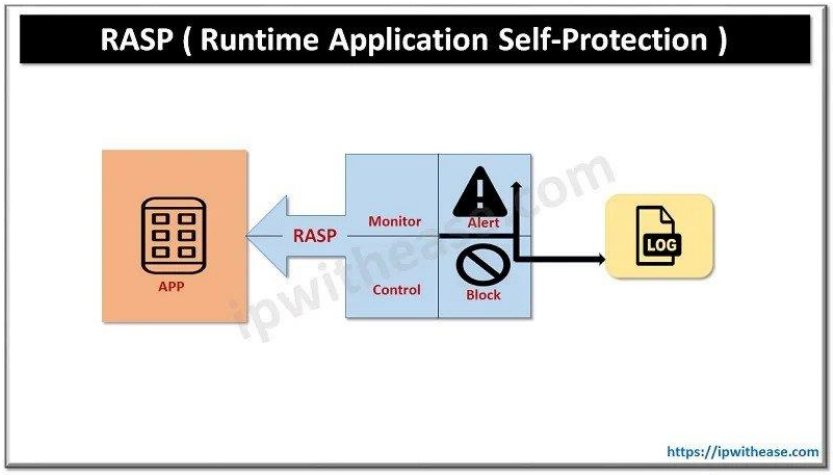
Sledeći u nizu je prevencija korišćenjem Runtime Application Self-Protection ili RASP (Slika 18), pretežno nove tehnologije. Šta je RASP tačno? RASP je tehnologija za aplikativnu zaštitu, koja je dio aplikacije, to jest koja je ugrađena u aplikaciju ili njeno okruženje. Ova tehnologija otkriva bezbjedonosne prijetnje na aplikativnom nivou. Sposobnosti koje RASP tipično sadrži jesu:

- Otvaranje zahtjeva i njihovo pregledanje u samoj aplikaciji;
- Nadgledanje i odbijanje zahjteva aplikacija, ako sadrže prijetnju;
- Funkcionalnost kroz REST API;
- Zaštita od svih klasa napada na aplikativnom nivou;
- Mogućnost da tačno odredi gdje se nalazi slabost koja bi mogla biti iskorišćena, čak i preciznost do linije koda u teoriji;

Prednost koju RASP nudi u odnosu na standardne načine i tehnologije koje služe za prevenciju bezbjedonosnih prijetnji, jeste što omogućava više načina detekcije, sa dosta velikom preciznošću i mnogo lakšim načinom primjene i upotrebe. A velika je prednost što je takođe kompletno integrisan sa aplikacijom čiji je dio [26]. Kako RASP tačno vrši detekciju i prevenciju? Pa za razliku od većine drugih tehnologija koje su zasnovane na detekcije na osnovu potpisa (neki vid identifikatora, kao što je string koda ili haš, koji se veže za neki malware), RASP je zasnovan na dekodiranju parametara i spoljnim referencama, maps funkcijama aplikacija i slično. Omogućava mnogo precizniju detekciju, kao i dodatnu optimizaciju i poboljšanje u performansama. RASP koristi strukturnu analizu i potpuno razumije bilo koji framework u kojem je implementiran, sve sa spoljnim bibliotekama i automatski detektuje kod koji je istekao (ako postoji novija verzija koda). Blokiranje se vrši kada je neka bezbjedonosna prijetnja detektovana, RASP izaziva da se javi aplikacijska greška, koja stopira da zahtjev dođe do servera, ali zbog

potencijalnih problema sa serverom, postoji mogućnost podešavanja odgovora koji vraće RASP. RASP može da funkcioniše u nekoliko modova, ali može da zavisi od dobavljača:

- Isključeni mod – RASP ne funkcioniše, odnosno svaki zahtjev se propušta;
- Mod nadgledanja – RASP nadgleda sve zahtjeve, dostavlja upozorenja i izvještaje, ali ne pokušava da ih spriječi;
- Blok mod – RASP blokira svaki zahtjev koji nema propratnu autorizaciju;
- Blok perimetarski mod – Slično WAF-u, blokira svaki zahtjev koji ne prati određene principe i pravila, koje mi sami predefinišemo;



Slika 18: Runtime Application Self-Protection (https://ipwithease.com/what-is-runtime-application-self-protection-rasp/)

5.4 Web Application & API Protection

Sledeća vrsta zaštite od bezbjedonosnih prijetnji jeste "*Web* Application and API Protection" ili WAAP. Ova vrsta tehnologije nudi razne zaštitne module i pretežno se koristi za cloud zaštitu od bezbjedonosnih prijetnji. Neke od glavnih mogućnosti WAAP tehnologije je uklanjane botova, WAF, zaštita programskih pristupa aplikacijama, zaštita od DDoS napada. Cilj

WAAP-a je da pronađe bezbjedonosnu prijetnju prije nego ona stigne do krajne tačke API. WAAP smatramo unaprijeđenim WAF-om, ali za razliku od WAF-a koristi vještačku inteligenciju i analizu ponašanja za blokiranje napada. Sprječava DDoS napade, ograničenje saobraćaja, zaštitu krajnjih tačaka i samih korisničkih naloga. Naravno neki od nedostataka su što nema zaštite URL-a i forme, potpisa kolačića i CSRF tokena, što može da dodatno oteža rad u zaštiti Cloud-a. WAAP se nalazi na spoljnoj ivici mreže, ispred javne strane *web*-a, gdje može da analizira gužvu, a uprkos postojećim problemima i dalje nudi više u odnosu na WAF koji polako počinje da zastarijeva, sa razvojem novih tehnologija.

5.5 Load Balancer & Reverse Proxy

O load balancer-u i reverse proxy-u smo rekli dosta u poglavlju dva. Svaki load balancer na nivou sedam OSI funkcioniše kao reverse proxy, ali to ne znači da je svaki reverse proxy load balancer. To znači da load balancer ima samo jednu ulogu, da preusmjerava zahtjeve ka određenom broju servera. Međutim, reverse proxy može da bude load balancer, ako se dobro konfiguriše, ali ne mora da znači da treba da vrši tu ulogu. Oni omogućavaju brži odgovor od servera, korišćenjem raznih tehnika ubrzanja (Acceleration):

1. Caching – Kada *web* server šalje odgovor klijentu, reverse proxy čuva kopiju tog odgovora lokalno, tako da kad klijent sledeći put pošalje isti zahtjev reverse proxy ne mora da prosleđuje zahtjev do servera, već automatski šalje odogovor klijentu;

2. Compression (Kompresiju) – Reverse proxy vrši kompresiju raznim algoritmima za kompresiju, u cilju smanjenja protoka neophodnog da bi server poslao odgovor;

3. SSL/TLS Offloading (Rasterećivanje) – Sposobnost da izvrši SSL enkripciju i autentifikaciju na sve zahtjeve/odgovore;

Svaki vid zaštite koji smo do sada prošli ima svoje prednosti i mane, ali je takođe efektivan u nekom polju. Sa razvojem napada razvijaće se i nove tehnologije, ali ove koje smo do sada prošli su i dalje najkorišćenije.

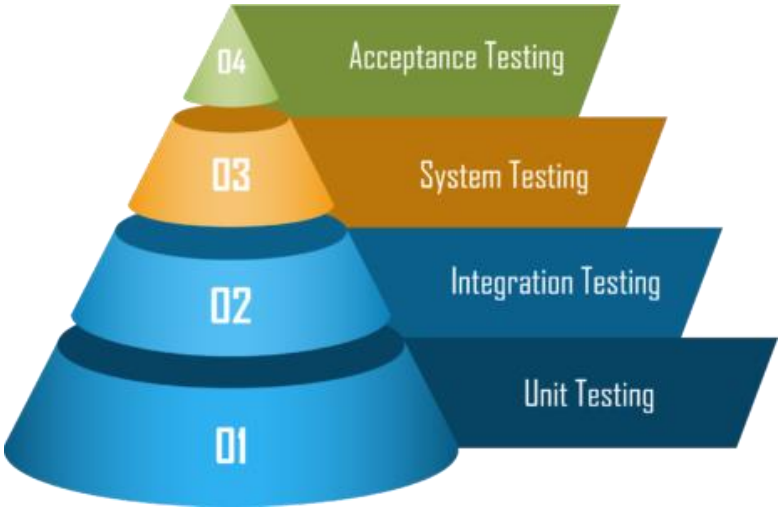
5.6 Testiranje

Još jedan od načina da se aplikacija zaštiti od upada i to preventivno, jeste da se testiraju svi njeni aspekti. Dva pojma koja moramo razlikovat u ovom polju jesu “bijeli šeširi” i “crni šeširi”. Bijeli šeširi su “hakeri” koji pokušavaju da hakuju aplikacije u cilju nalaženja slabosti i njihovog iskorijenjivanja. To su najčešće programeri koji su unajmljeni od strane neke kompanije da testiraju njihove aplikacije. Za razliku od bijelih šešira, crni šeširi su hakeri koji takođe pokušavaju da hakuju neku aplikaciju, ali sa razlogom krađe podataka ili nanošenja štete. Oni rade u ilegali i često ih je teško pratiti, jer sakrivaju svoje tragove. Oni koriste neki od bezbjedonosnih prijetnji gore navedenih i glavni su neprijatelji bijelim šeširima. Pojam “penetration testing” označava testiranje aplikacija u cilju nalaženja neke slabosti i pokušaja da se ta slabost iskorijeni. Pored svega toga prednosti koje ovaj vid prevencije donosi jeste smanjenje rizika od pojave bezbjedonosnih prijetnji, smanjenje legalnih i finansijskih odgovornosti, potvrda sigurnosti i slično. Postoji par metodologija penetracionog testiranja, bijela kutija, siva kutija i crna kutija. Bijela kutija je najskuplja i vremenski duga, ali i najočiglednija u vidu prijetnji, Crna kutija je najbliža originalnu, u situacijama gdje napadač nema potpuni pristup, odnosno radi na slijepo. Siva kutija je nešto između ove dvije. Penetraciono testiranje se sastoji iz četri sekcije [27]:

- Izviđanje – istražuje se koje se tehnologije koriste, infrastrukturu, konfiguracije i slično;
- Mapiranje – kreira se mapa stranica aplikacije i funkcionalnosti, komponente i veze između istih;
- Ranjivosti – u ovoj se fazi otkrivaju ranjivosti;
- Iskorišćavanje – poslednji korak je iskorišćavanje ranjivosti i ponavljanje koraka od početka;

Najsigurniji način sprečavanja bezbjedonosnih prijetnji, prije nego se i izvrše, jeste konstantno testiranje aplikacije i njene zaštite (Slika 19). Treba osmisliti dobar test, koji će proći kroz sve oblasti zaštite, što dodatno znači da treba biti kreativan, treba se staviti u poziciji napadača. Ali vrlo često se desi da prilikom testiranja aplikacija koje smo sami napravili, promaknu neki od očiglednih problema. Zato je dobro da se testiranje vrši i van kompanije. Treba definisati što je jasan cilj testiranja, preći detaljno preko rezultata istraživanja, treba otkriti da li kod radi stvari koje ne bi trebalo da radi. Test treba osmisliti sa načinom razmišljanja da greške

postoje i da će se naći. Ovo su samo neki od principa kojima se treba voditi prilikom osmišljavanja i kreiranja dobrog testa [28].



Slika 19: Testiranje (https://www.edureka.co/blog/software-testing-levels/)

U slučaju da je napad već izvršen, potrebno je odraditi određene korake kako bi se na vrijeme uočili rizici i propratno izvršio oporavak od napada. Potrebno je sklopiti tim za oporavak od napada koji bi se sastojao od članova svih sektora, kao i pojedinih rukovodećih lica. Potom je neophodno da se izvrši procjena rizika prema nekoj od postojećih metoda. Intersekcija između resursa, prijetnji, ranjivosti i vjerovatnoće da će se napad izvršiti predstavljaju rizik ($R = A \times T \times V \times P$) i koriste se često za povećanje vrijednosti biznisa [29]. Sledeći korak je prioritizacija procesa i operacija, odnosno koje djelove sisteme treba prvo oporaviti (koji sistem treba prvo podići, koje oblasti sistema treba dodatno osigurati i slično). Naravno prioritet imaju oni sistemi koje nazivamo kritičnim, odnosno oni bez kojih određeni sektori ne mogu da funkcionišu. Zatim treba odlučiti o strategiji oporavka, sakupljanja podataka (popis inventara aplikacija, softvera, dokumenata, komunikacionih veza, …), kreiranja plana oporavka i njegovog testiranja i na kraju odobrenje plana od samog vrha kompanije [30]. Ovi koraci se mogu vidjeti na slici 20.

Slika 20: Disaster recovery planning [30]

5.7 Softverska rješenja za detekciju i prevenciju prijetnji (komercijalna i open source)

Postoje razni softveri ili alati koje mogu da se koriste za detekciju bezbjedonosnih prijetnji na aplikativnom nivou. Sve oblasti detekcije ili zaštite koje smo prešli do sada čine te alate, ili se makar sadrže u njima.

Jedna od definicija softvera za detekciju bezbjedonosnih prijetnji je da je to softverski program ili uređaj koji pojedinci ili kompanije koriste da bi zaštitili svoje sisteme od neovlašćenog upada i pristupa informacijama. Mogu da se koriste samostalno, ali češće se koriste u kombinaciji sa drugim softverima iste ili slične namjene ili skriptama koje omogućavaju veću kontrolu nad nekom željenom akcijom. Kako smo do sada govorili o raznim oblicima zaštite, u ovom poglavlju ćemo nabrojati neke od najčešće korišćenih softvera za detekciju i prevenciju od bezbjedonosnih prijetnji.

Prvi softveri će biti WAF i među njima ćemo pomenuti softvere komercijalnog tipa kao što su AWS WAF, Azure WAF, Imperva WAF, Cloudflare WAF i mnogi drugi. Ovo su jedni od najpoznatijih brendova na tržištu, ali i najviše korišćenih. A njihov kvalitet podržava i ogroman broj korisnika. Ako gledamo i na to da većina komercijalnih WAF softvera, pored mjesečne ili godišnje pretplate, sadrži i “pay as you go“ (plati po upotrebi) model, cijena može da varira drastično, ali to ne znači da tu cijenu ne opravdavaju sa dodatnim, propratnim sadržajem koji potencijalno imaju u odnosu na svoje kompetitore.

Pored komercijalnih WAF softvera, možemo koristiti i open source WAF softvere. Neki od najčešće korišćenih su NAXSI, *Web*Knight, Coraza, OctopusWAF, ModSecurity i slično. Open source softveri pružaju prednost u pogledu cijena, promjene postojećeg rješenja, stalne dorade i doprinosa od strane zajednice, kao i njegov testiranja i pronalaženja bug-ova. Takođe pružaju podršku laganog početka, koji eventualno može da preraste u korišćenje velikih softvera komercijalnog tipa koje dolazi sa rastom same kompanije. Ali to ne znači da korišćenje open source softvera ne dolazi sa svojim poteškoćama kao što su napadi na framework-ove koji se inkorporiraju u WAF-u, zatim kompletan zaobilazak WAF-a, problemi sa konfiguracijom i održavanjem i slično. Takođe u većini slučajeva kompanija koja bude koristila WAF open source tipa imaće dodatne troškove obuke zaposlenih ili unajmljivanja lica koje je dovoljno stručno da podesi konfiguraciju i da održava sistem, ali se ovo odnosi na sve softvere koji se budu koristili u svim oblastima, a nisu komercijalnog tipa.

Što se tiče load balancer-a, najčešće korišćeni su Azure Application Gateway, NetScaler, F5 NGINX Plus, AWS Elastic Load Balancing i tako dalje. Isto kao sa WAF softverima, svaki od komercijalnih softvera ima svoje prednosti i mane. Na strani open source load balancer-a najučestaliji su Nginx, Haproxy, Traefik, Envoy, Katran i mnogi drugi.

Komercijalni proxy serveri koji mogu da se koriste i koji se najčešće i koriste su Smartproxy, Nimble, IPRoyal, Nexusnet, ScraperAPI i slično. Nasuprot njima open source su Nginx, Envoy, Kong, F5 BigIP, EZproxy, od kojih se Nginx koristi na skoro 5 miliona *web* sajtova (95% udjela na tržištu [31]).

Softversko rješenje koje treba posebno pomenuti je Wireshark. Wireshark je analizator mrežnog protokola koji ima za cilj da prati sav protok informacija, odnosno paketa ka i od servera. Kada nađemo zahtjev/odgovor koji nas interesuje, možemo da kliknemo na njega i da vidimo sve detalje o istom. Wireshark čita sav saobraćaj real-time (u trenutku), a za nas su bitni HTTP i HTTPs, FTP, SMTP i slični zahtjevi i odgovori koji koriste TCP za transport istih.

6. Simulacija upada i odbrane

U ovom poglavlju ćemo spojiti sve o čemu je pisano do sada u prethodnim poglavljima, od izvršavanja DDoS napada do detekcije i sprečavanja istog. Mora se navesti da je projekat realizovan u simuliranom okruženju sa kontrolisanim i ograničenim uslovima (sandbox), tako da bi u praksi okruženje i proces izgledali malo drugačije, bili bi mnogo veći, pojedini načini realizacije se potencijalno ne bi ni koristili (postoje bolja i praktičnija rješenja) i naravno sve bi bilo više riskantno, sa potencijalno stvarnim gubicima i troškovima. To bi takođe značilo da bi sredstva i resursi koji su izdvojeni za zaštitu bili mnogo veći, što bi dodatno omogućilo sprječavanje bezbjedonosne prijetnje pravovremeno. Mora se isto tako napomenuti da ova simulacija ne predstavlja upustvo kako da se zaštiti sistem u realnom okruženju, kao ni prakse koje se preporučuju, što više ne znači ni da su svi alati i dalje "in-support", već predstavlja jedno od rješenja za potrebu realizacije projekta i dokaza da se može, uz malo planiranja, naći način da se zaštiti arhitektura na aplikativnom nivou uz minimalne troškove.

Prvo ćemo proći kroz sve alate koji su korišćeni za realizaciju projekta, potom ćemo objasniti čemu služi koji alat i na kraju ćemo proći korak po korak kroz ideju realizacije zamišljenog, naravno uz propratne slike koje će prikazati kako to izgleda. Ali prvo da objasnimo ukratko što je projekat koji smo simulirali i koje smo oblasti prošli.

Cilj je bio da se kreira server na kojem se nalazi aplikacija koja komunicira sa bazom podataka. Kada posjetimo aplikaciju, šaljemo HTTP GET zahtjev koji dobija podatke iz baze i prikazuje nam stranicu koju vidimo. Kada se izvrši DDoS napad, u našem slučaju Slowloris DDoS, okupiraju se sve niti do tačke gdje ne može da se uspostavi ni jedna konekcija, što eventualno dovodi do pada servera. Kada pokušamo da pristupimo sajtu, on ga učitava beskonačno dok se ne izvrši "Timeout" i potom prikaže poruku da ne može da pristupi sajtu. Da bi primijetili da se dešava Slowloris napad, na način na koji smo mi izvršili detekciju, bilo je neophodno da nekako dobijemo poruku, što u našem slučaju radimo iz log fajla. Napad se ne dešava direktno na server već preko reverse proxy-ja, odnosno kapije (Application gateway) do našeg servera. Kada detektujemo veliki broj HTTP zahtjeva sa određene IP adrese, tu adresu upisujemo posebnom skriptom u zasebni log fajl, koji potom korišćenjem druge skripte stavljamo na crnu listu (blacklisting). Nakon ovog

procesa, računar sa te IP adrese ne može više da pristupi aplikaciji. Cijeli ovaj proces ćemo objasniti do detalja, nakon što prođemo ostale stavke koje smo naveli prije ovog kratkog objašnjenja.

Alati i softveri koje koristimo u ovom projektu ćemo izdijeliti u par zasebnih grupa. Prvo idu alati i softveri za okruženje i osnovni rad, alati za praćenje paketa, alati za detekciju, alati za prevenciju i skripte. Alati za okruženje i osnovni rad koje su korišćenje su VMWare Work Station 17 Player za podizanje virtualne mašine, XAMPP verzija 3.2.1 sa Apache 2.4 serverom na kojem je podignuta aplikacija, aplikacija sa bazom kreirana korišćenjem Hypertext Preprocessor (PHP) i HyperText Markup Language (HTML) u konekciji sa MySQL bazom podataka i na kraju NGINX verzija 1.18.0. Poslednji alat u prvoj grupi se koristi kao reverse proxy, ali ujedno i logger, odnosno čitač podataka, tako da ćemo njega staviti u dvije grupe. Pod alatima za praćenje paketa smo koristili alat koji smo već jednom pomenuli Wireshark. Kada kažemo alati za detekciju, na način na koji smo ih postavili, rade kao servisi i konstanto upisuju podatke u log fajlove, koji pomoću određenih skripti dovode do prevencije u slučaju da zapisi ispunjavaju određene uslove. Oni nisu tipični alati koji detektuju i alarmiraju vlasnike, ali uz dodatak notifikacija u skriptama mogli bi da postanu upravo to. Međutim kako konstantno nadgledaju zapise iz drugih alata i upisuju ih u log fajlove, smatramo ih alatima za detekciju. Tu spadaju NGINX i NXLog. Zatim imamo alate za prevenciju, odnosno jedan alat za prevenciju, koji u kombinaciji sa skriptama dovodi do prevencije, stavljanjem IP adrese na crnu listu. Naravno nakon što je stavi na crnu listu, NGINX u svome konfiguracionom fajlu trajno zabranjuje pristup računaru sa datom IP adresom. Kao što se može vidjeti NGINX je multifunkcionalan, ima sposobnost detekcije, ali i prevencije, naravno uz pomoć drugih alata ili skripti. Na samom kraju se nalaze skripte ili posebni djelovi koda pisani u našem slučaju u Perl i Python programskim jezicima, koji imaju sposobnost da se izvrše od strane nekog programa. Ovdje ih koristimo u kombinaciji sa drugim alatima koje smo do sada naveli.

VMWare Work Station 17 Player je softver za cloud komputaciju i virtualizaciju. Koristi se za podizanje virtualnih mašina (VM) na kojima se instalira operativni sistem. Umjesto fizičkog računara VM je softver koji emulira pravi računar za koji se alocira određeni dio sredstava. On ima svoj procesor, memoriju, mrežu i hard disk.

Aplikaciju nećemo opisivati detaljnije, jer sami proces kreiranja nije toliko bitan za ovaj projekat. Samo je bitno da aplikacija ima svoj frontend i backend, odnosno prednji i zadnji dio i naravno da je povezana sa bazom podataka, kako bi se mogli slati određeni HTTP zahtjevi.

XAMPP je open source *web* server rješenje koje se sastoji od Apache HTTP Servera, Maria DB i interpretera za skript programske jezike (PHP i Perl). Omogućava da se na njemu podigne *web* server Apache na kojem je postavljena *web* aplikacija. Dok god je *web* server aktivan, aplikaciji će se moći pristupiti na istoj Wi-Fi mreži sa bilo kojeg uređaja, preko posebne IP adrese i porta. Sva podešavanja na serveru možemo mijenjati direktno iz Apache konfiguracionog fajla (httpd.conf) na XAMPP-u, dok sami XAMPP možemo mijenjati iz njegovog konfiguracionog fajla. Ovim fajlovima se može pristupiti iz foldera ili preko samog XAMPP softvera.

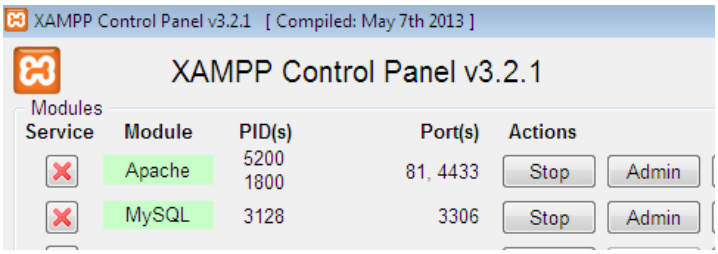
NGINX je softver napisan u C programskom jeziku (*web* server), koji smo u našem slučaju pretvorili u reversed proxy, ali može da funckioniše kao load balancer ili čak i HTTP keš. Koristi master-slave (gospodar-rob) arhitekturu, odnosno master sadrži sve podatke, dok rob je replikacija gospodara. Ovaj vid arhitekture se pretežno koristi za sigurno čuvanje podataka u slučaju gubitka (backup), ali i za povećanje efikasnosti podjelom posla između gospodara i mnogobrojnih robova. U NGINX robove nazivamo radnicima (Workers), koji obavljaju svoj posao tako što svaki provlači zahtjev, koji je stigao kroz zajednički socket, kroz petlje (petlja po radniku), što omogućava veliku efikasnost i obradu velikog broja zahtjeva. Gospodar čita i validira konfiguracije kreiranjem, vezivanjem i miješanjem ulaza. A pored gospodara i roba imamo i Proxy keš, koji se sastoji od keš loader-a i menadžera, koji zapisuje neophodne podatke. Prvi od dva zapisuje podatke u memoriji, dok drugi kontroliše poništavanje i trajanje novog keš-a.

NXLog je alat za sakupljanje podataka i njegovu centralizaciju, koja dodatno omogućava dva veoma bitna pojma enrichment (obogaćivanje) i forwarding (prosljeđivanje) logova. Prvi označava da pored podataka ima sposobnost i dodatnih informacija koje mogu olakšati analiziranje i tumačenje logova. Drugi pojam označava prosljeđivanje logova koji smo mi iskoristili u našem projektu. Pored ova dva pojma, NXLog ima i jednu veoma bitnu sposobnost, promjene logova, alarmiranja, filtriranja po uslovima i slično. Veoma korisan alat, koji može da se koristi na više različitih operativnih sistema.

Wireshark takođe nećemo opisivati, jer se o njemu isto tako govorilo u prethodnom poglavlju. Razlog zašto smo ga koristili u ovom projektu jeste da bi se vizuelno prikazalo kako izgleda kada se izvrši DDoS napad i da bi analizirali pakete.

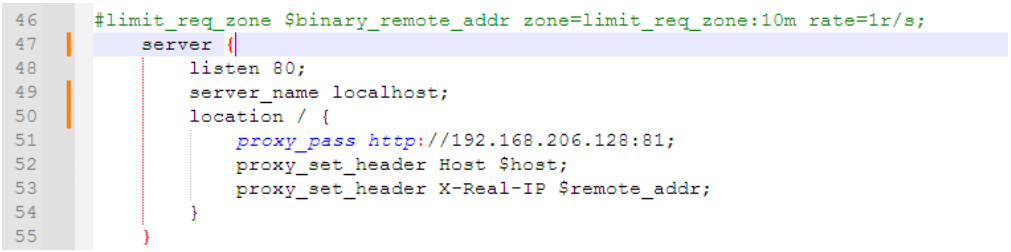
Skripte koje smo koristili za izvršenje DDoS Slowloris napada su dijelom preuzete sa interneta, što ćemo naravno pomenuti u dijelu izvršavanja napada ili su ručno pravljene kao kod dijela prevencije. One nisu alati, ali bez kombinacije sa istima ne bi smo bili u mogućnosti da povežemo sve u jednu veliku cjelinu. S' time na umu, sada ćemo opisati detaljno proces rada i realizacije ovog projekta.

Da bi simulirali više različitih uređaja bilo je neophodno kreirati više različitih uređaja korišćenjem Vmware radne stanice. Kreirali smo tri zasebne virtualne mašine. Prva virtualna mašina je na sebi imala instaliranu iso sliku operativnog sistema 32-bitne Windows sedmice. Ovo je uvelo nove probleme u pogledu pronalaženja alata koji bi mogli da funkcionišu na ovoj verziji operativnog sistema. Drugi problem je predstavljalo to što je većina alata za detekciju i prevenciju kreirana za operativne sisteme zasnovane na Debian Linux-u. Ali ove poteškoće su uvedene kao dokaz da čak i na manje popularnim ili korišćenim operativnim sistemima (na polju zaštite), sa otežanim uslovima može da se osmisli detekcija i zaštita na aplikativnom nivou. Druga virtualna mašina je takođe koristila istu sliku, dok je na treću podignut Kali Linux operativni sistem. Prvu mašinu ćemo zvati matica, drugu ćemo zvati klijent, a treću ćemo zvati napadač, radi lakšeg razlikovanja. Na matici smo instalirali XAMPP unutar kojeg smo postavili našu malu aplikaciju i sql fajl. Potom smo startovali Apache i MySQL module. Da napomenemo da je svaka aplikacija pokrenuta sa administratorskim ovlašćenjima. Konfigurisali smo Apache, tako da se može direktno pristupiti serveru preko porta 81. To ćemo naknadno ispraviti i ovaj port će se koristiti samo za internu komunikaciju između NGINX i Apache *web* servera. Ispod (Slika 21) se može vidjeti koji su moduli pokrenuti.



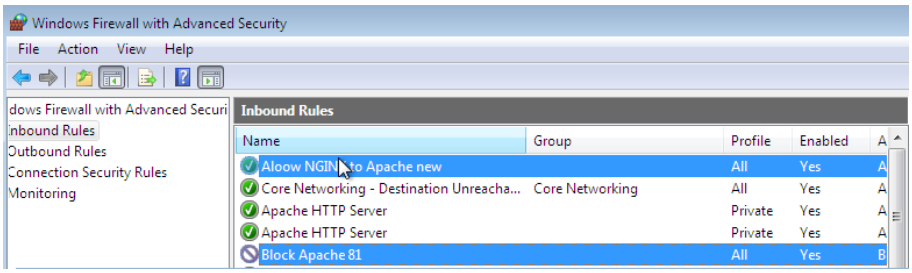
Slika 21: XAMPP

Sledeći korak je bilo konfigurisanje NGINX da radio kao reverse proxy. Otvorili smo konfiguracioni fajl NGINX-a i u njemu unutar "http" bloka unijeli sledeću konfiguraciju (Slika 22).

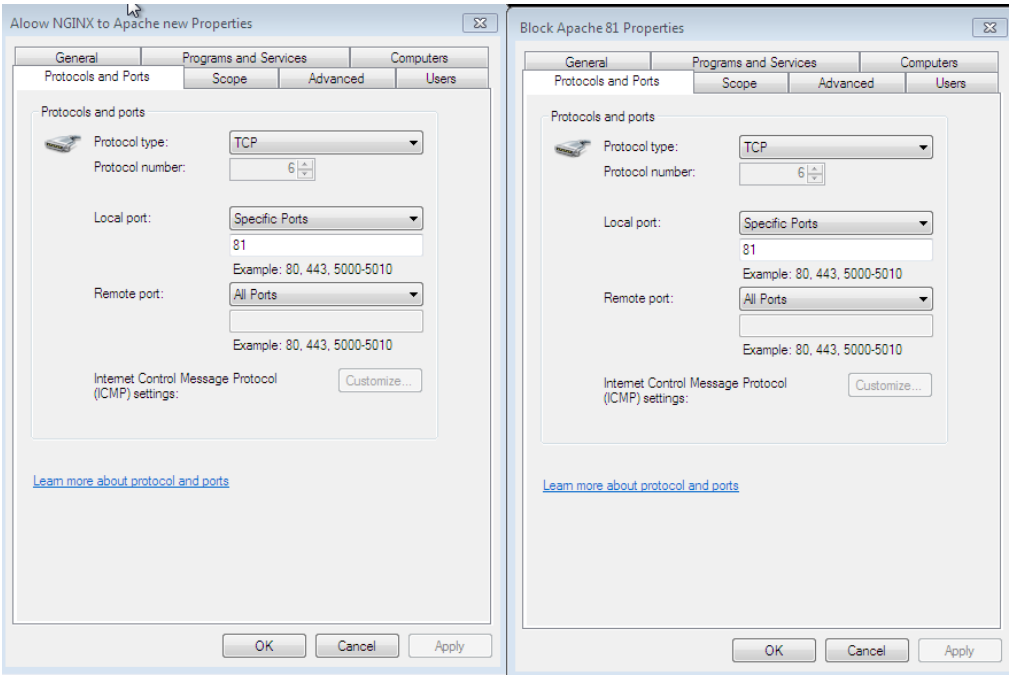


Slika 22: NGINX conf fajl

Unijeli smo novi blok sa "server" tag-om, unutar kojeg smo podesili NGINX da sluša HTTP zahtjeve preko porta 80, a ime servera na localhost. Zatim još jedan "location" blok unutar kojeg se definiše kako da se postupa sa zahtjevima do URL root putanje. "Proxy_pass" kaže da se zahtjevi trebaju prosleđivati do adrese (192.168.206.128) : porta 81, odnosno do našeg servera. "Proxy_set_header" je zadužen za postavljanje zaglavlja na istu vrijednost kao i kod originalnog vlasnika, da bi destinacioni server znao u kojem domenu da šalje zahtjev. "Proxy_set_header" služi za postavljanje još jednog dodatnog zaglavlja, koji će da obavijesti pozadinski server o IP adresi klijenta koji je poslao zahtjev. Sledeća stavka je da se onesposobi direktan pristup serveru preko porta apache servera (81), odnosno da se omogući samo komunikacija preko reverse proxy-ja (port 80). Ovo smo izvršili pomoću "Windows Firewall with Advanced Security" (Windows Fajervol sa naprednom zaštitom), ugrađenog firewalla na Windows-u. Kreirali smo dva Inbound (dolazna) pravila, odnosno pravila koja služe da spriječe sav direktni dolazni saobraćaj ka našem *web* serveru (Slika 23). Prvo pravilo, gledano od lijevo ka desno, omogućava komunikaciju NGINX-a sa *web* serverom, dok drugo pravilo sprečava sav dolazni saobraćaj ka istom *web* serveru, koji nije sa porta 80 (Slika 24).



Slika 23: Windows Firewall

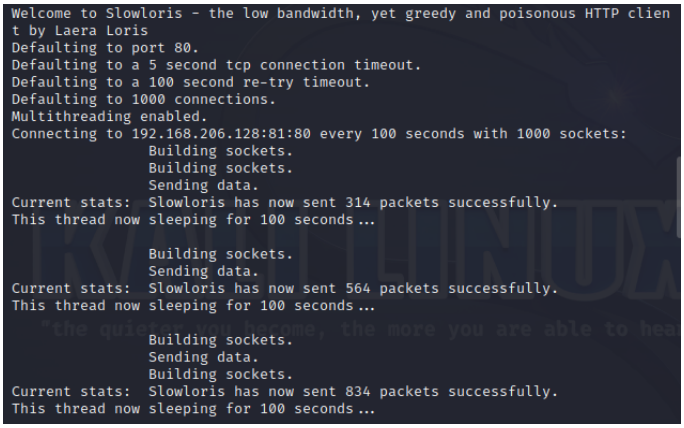


Slika 24: Inbound pravila

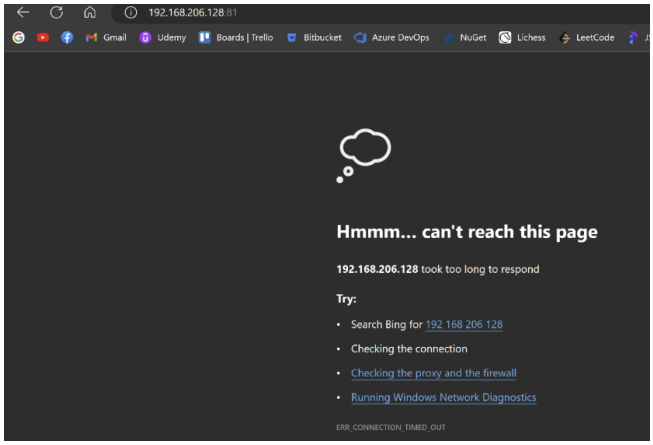
Ovo je omogućilo da se aplikaciji više ne pristupa preko porta 81, već preko porta 80, odnosno NGINX-a. Sledeći korak je bio kreiranje DDoS napada, ili Slowloris DDoS napada da budemo precizniji. Skriptu je kreirao user na GitHub-u sa korisničkim imenom “llaerallaera” [32], a kreirana je u Perl programskom jeziku. Skripta šalje veliki broj HTTP zahtjeva, na isti ulaz i okupira veliki broj niti, a vrijeme između ponovnog slanja zahtjeva može ručno da se podesi, tako da se mogu češće slati. U našem primjeru koristimo napadač na kome u terminalu unosimo komandu za pozivanjem skripte, koja se nalazi na našem desktop-u (Slika 25). Skripta generiše

odgovor gdje se može vidjeti da se generiše oko 1000 konekcija, veliki broj poslatih paketa i da će opet pokušati da pošalje još toliko u narednih 100 sekundi (Slika 26). Ovo će okupirati sve niti, što će zauzvrat generisati sledeću poruku (Slika 27) svakom uređaju, u našem slučaju klijent, koji pokuša da pristupi aplikaciji na *web* serveru.

Slika 25: Komanda za pokretanje skripte

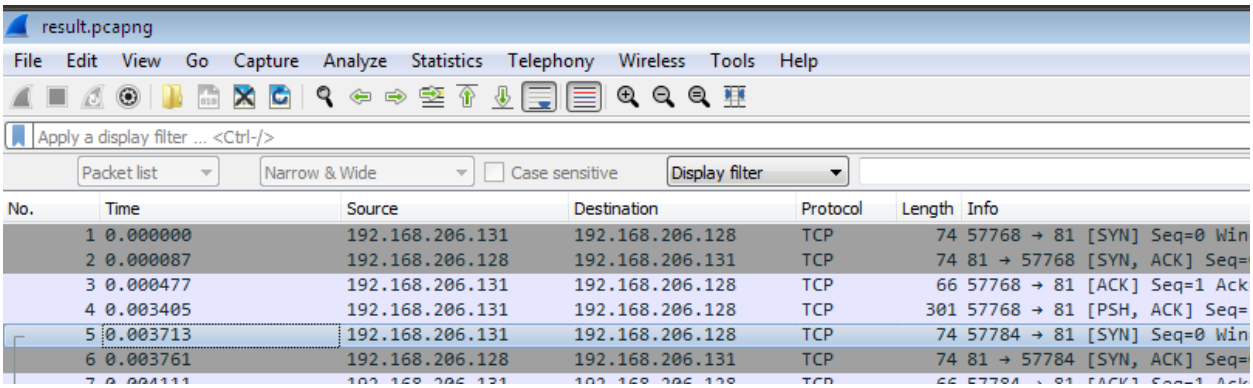


Slika 26: Generisan poruka skripte

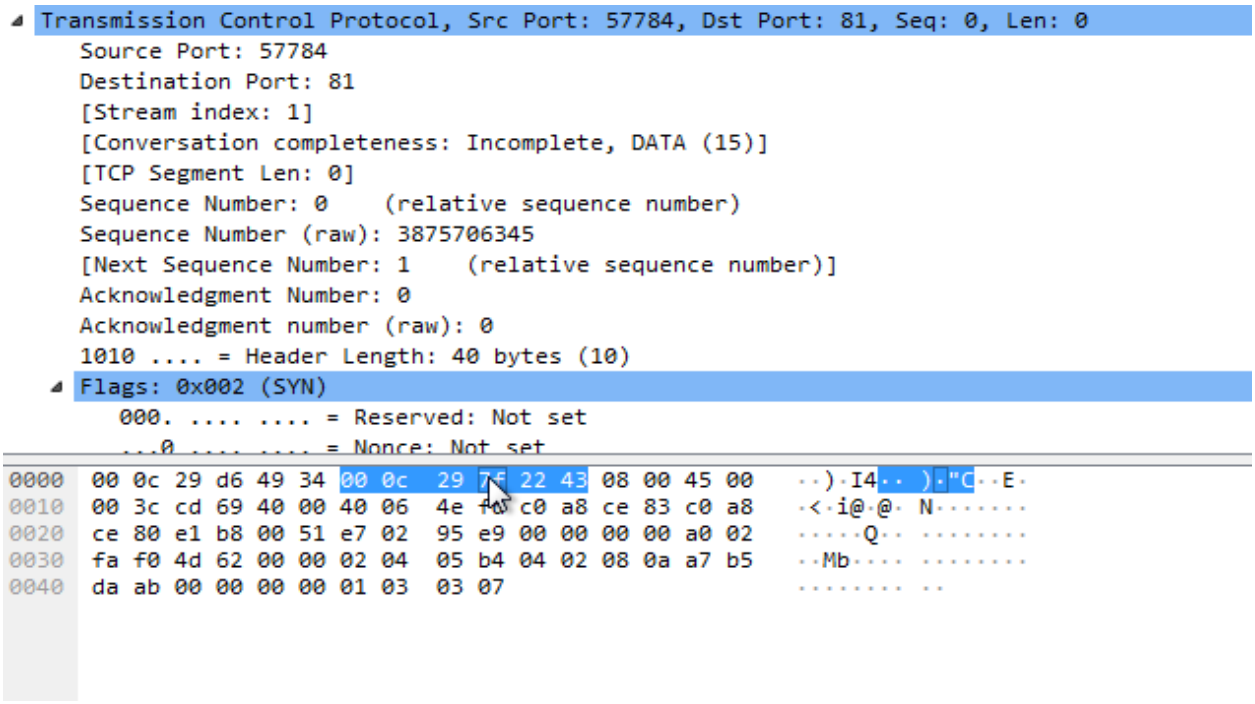


Slika 27: Timeout

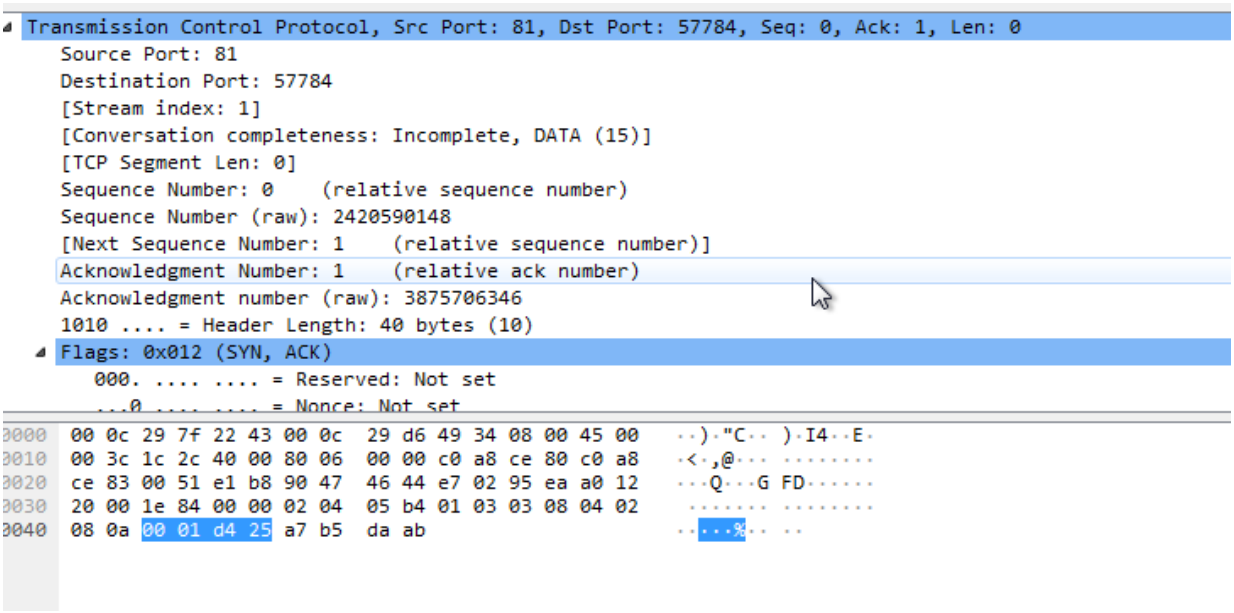
Ova skripta generiše veliki broj HTTP zahtjeva, odnosno pokušaja uspostavljanja TCP konekcije koji može da se prati preko Wiresharka. Na slici 28 možemo vidjeti ogroman broj TCP paketa, kolona "Protocol" opisuje tip, koje pokušavaju da dođu do svoje destinacije, takođe vidimo vrijeme zahtjeva, izvor IP adrese, odnosno adrese napadača, IP adresu destinacije ili matice, dužinu paketa i slično. Inicijalno bi trebalo da je ostvaren "Three way handshake", gdje se šalje SYN paket, server potom prima SYN-ACK paket i na kraju opet šalje ACK paket ka serveru, a potom se šalju samo SYN paket i SYN-ACK paket. U paketu možemo vidjeti u polju flags (zastave) na slikama 29 i 30, pri čemu je prva ona koja generiše SYN i kao izvor ima napadačevu IP adresu, a druga je ona koja generiše SYN-ACK i kao izvor ima maticinu IP adresu. Glavni pokazatelj u našem slučaju je veliki broj TCP paketa koji pristiže sa jedne lokacije, kao i flag "SACK_PERM = 1", koji označava gubitak paketa. Ovo je glavni pokazatelj u našem simuliranom okruženju, ali i ne dovoljan pokazatelj u realnoj situaciji. Ako želimo samo da se usresredimo na određeni tip protokola ili IP adresu ili port, to možemo da uradimo u polju iznad tabele, gdje kaže "Apply a display filter, ...". Detaljnije informacije možemo vidjeti klikom na sami zahtjev, kao što su izvorni port, da li je završena konverzacija, što u našem slučaju govori da socket još uvjek pokušava uspostaviti konekciju (SYN_SENT), veličinu prozora, veličinu zaglavlja, broj sekvence i mnoge druge bitne podatke.



Slika 28: Wireshark capture

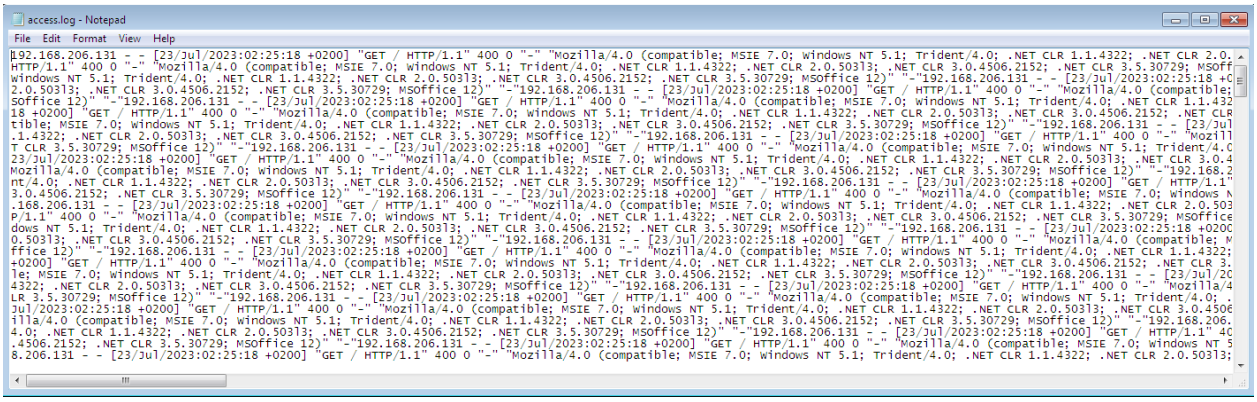


Slika 29: Paket SYN



Slika 30: Paket SYN – ACK

Kada napad započne izvršavanje, sledeća stvar koju smo uradili jeste čitanje log fajla iz NGINX foldera (Slika 31). U ovaj fajl se podaci upisuju neprestano i količina podataka i tip podataka je veoma različit. Zato smo morali smisliti način na koji bi čitali ove podatke i upisivali ih u poseban log fajl, koji smo kreirali, drugim riječima morali smo naći način da ih filtriramo.

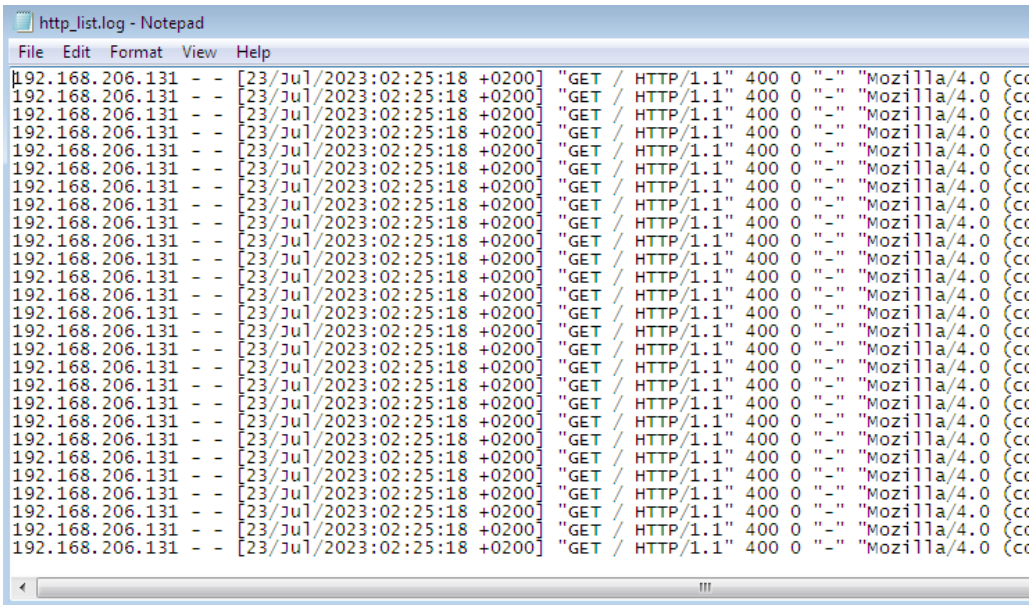


Slika 31: NGINX access.log fajl

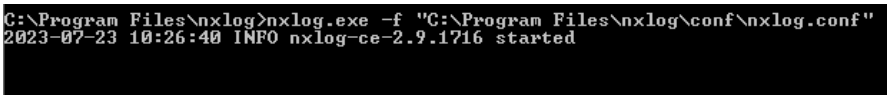
Ovo smo ostvarili korišćenjem NXLog alata, koji po određenom pravilu koje smo mi postavili filtrira podatke iz access.log fajla i upisuje u http_list.log fajl, odnosno fajl koji smo mi kreirali. Unutar NXLog konfiguracionog fajla smo postavili određena podešavanja (Slika 32). Prvo smo generisali pravilo “ONLY_HTTP” koje će posmatrati samo ako sadrži “GET /HTTP/ 1.1”, sve ostalo će ignorisati. Postavili smo par atributa za fajlove koje smatramo ulaznim fajlovima. Module se postavlja na “im_file”, što označava da se nadgledaju log fajlovi. Ovo naravno može biti “im_tcp” za primanje log fajlova preko TCP, “pm_grok” za manipulisanje logova i bilo koji drugi koji postoji (više se može pronaći u dokumentaciji). File je postavljen na “putanju/fajl” koji je meta i na kraju Exec koji služi da izvrši pravilo sa početka. Za fajlove koji su output tipa upisujemo vrijednosti iz fajla koji posmatramo kao ulazni. Atributi - Module koji je postavljen na “om_file” služi za upisivanje podataka u drugi fajl, a za File je postavljena putanja i naziv fajla (http_list.log). I na samom kraju gdje se nalaze oznake Route, u njima se nalazi pravac izvršavanja definicija, odnosno iz ulaznog u izlazni fajl. Na slici 33 se može vidjeti kako izgledaju podaci u http_list.log fajlu, s' time da svaka linija predstavlja novi red. A odmah nakon nje na slici 34 se može vidjeti i kako izgleda pokrenuti servis.

```
#define ROOT C:\Program Files\nxlog
define ROOT C:\Program Files\nxlog
define ONLY_HTTP if $raw_event =~ /^GET \/ HTTP\/1\.1\s/ drop();
#$raw_event !~ /http:\/\// &&

Moduledir %ROOT%\modules
CacheDir %ROOT%\data
Pidfile %ROOT%\data\nxlog.pid
SpoolDir %ROOT%\data
LogFile %ROOT%\data\nxlog.log

<Extension _syslog>
    Module      xm_syslog
</Extension>

<Input in_file>
    Module      im_file
    File        "C:\\nginx\\logs\\access.log"
    Exec        %ONLY_HTTP%
</Input>

<Output out_nginx>
    Module      om_file
    File        "C:\\Scripts_Folders\\http_list.log"
</Output>

<Route 2>
    Path        in_file => out_nginx
</Route>
```

Slika 32: NXLog konfiguracioni fajl

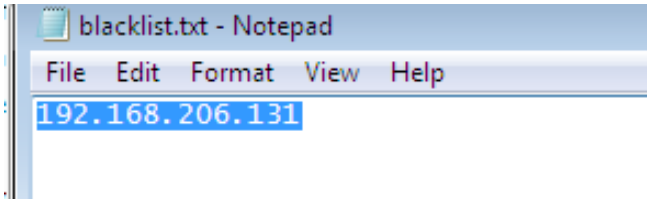


Slika 33: http_list.log



Slika 34: Pokretanje NXLog servisa

Kada smo upisali sve podatke u log fajl, neophodno je pomoću skripti da izdvojimo sve podatke u poseban tekstualni fajl koji smo nazvali "blacklist.txt", unutar kojeg ćemo čuvati samo IP adrese koje će biti dodate na crnu listu. Iako ovaj način rješava problem, u praksi se više preferira „whitelisting", odnosno dodavanje na bijelu listu. Ovaj način je bolji jer se na bijeloj listi nalaze samo IP adrese kojima je dozvoljen pristup, ali samo kada radimo sa ograničenim IP adresama ili pool-om IP adresa [33]. Svi redovi u http_list.log imaju isti format, te tako nije bilo teško osmisliti skriptu koja će da izvlači podatke koji su nam neophodni. Za potrebe pisanja skripti korišćeni su "Sublime text" IDE i Python 3.7 programski jezik. Pošto se radi o jednoj velikoj skripti, izdvojili smo je na dva dijela. Prvi dio prikazuje stavljanje na crnu listu, a drugi zabranu pristupa. Iskoristili smo dvije bitne cijeline iz http_list.log fajla, prvu koji predstavlja IP adresu i drugu koja predstavlja vrijeme upisivanja u log fajl. Kod nećemo prelaziti liniju po liniju, ali ćemo objasniti kako je osmišljen. Ako u rasponu od jednog minuta broj zapisa unutar http_list.log fajla pređe 100 instanci sa istom IP adresom, ta adresa se dodaje na crnu listu, odnosno upisuje u blacklist.txt fajl. Skripta pamti IP adresu, njen broj pojavljivanja i zadnje vrijeme pojavljivanja u memoriji, kreiranjem rječnika koji sadrži te iste vrijednosti. Takođe ovaj kod provjerava da li je IP adresa već dodata na crnu listu i ako jeste ne posmatra više redove sa tom IP adresom. Ako je vremenska razlika između prve i zadnje pojave IP adrese veća od minut podesi brojač na jedan i upiši sledeće vrijeme u nizu kao novo. Ovo je dobar način da se ograniči broj zahtjeva u minuti. Kreiran je kao beskonačna petlja, tako da stalno prolazi kroz http_list.log fajl, pri čemu prati da li je izvršena kakva promjena na fajlu i nastavlja gdje je stao. Ovim smo kreirali jednu vrstu servisa. Jeste memorijski zahtjevan proces, zbog potencijalnog broja IP adresa koje moraju da se pamte, ali s' obzirom da se radi o simuliranom okruženju ne predstavlja toliki problem. Način na koji bi se mogao riješiti problem sa memorijom je periodično čišćenje svih IP adresa upoređivanjem vremena koje je zapamćeno sa trenutnim vremenom. Ako je prošlo duže od jednog minuta kompletno izbriši IP adresu iz memorije. Ovo je task koji bi se mogao izvršavati na svaki par sati ili dana. Sve o čemu smo sada pričali se može vidjeti na slikama 35, 36, 37, 38 i 39.



Slika 35: blacklist.txt

```python
import os
import time
import re
# from watchdog.observers import Observer
# from watchdog.events import PatternMatchingEventHandler

http_list_file = "C:\\Scripts_Folders\\http_list.log"
blacklist_file = "C:\\Scripts_Folders\\blacklist.txt"
apache_config_file_path = "C:\\xampp\\apache\\conf\\httpd.conf"
print("Zapocni pretragu;")

class IP:
    def __init__(self, first_occurrence_time):
        self.first_occurrence = first_occurrence_time
        self.count = 1
```

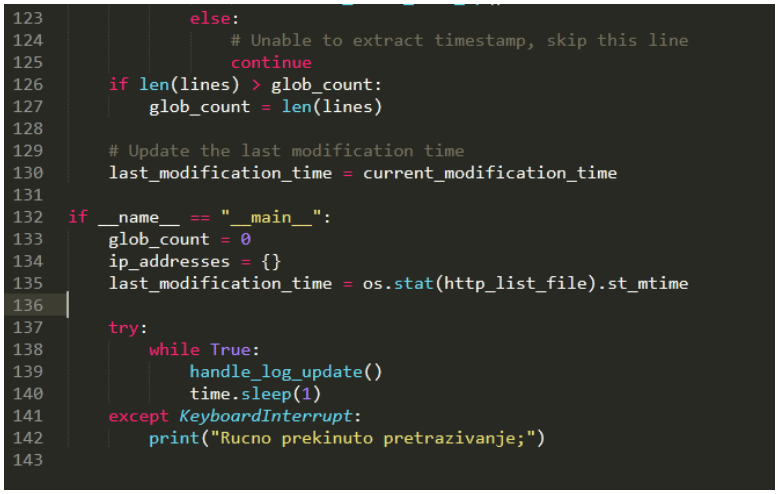
Slika 36: Kod_sa_putanjama

```python
def update_blacklist(ip):
    with open(blacklist_file, "a") as f:
        f.write(ip + "\n")

def is_blacklisted(ip):
    with open(blacklist_file, "r") as f:
        for line in f:
            if ip.strip() == line.strip():
                return True
    return False

def handle_log_update():
    global last_modification_time
    global ip_addresses
    global glob_count
    # Check if the file has been modified
    current_modification_time = os.stat(http_list_file).st_mtime
    if current_modification_time == last_modification_time:
        with open(http_list_file, "r") as f:
            lines = f.readlines()
            time.sleep(1)
            if len(lines) <= glob_count:
                return

    last_modification_time = current_modification_time

    with open(http_list_file, "r") as f:
        lines = f.readlines()
```

Slika 37: Kod_otvaranje_azuruiranje

```
    for line in lines[glob_count:]:
        # Use regular expression to extract IP address from the beginning of each line
        ip_match = re.match(r"^(\d+\.\d+\.\d+\.\d+)\s", line)
        if ip_match:
            ip = ip_match.group(1)

            # Check if IP address is already blacklisted or if the line is not starting with an IP address
            if is_blacklisted(ip):
                continue

            # Extract timestamp from the line
            timestamp_match = re.search(r"\[([^]]+)\]", line)
            if timestamp_match:
                timestamp_str = timestamp_match.group(1)
                timestamp = time.mktime(
                    time.strptime(timestamp_str, "%d/%b/%Y:%H:%M:%S %z")
                )

                # Check if IP address has occurred before
                if ip in ip_addresses:
                    # Check if one minute has passed since the first occurrence
                    if timestamp - ip_addresses[ip].first_occurrence > 60:
                        # Reset the count and update first_occurrence to the current timestamp
                        ip_addresses[ip].count = 1
                        ip_addresses[ip].first_occurrence = timestamp
                        glob_count += 1
                        print(glob_count)
                else:
                    # New IP address, add it to the dictionary with the first occurrence time
                    ip_addresses[ip] = IP(timestamp)

                # Check if the current timestamp is within the one-minute range
                if timestamp - ip_addresses[ip].first_occurrence <= 60:
                    # Increment the occurrence count
                    ip_addresses[ip].count += 1

                # Check if the count is 100 to add to the blacklist
                if ip_addresses[ip].count >= 100 and not is_blacklisted(ip):
                    update_blacklist(ip)
                    print("Uspjesno je dodata ip adresa " + ip + " u blacklist-u;")
```

Slika 38: Kod_ip_provjera_lista



```
            else:
                # Unable to extract timestamp, skip this line
                continue
    if len(lines) > glob_count:
        glob_count = len(lines)

    # Update the last modification time
    last_modification_time = current_modification_time

if __name__ == "__main__":
    glob_count = 0
    ip_addresses = {}
    last_modification_time = os.stat(http_list_file).st_mtime

    try:
        while True:
            handle_log_update()
            time.sleep(1)
    except KeyboardInterrupt:
        print("Rucno prekinuto pretrazivanje;")
```

Slika 39: Kod_ogranicenje

U drugoj cjelini (Slika 40, Slika 41) se može vidjeti metoda koja prolazi kroz blacklist.txt fajl, čita iz fajla i sve IP adrese upisuje u nginx.conf fajl. Naravno ona ih ovdje zvanično stavlja na listu IP adresa koje ne mogu da pristupe serveru. Unutar location tag-a stavlja “deny“ (odbiti) komandu sa IP adresom koju želi da blokira (Slika 42). Jedini problem je što ne postoji dobar način da se ovo upiše u neki fajl bez brisanja postojećeg sadržaja. S' tim razlogom je sav sadržaj

konfiguracionog fajla podijeljen na dva dijela u posebne tekstualne fajlove. Kada treba da se blokira IP adresa, on uzima sadržaj iz oba fajla i kao sendvič stavlja komadnu “deny“ između na pravoj lokaciji sa svim IP adresama. Na ovaj način je riješeno upisivanje koda u fajlu. Na kraju kada napadač želi da pristupi aplikaciji preko adrese i porta, dobije poruku “403 Forbidden” ili zabranjeno (Slika 43). Na slici ispod se može vidjeti tekst koji ispisuje konzola prilikom pokretanja i rada (Slika 44).

```
def combine_files_with_ip():
    file1_path = "C:\\nginx\\conf\\conf_split\\conf1.txt"
    file2_path = "C:\\nginx\\conf\\conf_split\\conf2.txt"
    ip_addresses_file_path = "C:\\Scripts_Folders\\blacklist.txt"
    output_file_path = "C:\\nginx\\conf\\nginx.conf"


    def read_file_content(file_path):
        with open(file_path, 'r') as file:
            content = file.read()
        return content

    def append_ip_addresses(file_content_first, ip_addresses_file_path):
        file_content = ''
        with open(ip_addresses_file_path, 'r') as ip_file:
            for ip in ip_file:
                file_content += "\n            deny " + ip.strip() + ";\n"
        file_new_content = file_content_first + file_content
        print("IP adresa/e blacklist-ana u konfiguracije NGINX servera.")
        return file_new_content

    # Read the content of file1 and file2
    file1_content = read_file_content(file1_path)
    file2_content = read_file_content(file2_path)
    combined_content = ""
    combined_content += file1_content.strip()
    combined_content = append_ip_addresses(combined_content, ip_addresses_file_path)
    combined_content += file2_content
    with open(output_file_path, 'w') as final_file:
        final_file.write(combined_content)
```

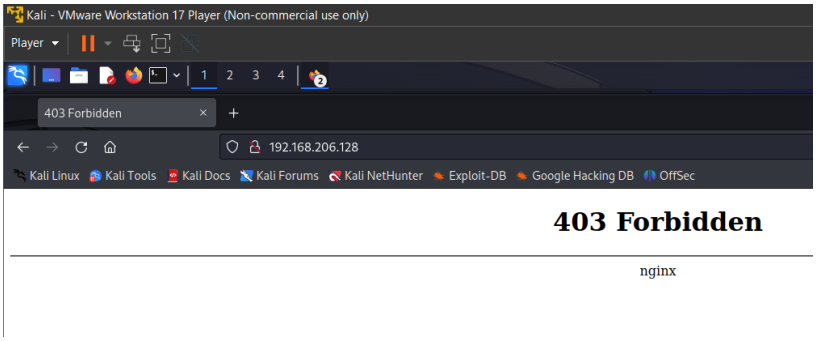
Slika 40: Kod_upisivanje_u_fajl

```
if ip_addresses[ip].count >= 100 and not is_blacklisted(ip):
    update_blacklist(ip)
    print("Uspjesno je dodata ip adresa " + ip + " u blacklist-u;")
    combine_files_with_ip()
```

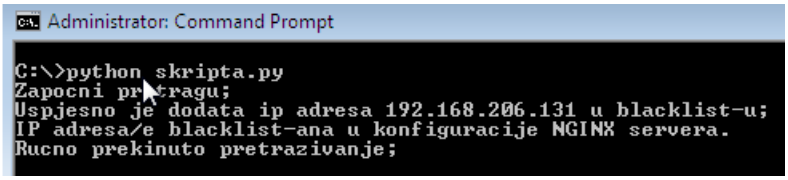
Slika 41: Kod_dodat_na_blacklistu

```
server {
    listen 80;
    server_name localhost;

    location / {
        deny 192.168.206.131;
        proxy_pass http://192.168.206.128:81;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
```

Slika 42: Deny



Slika 43: 403 Forbidden



Slika 44: Izvršenje i blokada

Projekat je realizovan i može biti dopunjavan na razne načine i sa raznim alatima koji bi poboljšali detekciju i prevenciju. Naravno ima svoje nedostatke (primjer: radi samo sa jednom vrstom napada), ali i prednosti. Cilj bio da se pokaže da je moguće realizovati isti uz malo kreativnosti, čak i pri otežanim uslovima, međutim ne treba bježati ni od komercijalnih softvera. Dobra odbrana se ostvaruje korišćenjem komercijalnih i open source softvera u kombinaciji sa skriptama. Ne postoji idealno rješenje koje može da obavlja sve uloge i da spriječi sve vrste napada, kao što ne postoji napad koji može da prođe kroz sve nivoe odbrane.

7. Rezultati i diskusija

Ovo istraživanje je bilo uspješno u pogledu realizacije izvršenja napada kao i detekcije i prevencije istog. Uz resurse koji su dodatno otežavali rad i realizaciju uspjeli su se naći alati koji bi uspješno realizovali detekciju i prevenciju napada, a pri čemu su troškovi bili nepostojeći.

Projekat je uspio da pokaže da mala preduzeća, pogotovo u početku rada mogu da se probiju korišćenjem open source alata, a pri čemu i dodatno mogu da umanje troškove virtualizacijom, uz sve alate za detekciju i prevenciju koji su već implementirani u samom cloud rješenju. Takođe dodatno smanjenje troškova mogu pronaći uz stalna testiranja i potragom za slabostima, kao i "Out-sourcing", odnosno unajmljivanjem drugih preduzeća koji će ovaj posao da obavljaju za njih, što se dugoročno isplati, pogotovo ako se uračuna da preduzeće nema potrebe za plaćanjem zaposlenih lica i njihove edukacije.

Kombinacija alata koji su open source i alata koji su komercijalni je mnogo lakša, zbog podrške velikog tima koji stoji iza komercijalnog alata. U projektu smo uspjeli da kombinujemo određeni broj open source alata uz nepostojeću podršku, što je veoma ohrabrujuća činjenica, pogoto kada preduzeća imaju stručna lica koji znaju da obavljaju ovaj posao.

Treba naglasiti da projekat nismo bili u mogućnosti da realizujemo sa komercijalnim alatima, jer istim nismo mogli da dobijemo pristup. Isto tako nismo mogli da primijenimo test u pravim preduzećima sa pravim serverima, pravim okruženjem i pravim podacima. Zbog potencijalnog rizika i sigurnosti podataka ni jedna kompanija nije mogla da ustupi svoje resurse zarad realizacije DDoS napada, kao i realizacije master rada.

U procesu implementacije alata za detekciju i prevenciju je naučeno mnogo, posebno kod alata koji se masivno koriste kao što su Wireshark i NGINX, a posebno zadovoljstvo predstavlja osmišljen način realizacije detekcije i prevencije, koji se nakon dugog procesa punog "trial and error" (pokušaja i griješenja) uspio ostvariti.

8. Zaključak

Realizacijom ovog projekta dokazano je da se mala i velika preduzeća mogu, na jedan od načina pomenutih u master radu, bilo to korišćenje open source softvera, vršenje analiza, učestalih testiranja i slično, zaštititi od raznih napada na aplikativnom nivou. Alati se mogu naći za razne operativne sisteme, što više mogu se koristiti razni operativni sistemi za razna ukrštanja alata u pogledu detekcije i zaštite. Dokazano je da se mogu kombinovati razni alati i implementirati na razne načine i oblike. Uz propratnu edukaciju i ocjenu rizika, problemi se mogu rješavati pravovremeno uz minimiziranje potencijalnih troškova.

U radu su se prošle razne teme kao što su bezbjedonosne prijetnje, njihova podjela, klasifikacija, što je veoma bitno da bi se upoznali sa načinom njihovog funkcionisanja, ali u upoznavanja sa načinom razmišljanja napadača, kao i slabosti koje bi isto iskoristio. Ovo nas je dovelo do nekih od načina detekcije tih upada, bez čega ne bi mogli da pokrenemo proces prevencije i oporavka od istih. Da bi se napad detektovao veoma je bitno da znamo o kojem se napadu radi, koje oblasti cilja i da se procijeni rizik koje potencijalno može nanijeti. Nakon toga smo prošli kroz neke od najčešće korišćenih načina odbrane, i neke od učestalih praksi kada se radi o testiranju aplikacija i provjere nivoa sigurnosti istih. Na samom kraju smo prikazali samu simulaciju i dokazali da je moguće da se zaštiti aplikacija na jedan od načina koji se koristi u praksi, koristeći alate koji su “open source“. Uspješno realizovan projekat trebalo bi da bude od značaja za mala i srednja preduzeća i dokaz da mogu održati kompetitivnost i pored velikih preduzeća ili uopšteno ostvariti uspješan početak ne brinući o visini troškova i zaštiti ličnih podataka i podataka svojih klijenata, naravno uz određeni balans u načinima implementacije zaštite.

Ovim istraživanjem je postavljen temelj za sva mala preduzeća, nova i postojeća, koja na neki od načina žele da umanje troškove i da upoznaju lica zadužena za sigurnost i zaštitu sa potencijalnim načinima detekcije i prevencije, pogotovo u oblasti aplikativnog nivoa.

9. Literatura

[1] Cesar Bravo and Darren Kitchen, "Mastering Defensive Security: Effective techniques to secure your Windows, Linux, IoT, and cloud infrastructure", January 2022, 528 Pages, ISBN: 978-1800208162.

[2] “Black Friday Report: banking credentials theft doubled in 2022”, 2023 AO Kaspersky Lab., November 23, 2022, https://www.kaspersky.com/about/press-releases/2022_black-friday-report-banking-credentials-theft-doubled-in-2022.

[3] Dr. Kevin E. Foltz and Dr. William R. Simpson, “Enterprise Level Security”, Taylor & Francis Group, LLC, February 2016, ISBN: 978-1-4987-6445-2 (Hardback).

[4] Michael Roytman and Ed Bellis, “Modern Vulnerability Management: Predictive Cybersecurity”, Artech House, 2023, ISBN 13: 978-63081-938-5.

[5] Ravi Das and Greg Johnson, "Testing and Securing *Web* Applications", August 2020, 224 Pages, ISBN: 978-0367333751.

[6] Subrahmanian, V.S., “Cybersecurity Needs a New Alert System,” The Wall Street Journal, March 29, 2021, https://www.wsj.com/articles/cybersecurity-needs-a-new-alert-system-11617039278.

[7] Richa Gupta, "Hands-on Penetration Testing for *Web* Applications: Run *Web* Security Testing on Modern Applications Using Nmap, Burp Suite and Wireshark (English Edition)", March 2021, ISBN: 978-9389328547.

[8] Pretty Mishra, Emmanuel S Pilli and R C Joshi, "Cloud Security: Attacks, Techniques,Tools, and Challenges", December 2021, 240 Pages, ISBN: 978-0367435820.

[9] Derrick Rountree, Ileana Castrillo, “The Basics of Cloud Computing”,
1st Edition - September 3, 2013, ISBN: 9780124055216.

[10] Sreekanth Iyer, “Hybrid Cloud Security Patterns”, Packt Publishing Ltd., December 2022, ISBN: 978-1-80323-358-1.

[11] Brian T. O'Hara and Ben Maliso, "(ISC)2 CCSP Certified Cloud Security Professional Official Study Guide: 2nd Edition", March 2021, ISBN: 9781663711120.

[12] Brij Gupta, Dharma P. Agrawal, Shingo Yamaguchi, “Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security “, ISBN: 9781522501060, 2016.

[13] “AVOIDIT: A Cyber Attack Taxonomy”, Chris Simmons, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Qishi Wu, Department of Computer Science, University of Memphis, Memphis, TN, USA, 01.01.2009., ONR grant: N00014-09-1-0752.

[14] Yuri Diogenes and Erdal Ozkaya, “Cybersecurity – Attack and Defense Strategies: Second Edition”, Packt Publishing Ltd., December 2019, ISBN: 978-1-83882-779-3.

[15] Robert Douglass, Keith Gremban, Ananthram Swami, Stephan Gerali, “IoT for Defense and National Security”, The Institute of Electrical and Electronics Engineers, Inc., 2023, Hardback ISBN: 978-1119892144.

[16] Sabri Shima, Ismail Noraini, Hazzim Amir, “Slowloris DoS Attack Based Simulation“, IOP Conference Series: Materials Science and Engineering, February 2021, DOI: 10.1088/1757-899X/1062/1/012029

[17] Chad Calvert, Clifford Kemp, Taghi M. Khoshgoftaar, Maryam M. Najafabadi, “Detecting Slow HTTP POST DoS Attacks Using Netflow Features“, Florida Atlantic University, Boca Raton, FL 33431

[18] Junhan Park, Keisuke Iwai, Hidema Tanaka, Takakazu Kurokawa, “Analyses of Slow Read DoS Attack and Countermeasures”, November 2014.

[19] A.N.H.D. Sai, B.H. Tilak, N.S. Sanjith, P. Suhas, R.Sanjeetha, “Detection and Mitigation of Low and Slow DDoS attack in an SDN environment“, IEEE, 2022, DO: 10.1109/DISCOVER55800.2022.9974724

[20] Yu Shui, Zhao Guofeng, Guo Song, Xiang Yang, Vasilakos Athanasios, “Browsing Behavior Mimicking Attack on Popular Web Sites For Large Botnes“, May 2011, DO: 10.1109/INFCOMW.2011.5928949

[21] S.T. Zargar, J. Joshi, D.Tipper, “A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks“, IEEE, 2013, SN: 1553-877X, DO: 10.1109/SURV.2013.031413.00127

[22] Yu Shui, Zhao Guofeng, Guo Song, Xiang Yang, Vasilakos Athanasios, “Browsing Behavior Mimicking Attacks On Popular *Web* Sites For Large Botnets, jo: 2011 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2011, do: 10.1109/INFCOMW.2011.5928949.

[23] Ettore Galluccio, Edoardo Caselli and Gabriele Lombar, "SQL Injection Strategies: Practical techniques to secure old vulnerabilities against modern attacks", July 2020, 210Pages, ISBN: 978-1839215643.

[24] Sandeep Saxena and Ashok Kumar Pradhan, “Internet of Things: Security and Privacy in Cyberspace”, Springer Nature Singapore Pte Ltd., 2022, ISBN: 978-981-19-1585-7.

[25] David Scott and Richard Sharp, "Abstracting application-level *web* security", May 2002, doi: 10.1145/511446.511498.

[26] Mr. Rahul Suryawanshi, Aniket Sorte, Kunal Sahare, Sahil Tembhare, “Runtime Application Self Protection”, International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), Volume, Issue 5, June 2022, ISSN (Online) 2581-9429, doi: 10.48175/IJARSCT-4885.

[27] Christian Martorella, “Learning Python *Web* Penetration Testing”, Packt Publishing Ltd., June 2018, ISBN: 978-1-78953-397-2.

[28] Glendford J. Myers, Tom Badgett and Corey Sandler, "The Art Of Software Testing: Third Edition", November 2011, 256 pages, ISBN: 978-1118031964.

[29] Velu Vijay Kumar, "Mobile Application Penetration Testing", March 2016, ISBN: 9781785888694.

[30] Yuri Diogenes and Erdal Ozkaya, “Cybersecurity – Attack and Defense Strategies: Third Edition”, Packt Publishing Ltd., September 2022, ISBN: 978-1-80324-877-6.

[31] Wappalyzer.com - https://www.wappalyzer.com/technologies/*web*-servers/nginx/.

[32] Slowloris DDoS - https://github.com/GHubgenius/slowloris.pl.

[33] Roman Canlas, “ASP.NET Core 5 Secure Coding Cookbook”, Packt Publishing Ltd., June 2021, ISBN: 978-1-80107-156-7.