SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1946

Prototip sustava za uvježbavanje analitičara za kibernetičke prijetnje

Mladen Petr

Zagreb, lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 8. ožujka 2019.

DIPLOMSKI ZADATAK br. 1946

Pristupnik: **Mladen Petr (0036479433)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Prototip sustava za uvježbavanje analitičara za kibernetičke prijetnje**

Opis zadatka:

Sigurnost tvrtki i država temelji se na dobrom poznavanju prijetnji kako bi se uvele odgovarajuće zaštite te u slučaju napada umanjila šteta zbog incidenta. Ključna aktivnost u tom smislu je područje koje se bavi analizom kibernetičkih prijetnji, a koje je spoj klasičnog obavještajnog rada i kibernetičke sigurnosti. Analizu prijetnji provode osobe - analitičari - te se radi o vrlo specifičnoj vještini koju je vrlo teško automatizirati. Iz tog razloga postoji potreba za takvim kadrom, ali edukacija kadra je dugotrajna i zahtjevna te se uglavnom temelji na radu s iskusnim analitičarima, pokušajima i pogreškama. Sustav koji bi stavljao analitičara u fiktivnu situaciju u kojoj mora istraživati neku konkretnu prijetnju bi značajno pomogao budući da se ne bi moralo čekati na stvarne događaje koji se potom istražuju. Na taj način uvježbavanje bi se moglo provoditi brže i u kontroliranoj okolini.
U sklopu ovog diplomskog rada potrebno je izraditi prototip sustava za uvježbavanje analitičara za kibernetičke prijetnje te demonstrirati njegov rad na jednom scenariju otkrivanja informacija o kibernetičkoj prijetnji. Identificirati probleme koji se mogu očekivati prilikom izrade proizvoda temeljenog na prototipu te ukazati na tehnologije koje bi bilo pogodno koristiti za implementaciju proizvoda. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Doc. dr. sc. Stjepan Groš

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Marko Čupić
Doc. dr. sc. Marko Čupić

SADRŽAJ

1. Uvod

U zadnja tri desetljeća digitalizacija podataka je dosegla vrlo visoku razinu. Danas je mnogo privatnih i osjetljivih podataka od tvrtki i pojedinaca spremljeno na internetu što ujedno stvara potencijalne prijetnje od krađe podataka. Konstantna napredovanja i promjene ponašanja prijetnji i njihovih izvođača stvorile su vrlo veliku potražnju za sigurnosnim mehanizmima koji bi zaštitili osjetljive podatke od njih [15].

Disciplina sigurnosti koju ovaj rad obrađuje je *Cyber threat intelligence* koja predstavlja područje kiber sigurnosti koje se odnosi na prikupljanje i analizu podataka o trenutnim i potencijalnim prijetnjama u svrhu zaštite organizacija i njihovih privatnih podataka. Cilj ove discipline je dobiti detaljan uvid u trenutno stanje sustava, definirati potencijalne prijetnje te prikupiti sve moguće informacije o njima. Ljudi koji koriste ovu disciplinu u svrhu zaštite računalnih sustava zovu se analitičari kibernetičkih prijetnji. Analitičar kibernetičkih prijetnji ili *Cyber threat analyst* se bavi obradom dobivenih podataka u korisnu inteligenciju preko koje se mogu poduzeti određeni sigurnosni koraci. Količina dostupnih podataka eksponencijalno raste iz dana u dan, a njihova obrada zahtjeva računalne resurse koji se ne mogu realizirati pa zbog toga analitičari moraju koristiti razne metode koje će usmjereno pretraživati podatke. Posljedično tome je iznimno teško napraviti realističnu simulaciju nekih prijetnji preko kojih bi se analitičar mogao educirati. Simulacija čitavih sustava i njihovo korištenje ne daju stvaran osjećaj količine dostupnih podataka na internetu.

Ovaj rad predlaže i implementira jedno rješenje za edukacijsku platformu za *Cyber threat* analitičare koja će koristiti stvarne sustave za simulaciju potencijalnih napada i pružiti realno iskustvo analize dobivenih podataka. U drugom poglavlju će se opisati kibernetička inteligencija, obraditi različite vrste i razine kibernetičke inteligencije te definirati teoretska pozadina problema. Treće poglavlje će opisati predloženi sustav za edukaciju analitičara kibernetičkih prijetnji i definirati korištene tehnologije. Programska implementacija obrađena je u četvrtom poglavlju uz prikaz isječaka programskog koda i opis uočenih problema prilikom implementacije. Testiranje napravljenog prototipa uz jednostavnu vježbu je prikazano u petom poglavlju.

2. Kiber inteligencija

U studenom 2018. godine je svjetski trošak na kibernetičke prijetnje procijenjen na vrtoglavih 600 milijardi dolara [1]. Razlog toga je što danas za izvesti napad nije potrebno znanje kao prije. Razvoj raznih malicioznih sofisticiranih alata je omogućio ljudima s manje ekspertize da efikasno izvedu napade. Iza tih napada je često financijska motivacija iako se sve češće pojavljuju politički razlozi ili jednostavno želje za ometanjem rada nekog sustava radi osobnog zadovoljstva.

Bivši direktor FBI-a *James Comey* tvrdi da danas postoje dvije vrste tvrtki u SAD-u. One koje su hakirane od strane Kine i one koje još ne znaju da su hakirane od strane Kine [16]. Takva tvrdnja osobe na tom položaju dokazuje veliku raširenost kibernetičkog kriminala i važnost analize i obrane od trenutnih i budućih napada. Upravo radi ogromne količine podataka na internetu te dinamičnog i nepredvidivog razvoja prijetnji, proces predviđanja potencijalnih napada je jako zahtjevan. Radi toga analitički tim ima vrlo težak zadatak i veliku odgovornost kod prikupljanja informacija o prijetnjama koje će se koristiti za pripremu obrane neke organizacije.

2.1. Postupci istraživanja i analize podataka

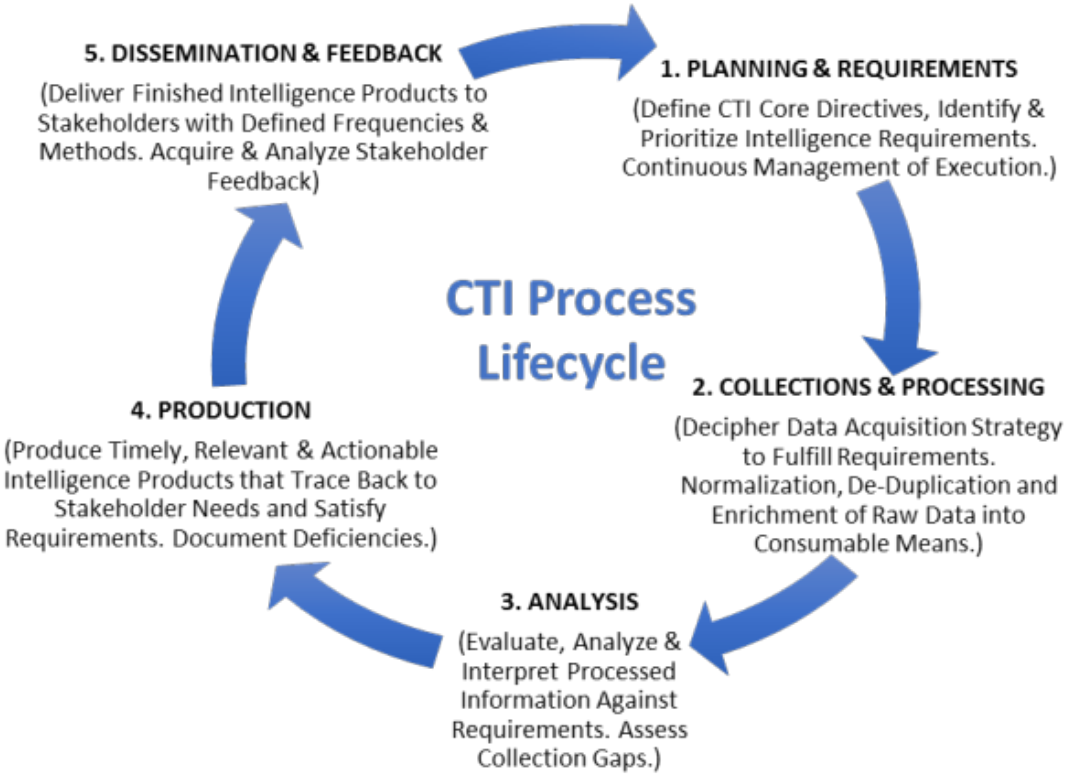
Običnom pretragom po internetu se nailazi na hrpe podataka, često nepovezanih i bez ikakvog posebnog značenja. Labeliranjem tih podataka dolazimo do informacija čijom obradom se dobije inteligencija preko koje se mogu donijeti određeni zaključci.

Zadatak tima analitičara je istražiti prijetnju te isporučiti izvještaj u kojem se donose zaključci dobiveni iz prikupljene inteligencije i mogući koraci koji se mogu poduzeti da se pojača obrana u tom kontekstu. Proces prikupljanja i obrade podataka se dijeli na pet glavnih dijelova (Slika 2.1) te se vrlo često prikazuje kao ciklična funkcija koja konstantno mora ažurirati dobivenu inteligenciju i prilagoditi se na nove prijetnje.

Planiranje i određivanje smjera pretraživanja uključuje definiranje glavnih vrijednosti organizacije, procjenu potencijalne štete koja se može dogoditi te traženje odgovora na pitanje "Zašto bi netko napao ovu organizaciju?". Djelotvorno definiranje ovih

koraka uvelike olakšava i smanjuje opseg prikupljanja podataka.

Faza prikupljanja podataka se provodi prema smjernicama iz prethodnog koraka. Izvori prikupljanja tih podataka mogu biti interni (mrežne log datoteke i slično) i vanjski (pretraživanje *deep* i *dark weba*). Podaci o prijetnjama obično budu u formatu lista raznih indikatora ugroženja sustava kao što su zloćudne IP adrese, domene i *hash* vrijednosti datoteka [13]. Dobivene sirove podatke je potrebno prikladno organizirati, sortirati i maknuti nepotrebne informacije. Radi ogromne količine podataka ovu fazu je nužno automatizirati stoga postoji mnogo programa koji vrše prikupljanje i razvrstavanje podataka s interneta. Jedno od poznatijih rješenja koje mnogo firmi koristi i implementira na različite načine je *SIEM* (engl. *Security information and event management*). Na slici 2.2 se može vidjeti graf koji pokazuje količinu podataka koji po završetku procesiranja budu spremni za analizu.

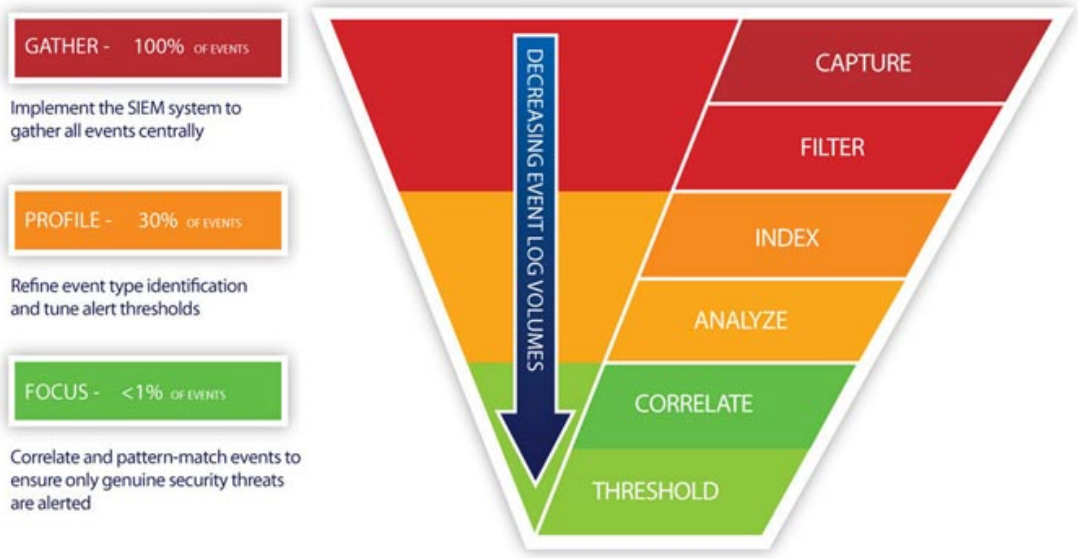


Slika 2.1: Životni vijek procesa prikupljanja podataka u kibernetičkoj sigurnosti [12]

Analizom se preko detaljne interpretacije i evaluacije informacija dolazi do određenih zaključaka o potencijalnim prijetnjama i trenutnim slabostima organizacije.

Rezultat analize je tehnički izvještaj koji detaljno opisuje problem i pronađenu inteligenciju u tom kontekstu, sve unutar zadanog vremenskog roka. Dovršeni izvještaj se distribuira klijentima prema dogovorenim vremenskim intervalima i metodama is-

poruke. Svaka povratna informacija se bilježi i uzima u obzir kod sljedeće iteracije svih navedenih faza.



Slika 2.2: Piramida količine obrađenih podataka u fazi prikupljanja [9]

2.2. Razine kiber inteligencija

Kiber inteligencija se može podijeliti na četiri različite razine odn. sloja koji se razlikuju po interpretaciji podataka i načinu njihovog prikaza. U nastavku su nabrojane i ukratko opisane sve četiri razine.

2.2.1. Strateška kiber inteligencija

Ova vrsta kiber inteligencije se prikuplja za glavne ljude u nekoj organizaciji u svrhu donošenja dugoročnih odluka oko zaštite sustava i stvaranja šire slike trenutnog sigurnosnog stanja. Prikupljanje podataka se izvodi preko praćenja modernih i geopolitičkih trendova te traženja potencijalnih obrazaca preko kojih se potencijalno može predvidjeti prijetnja. Čitav sadržaj ove inteligencije nije tehnički i predstavlja se u obliku izvještaja. Najčešći izvori informacija za ovu razinu kiber inteligencije su [13]:

- Državni politički dokumenti
- Vijesti iz lokalnih i državnih medija
- Konzultacije s ekspertima u području kiber sigurnosti
- Rezultati raznih istraživanja u ovom području, znanstveni radovi

2.2.2. Operativna kiber inteligencija

Glavni cilj operativne kiber inteligencije je predviđanje i otkrivanje potencijalnih vektora napada. Taj zadatak je izrazito težak zbog nepredvidivosti ljudskih radnji pa zbog toga većina pronađenih informacija dolazi iz privatnih izvora (doušnici, osobe koje su se infiltrirale u napadačku grupu i slično) iako u zadnje vrijeme neke informacije znaju procuriti na društvenim mrežama i raznim forumima. Napredne grupe će se jako teško otkriti, dok neke pobunjeničke i idealističke grupe su sklonije otkrivanju vlastitih planova preko javnih izvora. Iako to predstavlja dosta veliku prepreku kod provođenja ove vrste istraživanja, često se istražuju slični napadi i uzima u obzir činjenica da grupa ima ograničene resurse na raspolaganju pa će zbog toga koristiti slične metode ponovnih napada.

2.2.3. Tehnička kiber inteligencija

Analizu infrastrukture i alata koji napadač koristi radi tehnička kiber inteligencija. Tehničke informacije se prikupljaju uglavnom od provjerenih izvora koji dijele ovu vrstu inteligencije. Primjer je pristup listama malicioznih *hash* vrijednosti datoteka, e-pošte i njihovog sadržaja kod *phishing* napada, IP adresa malicioznih servera i slično. Zato što istražuje specifične napade, ova vrsta inteligencije se često pribraja operativnoj kiber inteligenciji. Unatoč tome tehnička kiber inteligencija obično ima kraći životni vijek upotrebljivosti od ostalih razina radi dinamičkog ažuriranja malicioznih programa.

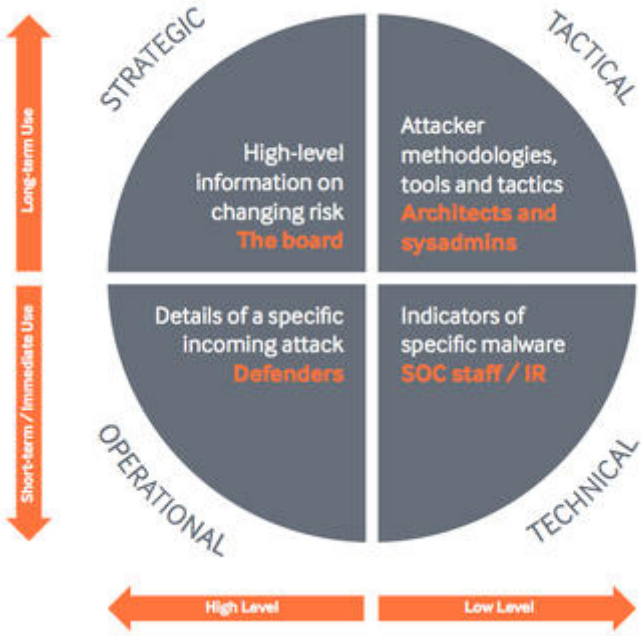
2.2.4. Taktička kiber inteligencija

Taktička kiber inteligencija definira detaljne informacije oko tehnika i procedura koje napadač poduzima pri napadu te obavještava tim zadužen za sigurnost (sistemski arhitekti, administratori i slično). Prikupljena inteligencija je namijenjena za tehničko osoblje a govori o specifikacijama alata koji se koriste u napadu i identificira vektore napada i mete napada. Podaci se skupljaju iz mnogo izvora uključujući:

- Javno dostupne podatke
- Podatke sa *deep* i *dark weba*
- Analizu *malwarea*
- Podatke od interakcija s ljudima

Učinkovita taktička kiber inteligencija omogućava kvalitetan odabir potrebnih koraka da bi se šteta sanirala ili u potpunosti izbjegla.

Na slici 2.3 se vidi grafički prikaz i kategorizacija razina kiber inteligencije.



Slika 2.3: Razine kiber inteligencije [11]

2.3. Tipovi kiber inteligencija

U poglavlju 2.2 inteligencija se dijelila po načinu interpretacije i formatu prikaza. U ovom poglavlju su obrađeni različiti izvori podataka te načini njihovog prikupljanja. U nastavku su nabrojeni tipovi kiber inteligencije koji se najčešće koriste.

2.3.1. HUMINT

Ovaj tip prikuplja podatke preko interakcije s ljudima. Podaci se prikupljaju kroz razgovore, intervjue ili ispitivanja ljudi za koje se vjeruje da posjeduju kritične informacije o trenutnom istraživanju. Vjerodostojnost informacija ovakvog tipa može biti jako upitna, pa je zato potrebno verificirati dobivene podatke nekim drugim metodama. Često je ljudski faktor ulazna točka kod napada pa zbog toga informacije koje osumnjičeni ili žrtve daju mogu dovesti do većih pomaka u istrazi. Najčešći izvori informacija su [10]:

- Savjetnici za kiber sigurnost
- Špijunski podaci

- Žrtva napada
- Osumnjičeni za napad

Za učinkovito prikupljanje ovih vrsta podataka je potrebna intuicija za razumijevanje ljudi i mogućnost analize ljudskog ponašanja [2]. Uz te vještine je moguće brzo i pouzdano pronaći izvor ulazne točke i spriječiti širenje napada i veću štetu.

2.3.2. SIGINT

Signals intelligence se orijentira na prikupljanje podataka preko presretanja signala. Signali mogu predstavljati bilo kakav oblik komunikacije između entiteta pa samim time u ovaj tip prikupljanja podataka spada i prisluškivanje razgovora između dvoje ljudi. Ova disciplina se počela primjenjivati već u Prvom svjetskom ratu kada su saveznici iz Antante prisluškivanjem njemačkih telegrama dobili potrebnu prednost te u konačnici izašli kao pobjednici [19]. Presretanje i dekriptiranje poruka im je dalo uvid u trenutno stanje neprijateljske vojske i njihov smjer kretanja. Upravo to je bio jedan od glavnih razloga britanske pobjede nad njemačkim podmornicama u Sjevernom moru 1915. godine. SIGINT se nastavlja koristiti i biti od važne uloge kako u Drugom svjetskom ratu tako i u modernom dobu u borbi protiv terorizma i kiber terorizma.

2.3.3. OSINT

Akronim *OSINT* označava inteligenciju dobivenu od javno dostupnih podataka. Izvori javno dostupnih podataka su obično razni mediji (novine, radio, televizija), javno dostupni podaci od vlade, tehnička literatura, internet. Upravo zbog velike količine izvora preko javno dostupnih podataka se može izvući mnogo više korisnih informacija koje će se poslije analizirati i pretvoriti u upotrebljivu inteligenciju. Prednost svega ovoga je ujedno i problem zbog toga što eksplozijom informacija je ujedno porastao i broj neistinitih podataka. Unatoč tom problemu, prikupljanje javno dostupnih podataka je jedna od najviše korištenih disciplina kod analize potencijalnih sigurnosnih prijetnji te upravo zato je fokus na ovom tipu kiber inteligencije u ovome radu. Vrlo je važno napomenuti da kod neke istrage često nema dovoljno ljudskih resursa da se mogu angažirati analitičari u svim tipovima inteligencije. U tom slučaju se radi evaluacija problema te definiranje polaznih točaka i eliminacija disciplina koje će biti od manjih koristi u istrazi.

Prototip napravljen u ovom radu se fokusirao na prikupljanje podataka iz javno dostupnih izvora odn. *OSINT* tipa kiber inteligencije. Sva prikupljena inteligencija se

temeljila na opisivanju trenutne prijetnje i analizu napada koji se dogodio. Takva vrsta inteligencije je operativna kiber inteligencija kao što je opisano u potpoglavlju 2.2.2.

2.4. Analitičar kibernetičkih prijetnji

Pojam *Threat intelligence* se pojavljuje puno prije početka informatičkog doba. Pristup ključnim informacijama je u prošlosti odlučivalo pobjednike bitki i ratova a prikupljale su se preko špijuna (potpoglavlje 2.3.1), presretanja signala (potpoglavlje 2.3.2) i međusobnih ljudskih interakcija. Sve prethodno navedene tehnike su se prebacile na internet s njegovom pojavom.

Analizu potencijalnih prijetnji na mreži odrađuju kiber analitičari. Njihova uloga je preko prikupljanja ključnih informacija o trenutnoj istrazi iz svih dostupnih izvora, digitalnom forenzikom i praćenjem trendova na mreži identificirati i pratiti prijetnje organizaciji te poduzeti potrebne korake za obranu od njih. Proces analize se ponekad može staviti u kontekst s umjetnošću radi baratanja s vrlo divergentnim podacima te rješavanja kompliciranih i raznovrsnih problema. Samim time svaki analitičar kibernetičkih prijetnji treba imati kreativnost i inicijativu kojom efektivno definira trenutne i buduće probleme te rješava ih relativno brzo. Veliki udio u ovom području igra i količina tehničkog znanja zbog toga što se pri samoj analizi pristupa raznim alatima i raznim programskim jezicima koji se teško razumiju bez nekakve pozadine. Kombinacijom ovih vještina osoba, zajedno sa dobrim analitičkim sposobnostima, postaje potencijalni kandidat za kompetitivnog analitičara kibernetičkih prijetnji.

Zbog kompleksnosti područja posao analitičara kibernetičkih prijetnji nije namijenjen za osobe koje nisu imali doticaja s tom temom. Tipično se od kandidata za posao očekuje diploma u području računarske znanosti uz solidno iskustvo u tehničkim područjima kao što su [17]:

- Sigurnost mreža
- Sigurnost operacijskih sustava
- Poznavanje sustava za otkrivanje i sprječavanje upada na računalne mreže
- Poznavanje metodologija napada i napadačkih alata

Osim tehničkog znanja, dobar analitičar mora znati točno prepoznati obrasce kod potencijalnih prijetnji, znati razmišljati kao napadač, dobro uočavati detalje, biti organiziran, dobro funkcionirati u timu i imati dobre vještine pisanja izvještaja te još mnogo toga.

Navedene vještine se stječu praktičnim iskustvom. Radi toga je pravilna i detaljna edukacija analitičara iznimno bitna za efikasnu obranu od trenutnih i budućih napada.

2.5. Edukacija analitičara kibernetičkih prijetnji

Istraživanjem dostupnih podataka s interneta može se naći više *online* tečaja za analizu kibernetičkih prijetnji. Većina tečaja se odvija preko gledanja video materijala koji opisuju metode prikupljanja i obrađivanja podataka no nemaju puno praktičnog pristupa. Neki tečaji kao praktični dio nude mogućnost istraživanja unaprijed pripremljenih podataka s mrežnih *log* datoteka, podataka iz radne memorije i slično [8]. Iako mogu napraviti dobre rezultate, takvi zadaci ne pružaju polazniku iskustvo pronalaženja, identificiranja i izdvajanja relevantnih informacija iz gomile na internetu.

Neke radionice za kibernetičku sigurnost koriste simulirane sustave za provođenje vlastitih vježbi. Ovakav pristup pruža polazniku mogućnost da sam traži podatke no generiranje vjerodostojnih podataka koji će simulirati stanje stvarnog svijeta je jako kompleksan zadatak.

Upravo se nakon sudjelovanja na radionici *Cyber Europe* stvorila ideja za edukacijsku platformu koja može pružiti stvarnije iskustvo prikupljanja podataka i analize informacija. Fokus edukacijske platforme bi bio na praktičnom pristupu i stjecanju iskustva preko direktnog dodira sa sustavima iz stvarnog svijeta. Predloženi sustav koji ima tu funkcionalnost je opisan u nastavku.

3. Prototip predloženog rješenja

Idejno rješenje simulacije sustava zasniva se na principu ubacivanja relevantnih komponenti vježbe korisniku u određenim trenucima. Da bi se ubacivanje izvršilo korisnik treba napraviti jednu od unaprijed definiranih radnji. Samo ubacivanje se izvršava preko posebnog *proxy* poslužitelja koji čita korisnikove zahtjeve i mijenja odgovore od ostalih poslužitelja. Dakle uloga *proxy* poslužitelja u ovom diplomskom radu konkretno je provjeriti svaki korisnikov zahtjev, po potrebi ga preusmjeriti te po potrebi izmijeniti sadržaj odgovora u skladu sa zadanom vježbom. U ovom radu se koristio javno dostupan alat *mitmproxy* koji pruža mogućnost jednostavnog postavljanja i korištenja traženog *proxy* poslužitelja.

3.1. Mitmproxy

Mitmproxy je besplatan interaktivan *HTTPS* "*man-in-the-middle*" *proxy* poslužitelj s javno dostupnim kodom koji pruža mogućnost presretanja *HTTP* i *HTTPS* mrežnog prometa, kod potonjeg preko falsificiranja *SSL* certifikata [6]. Nudi mnoštvo različitih funkcionalnosti a neke od njih su:

- Dinamično modificiranje *HTTPS* tokova
- Spremanje mrežnog prometa
- Ponovno slanje klijentskih zahtjeva
- Ponovno slanje poslužiteljskih odgovora
- Mod obrnutog *proxy* poslužitelja koji preusmjerava promet prema određenom poslužitelju
- Transparentni mod na *Linux* i *OSX* operacijskim sustavima
- Podrška za pisanje dodataka koji izvršavaju proizvoljne funkcionalnosti

Unutar samog projekta postoje tri vrste alata koji na različite načine izvode zajedničku funkcionalnost.

Tri alata koja se koriste u mitmproxy projektu su:

- *mitmproxy*
- *mitmdump*
- *mitmweb*

Mitmdump preko komandne linije pruža mogućnost analize, filtriranja, ponavljanja i spremanja otkrivenog mrežnog prometa. Svaka podržana funkcionalnost alata se provodi preko naredbi u komandnoj liniji. Kao primjer korištenja je u nastavku navedena naredba:

```
>> mitmdump -w izlazna_datoteka
```

Gornja naredba pokreće *mitmdump* alat te u datoteku *izlazna_datoteka* sprema sav uhvaćeni mrežni promet.

Mitmweb je inačica *mitmproxy* alata s korisničkim sučeljem. Pruža mogućnosti dinamične analize i izmjene odlaznog i dolaznog mrežnog prometa. Razlikuje se od *mitmdump* alata po tome da je sav promet spremljen u radnoj memoriji što znači da je namijenjen za uzimanje manjih skupova podataka [5]. Primjer izgleda sučelja je prikazan na slici 3.1.

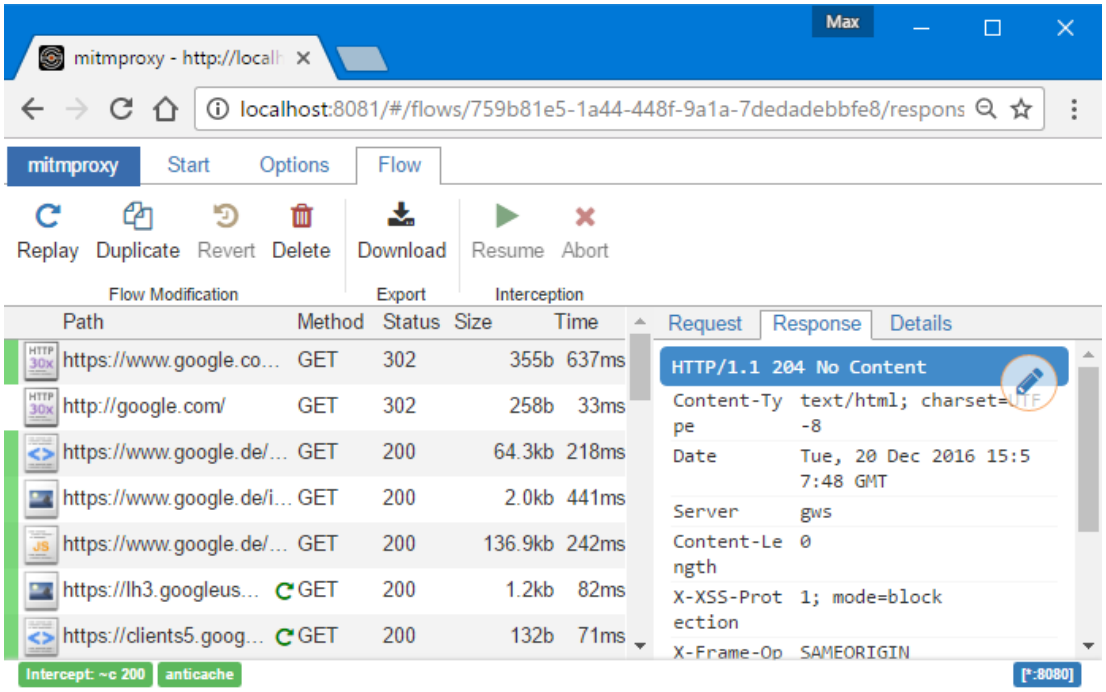
Mitmproxy, kao glavni alat u paketu, preko zasebne konzole obavlja istu funkcionalnost kao i *mitmweb* no sadrži više mogućnosti kod obrade podataka. U ovom radu je korišten ovaj alat za brzo presretanje i izmjenu tokova zbog transparentnog rada u pozadini te mogućnosti za pisanje programskih dodataka.

3.1.1. Programska podrška za pisanje dodataka

Podrška za pisanje dodataka se sastoji od nekoliko sučelja za programiranje aplikacija koja podržavaju više razina složenosti napisanih skripti. Preko tih sučelja skripta može komunicirati sa *mitmproxy* alatom te izvesti određene akcije u točno definiranim trenucima.

Skripta komunicira s *mitmproxy* alatom preko *događaja*. *Događaj* je ostvaren kao funkcija koja se pokrene svaki puta kada korisnik ili poslužitelj naprave određenu radnju. Na primjer jedni od najčešće aktiviranih *događaja* u ovom radu su bili: korisnik šalje zahtjev prema poslužitelju, poslužitelj odgovara korisniku, pokretanje skripte i slično.

Jedini programski jezik u kojem je podržano pisanje dodataka je *Python* [7]. *Mitmproxy* u svom paketu za podršku za pisanje dodataka sadrži i nekoliko izvršnih datoteka



Slika 3.1: *Mitmweb* sučelje [5]

koje mogu pokrenuti napisanu skriptu dodanu kao argument u naredbi. Izvršne datoteke koje su trenutno napravljene su:

- *mitmproxy*
- *mitmdump*
- *mitmweb*
- *pathod*
- *pathoc*

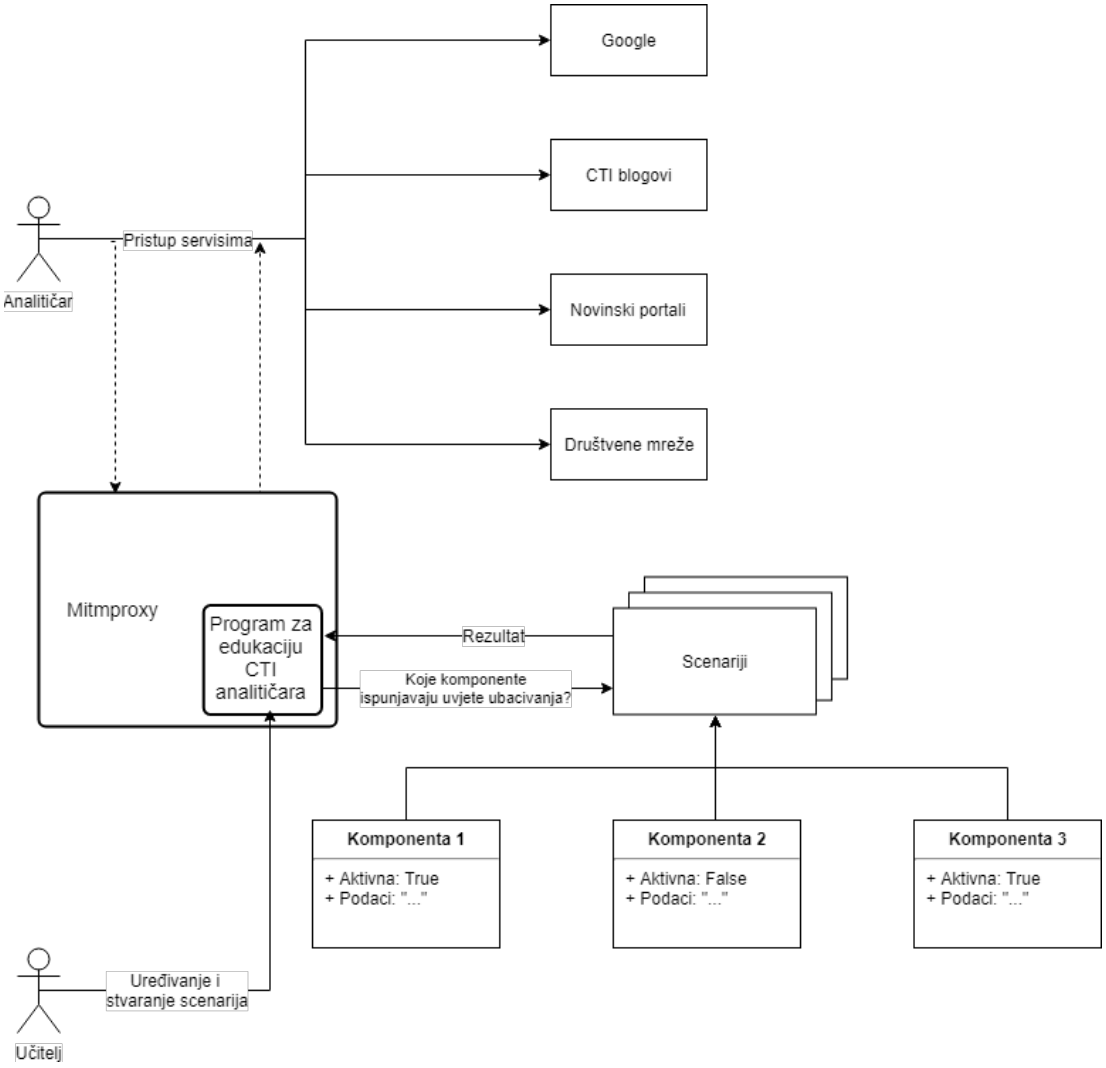
Osim prve tri izvršne datoteke koje su navedene u poglavlju 3.1, *pathod* i *pathoc* simuliraju ekstremne verzije aplikacija na kojima se može vršiti razgovor između klijenta i poslužitelja koji bi inače kršio obične standarde [7].

U ovom radu se kod pokretanja skripte koristila izvršna datoteka *mitmproxy* uz zastavicu kojom se omogućava učitavanje programskih dodataka. Naredba za pokretanje je navedena u nastavku.

```
>> mitmproxy -s addons/ime_dodatka
```

3.2. Scenario vježbe

Scenario vježbe u ovom radu označava skupinu komponenti koje u određenom vremenskom roku utječu na stvarne sustave tako da se izvrše akcije koje te aktivne komponente opisuju. Komponente mogu označavati ubacivanje određenog rezultata pretraživanja u google, ubacivanja specifičnog bloga na stranicu, izmjenu profila na društvenoj mreži i slično. Svaka vježba ima svoj period aktivnosti kojim aktivira ili deaktivira sve komponente sadržane unutar njezine strukture. Prilikom svakog *HTTP* ili *HTTPS* zahtjeva alat *mitmproxy* preko programa za edukaciju ispituje postoji li komponenta u nekom scenariju koja ispunjava uvjete ubacivanja. U slučaju da postoji jedna ili više komponenti koje zadovoljavaju sve uvjete, za svaku pojedinačno se izvršava proces ubacivanja komponente u sustav kojem je korisnik pristupio. Na slici 3.2 je prikazana arhitektura opisanog sustava.



Slika 3.2: Arhitektura sustava za edukaciju analitičara kibernetičkih prijetnji

Alat *mitmproxy* radi transparentno u odnosu na korisnika. Korisnik iz svoje perspektive na normalan način otvara određenu stranicu ali sadržaj koji mu se prikaže može se znatno razlikovati od sadržaja koji bi se prikazao za tu stranicu u slučaju da alat nije pokrenut. *Mitmproxy* može izmijeniti i zahtjev prema poslužitelju i odgovor od poslužitelja. Funkcionalnost programa je temeljena na ispravnoj reakciji na pojedini korisnikov zahtjev.

Jednu vježbu u programskom smislu opisuje jedna datoteka (poglavlje 4.1). Svaka vježba sadrži vlastito razdoblje aktivnosti te popis komponenti koje čine samu vježbu odn. scenarij.

3.2.1. Stvaranje vježbe

Vježbu stvara i uređuje *scenario leader* odn. učitelj. Njegova zadaća je stvoriti priču koja opisuje jedan ili više malicioznih napada ili potencijalnu prijetnju koju je potrebno istražiti. Stvorena vježba ili scenario se sastoji od komponenti (poglavlje 3.2) koje učitelj mora definirati. Definirane komponente će prikazivati informacije o priči samo korisniku koji koristi program za edukaciju. Informacije mogu ali i ne trebaju biti korisne za samu istragu uzimajući u obzir da u realnoj situaciji postoji puno lažnih vijesti i informacija. Korisni dio informacija treba biti raspodijeljen tako da korisnik ne primijeti razliku između ubačene i stvarne informacije na sustavu.

Svakoj definiranoj komponenti učitelj zadaje vremenski interval u kojem će ona postati aktivna. Osim vremenskog intervala komponenti se zadaje platforma, podaci koji se ubacuju, tip komponente, lista komponenti roditelja te informacije koje definiraju dodatne uvjete ubacivanja. Platforma definira ime poslužitelja na kojem ta komponenta može biti aktivirana. Lista komponenti roditelja predstavlja niz komponenti koje moraju biti aktivirane odn. ubačene prije nego što se trenutna komponenta ubaci. Komponenta će biti ubačena samo ako je aktivna i ako ispunjava sve uvjete zadane unutar same komponente.

U radu su implementirana dva tipa komponente: *inject* i *redirect*. *Inject* mijenja dio odgovora od poslužitelja koji je programski definiran s podacima komponente dok *redirect* preusmjerava zahtjev korisnika prema poslužitelju definiranom u komponenti. Prilikom preusmjeravanja se svi dodatni parametri koje je korisnik zadao u zahtjevu zadržavaju.

3.3. Korištenje programa

Prije početka korištenja programa potrebno je napraviti nekoliko koraka kod postavljanja okruženja. Kao što je napisano u poglavlju 3.1 ovaj rad koristi već gotov *proxy* poslužitelj kojeg je potrebno preuzeti i postaviti na vlastito računalo. Osim instalacije *mitmproxy* alata potrebno je imati postavljen programski jezik *Python* verzije 3.0 ili više.

Program nakon pokretanja radi u pozadini te nije potrebna nikakva dodatna interakcija s njim. Program se pokreće iz komandne linije naredbom:

```
mitmproxy --no-https2 -s addons/cti_analyst_education_platform.py
```

Detaljnije upute oko instalacije okruženja su opisane u dodatku A.

4. Programsko ostvarenje sustava

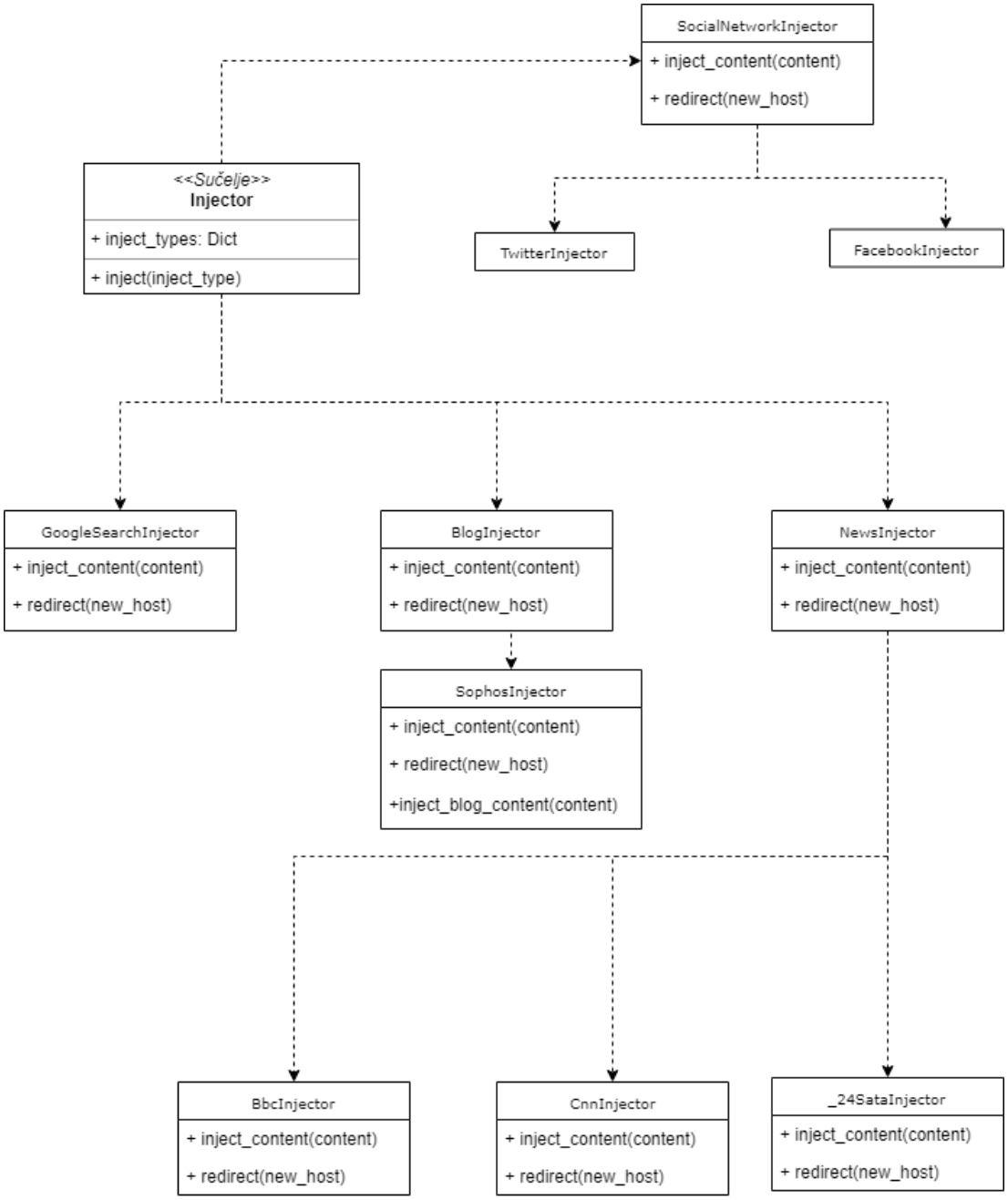
Prototip je napisan u potpunosti u programskom jeziku *Python* u obliku programskog dodatka za alat *mitmproxy*. Rješenje se sastoji od glavnog modula koji se pokreće uz *mitmproxy* alat te nekoliko međusobno nezavisnih razreda koji opisuju različite vrste ubacivanja podataka. Struktura tih razreda je prikazana na slici 4.1.

4.1. Programsko stvaranje vježbe

Svaka vježba je opisana vlastitom datotekom u kojoj se nalaze sve njezine komponente koje se ubacuju u sustav. Format zapisa tih komponenti je *JSON*. Zapis početka vježbe i jedne komponente izgleda ovako:

```
{
   "exercise_start":"2019-01-02T18:00:30",
   "exercise_end":"",
   "prvi_inject": {
      "platform": "sophos",
      "type": "inject",
      "data": "<article id=\"post-446488\" ...",
      "keywords":"article",
      "parent":"",
      "active_interval_start": "2019-04-05",
      "active_interval_end": ""
   },
   "drugi_inject":{
   ...
   },
   ...
```

Parametri *exercise_start* i *exercise_end* označavaju vremenski interval aktivnosti čitave vježbe. Ako vježba nije aktivna nijedna njezina komponenta neće biti ubačena, neovisno o stanju aktivnosti same komponente. Parametri samog *injecta* su zapisani



Slika 4.1: Programska struktura prototipa

kao ugniježđeni objekt a detaljno su opisani u poglavlju 3.2.1.

Svaka vježba će prilikom pokretanja programa biti učitana u listu u glavnoj skripti. Ovisno o tome da li je scenario aktivan ili ne zapisuje se u varijablu koja sadrži aktivne scenarije odn. varijablu koja sadrži arhivirane scenarije. Prilikom rada samog programa scenario se može prebaciti iz arhiviranog u aktivnog ili obrnuto ako uđe ili izađe iz perioda aktivnosti zadanog s vlastitim parametrima.

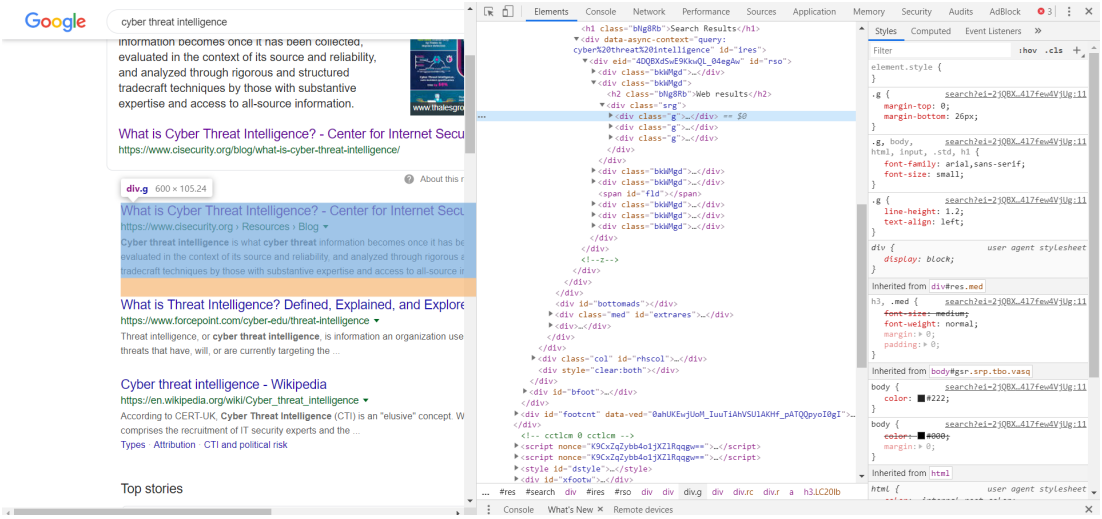
4.1.1. Definiranje podataka komponenti

Glavni dio svake komponente u scenariju su podaci koji se ubacuju u nekom trenutku u sustav. Ti podaci su u *HTML* obliku s ciljem oponašanja identične strukture odgovora poslužitelja u koji se taj sadržaj ubacuje. Za identifikaciju relevantnih komponenti kod poslužitelja se koristila funkcionalnost internet pretraživača *inspect*. Pomicanjem miša preko bloka podataka u sadržaju odgovora se označi dio na prikazu koji je taj blok definirao. Koristeći te funkcionalnosti je moguće lako pronaći i preuzeti dio sadržaja koji će se poslije ponovo koristiti u druge svrhe. Navedeni postupak je za primjer identificiranja komponente koja predstavlja rezultat pretrage na *Google* servisu prikazan na slici 4.2.

Definirana komponenta se kopira i spremi lokalno za potrebne izmjene. Pazeći da se struktura komponente ne dira, izmjenjuje se sadržaj komponente tako da se mijenjaju poveznice, naslovi, paragrafi i ostale vrste čistoga teksta. Radi olakšane izmjene tih dijelova se koristio već gotovi alat za uređivanje *HTML* sadržaja *html-online*. Nakon potrebnih izmjena sadržaja se tekst pomoću istog alata kompresira tako da bude u jednom retku te se uz pomoć alata *freeformatter* izbjegnu svi posebni znakovi koji mogu izazvati greške kada se sadržaj stavi u *json* datoteku. Zadnji korak je kopiranje tog retka u *json* datoteku željenog scenarija pod ključem "*data*".

4.1.2. Ispitivanje stanja aktivnosti komponenti

Aktivnost komponente i scenarija je određena početnim i završnim vremenom aktivnosti zadanom u *json* datoteci gdje su opisane pojedine vježbe. Ispitivanje da li je određena komponenta aktivna se definira usporedbom tih parametara s trenutnim vremenom. Samim time moguće su situacije gdje komponenta promijeni stanje aktivnosti tijekom izvođenja programa. Taj problem se riješio tako da se prilikom svake obrade odgovora poslužitelja ispituje koje su komponente trenutno aktivne. Osim samog vremenskog intervala, aktivnost komponente određuje lista komponenti roditelja koje se



Slika 4.2: Postupak identifikacije komponenti za ubacivanje

moraju izvršiti da bi trenutna komponenta bila aktivna. Izvorni kod koji ispituje aktivnost komponente je sljedeći:

```
def is_inject_active(self, exercise, inject):
  time_start = self.get_datetime_object
      (self.exercises[exercise][inject]["active_interval_start"])
  if time_start is None:
    time_start = datetime.datetime.min
  time_end = self.get_datetime_object
      (self.exercises[exercise][inject]["active_interval_end"])
  if time_end is None:
    time_end = datetime.datetime.max
  time_today = datetime.datetime.today()
  parent_condition = self.are_parents_activated(exercise, inject)
  return parent_condition and time_today > time_start and time_today
      < time_end
```

Varijabla *parent_condition* je zastavica koja poprima istinitu vrijednost ako su sve komponente roditelji već izvršeni. Metoda koja ispituje taj uvjet učitava listu svih komponenti roditelja od zadane komponente te za svaki od njih ispituje da li je već izvršen. Ako su svi izvršeni vraća se istinita vrijednost dok se u suprotnom vraća neistinita vrijednost. Izvorni kod za tu metodu je:

```
def are_parents_activated(self, exercise, inject):
  parent_injects = self.exercises[exercise][inject]["parent"]
  if not parent_injects:
```

```
    return True
  parent_injects = [x.strip() for x in parent_injects.split(",")]
  for parent_inject in parent_injects:
    if parent_inject not in self.activated_injects:
      return False
  return True
```

4.1.3. Dohvat aktivnih komponenti

Dohvat aktivnih komponenti se svodi na popunjavanje liste koju čine sve aktivne komponente. Dodatan uvjet dohvata je da poslužitelj za koju je komponenta definirana mora odgovarati trenutnom poslužitelju kojem se pristupa. Metoda koja dohvaća listu aktivnih komponenti izgleda ovako:

```
def get_all_available_injects(self, hostname):
  for exercise in self.active_exercises:
    for inject in self.exercises[exercise].keys():
      if type(self.exercises[exercise][inject]) is dict and
          self.exercises[exercise][inject]["type"] == "inject":
        if (self.exercises[exercise][inject]["platform"] == hostname)
            and self.is_inject_active(exercise, inject):
          self.available_injects.append((exercise, inject))
  return self.available_injects
```

4.2. Glavna skripta

Glavna skripta se pokreće prilikom pokretanja alata *mitmproxy* preko komandne konzole kao dodatni argument (poglavlje 3.1.1). Njezina uloga je obrađivanje relevantnog dolaznog i odlaznog prometa po pravilima definiranim unutar same skripte i uvjetima iz napravljenih vježbi.

4.2.1. Učitavanje vježbi

Instanciranjem skripte se inicijaliziraju početne vrijednosti varijabli glavne klase. Nakon instanciranja glavne klase prva metoda koja se pozove je metoda *load*. Automatsko pozivanje ove metode nakon učitavanja dodatka je funkcionalnost alata *mitmproxy* [4]. Sadržaj *load* metode je sljedeći:

```
def load(self, _):
  PATH = "%s/addons/cti_json" % os.getcwd()
  exercises_data = [f for f in os.listdir(PATH) if isfile(join(PATH,
      f))]
  for ex in exercises_data:
    ex_name = ex.split(".")[0]
    with open("%s/%s" % (PATH, ex)) as exercise:
      try:
        self.exercises[ex_name] = json.load(exercise)
      except Exception:
        raise("Failed to load json data!")
  self.active_exercises = self.get_all_active_exercises()
```

Metoda učitava sve *json* datoteke iz direktorija gdje su vježbe smještene u rječnik glavnog razreda *all_exercises*. Svaka pročitana vježba predstavlja jedan unos u rječnik. Ključ svakog unosa je ime same vježbe. Metodom *get_all_active_exercises*() se dohvaćaju sve aktivne vježbe iz prethodno popunjenog rječnika.

4.2.2. Obrada korisničkog zahtjeva

Prilikom svakog *HTTP* ili *HTTPS* korisničkog zahtjeva se poziva metoda *request*. Parametar te metode *flow* sadrži razne informacije o zahtjevu kao što je ime poslužitelja kojem je zahtjev upućen, sadržaj zahtjeva, odredišni *port* i slično. U metodi se obrađuje zahtjev tako da se ispita da li je odredišni poslužitelj trenutno podržan od strane dodatka za ubacivanje sadržaja. U potvrdnom slučaju se trenutni poslužitelj sprema u atribut razreda te se ispituje treba li izvršiti preusmjeravanje sadržaja odn. postoji li aktivna komponenta koja opisuje preusmjeravanje sa zadanog poslužitelja na neki drugi. Ako je taj uvjet istinit komunikacija se presreće te se izvodi preusmjeravanje na odredišni poslužitelj definiran s aktivnom komponentom. U slučaju da prethodni uvjet nije ispunjen tada se metoda završava. Izvorni kod metode je dan u nastavku:

```
def request(self, flow):
  self.host = self.is_host_supported(flow.request.host)
  if self.host is not None and len(self.active_exercises) != 0:
    redirect_action = self.get_active_redirect(self.host)
    if redirect_action is not None:
      flow.intercept()
      self.injector = self.new_injector(self.host, flow)
      self.injector.redirect(flow, None, redirect_action)
      flow.resume()
  return flow
```

4.2.3. Obrada odgovora poslužitelja

Obradu odgovora od poslužitelja radi metoda *response*. Kao i kod obrade korisničkog zahtjeva metoda prvo provjerava da li je poslužitelj koji trenutno sudjeluje u komunikaciji podržan od strane programa. Osim tog uvjeta metoda će nastaviti s izvođenjem samo ako u odgovoru postoji sadržaj te ako postoji barem jedna aktivna vježba. Kada su svi uvjeti ispunjeni metoda sadržaj odgovora pretvara u standardizirani *HTML* oblik preko kojeg se obavljaju potrebne manipulacije. Pretvorba se odrađuje preko vanjske *python* biblioteke *BeautifulSoup*. U slučaju da sadržaj nije u HTML obliku javlja se greška te metoda završava i program normalno nastavlja raditi.

U sljedećem koraku se dohvaćaju sve moguće akcije ubacivanja sadržaja za trenutnog poslužitelja. U slučaju da postoji aktivna akcija stvara se nova prikladna instanca *Injector* klase, izvršava se akcija ubacivanja te se zapisuje u listu jedinstvenih objekata *activated_injects*.

Metoda *reset_values* je zadužena za brisanje privremenih sadržaja nekoliko atributa klase. Pojedine funkcije pozivane u ovoj metodi su obrađene u poglavljima koji slijede. Izvorni kod metode je sljedeći:

```
def response(self, flow):
  self.host = self.is_host_supported(flow.request.host)
  if self.host is not None and len(flow.response.get_text()) > 0 and
      len(self.active_exercises) != 0:
    flow.intercept()
    body_exists = True
    try:
      soup = BeautifulSoup(flow.response.get_text(), 'html.parser')
      except ValueError as e:
```

```
9       body_exists = False
10      flow.resume()
11      return flow
12    self.available_injects =
        self.get_all_available_injects(self.host)
13    if body_exists:
14      if len(self.available_injects) != 0:
15        self.injector = self.new_injector(self.host, flow)
16        for exercise, inject_name in self.available_injects:
17          inject_result = self.injector.inject(flow, soup,
              self.exercises, exercise, inject_name)
18          if inject_result is not None:
19            self.activated_injects.add(inject_name)
20    self.reset_values()
21    flow.resume()
22  return flow
```

4.3. Sučelje Injector

Sučelje *Injector* u trenutnoj verziji prototipa sadrži samo jedan atribut koji se fiksno postavlja odmah kod instanciranja te dvije definirane metode. Vrijednost atributa su svi mogući tipovi ubacivanja sadržaja tipa *string*. Glavna funkcionalnost ovog sučelja je stvaranje apstraktnog sloja poveznice između svih klasa koje ubacuju sadržaj. Poveznica se definirala preko metode inject koja se zove iz metode response (poglavlje 4.2.3, linija koda 17). Cilj metode je dinamički pozvati metodu prikladne klase. Preko atributa klase se dolazi do imena metode te se preko ugrađene funkcije *getattr* poziva metoda potklase definirana danim parametrima.

```
1  def inject(self, flow, decoded_flow_body, exercises, exercise_name,
       inject_name):
2    inject_type = exercises[exercise_name][inject_name]["type"]
3    data = exercises[exercise_name][inject_name]["data"]
4    keywords = [x.strip() for x in
         exercises[exercise_name][inject_name]["keywords"].split(",")]
5    return getattr(self, self.inject_types[inject_type])(flow,
         decoded_flow_body, data, keywords)
```

Svaka klasa koja opisuje ubacivanje sadržaja na nekog poslužitelja nasljeđuje ovo sučelje.

Metoda *replace_content* ubacuje odn. zamjenjuje jednu stvarnu *HTML* komponentu s aktiviranom komponentom scenarija. Dodatni parametri metode, *attributes* i *inject_range* pružaju dinamično pozivanje te metode iz svih potklasa koje rade jednostavnu zamjenu sadržaja. Parametar *attributes* definira rječnik atributa koji određuje uvjete po kojim se traže elementi sadržaja. Neobavezan parametar *inject_range* označava na koje mjesto u redu pronađenih komponenti se vrši izmjena, doprinoseći dinamici prikaza relevantnog sadržaja korisniku te radu samog programa. Ako taj parametar nije definiran, mijenja se jedina odn. prva komponenta pronađena u sadržaju. Izvorni kod za metodu *replace_content* je:

```
def replace_content(self, flow, decoded_flow_body, data,
    attributes, inject_range=None):
  found_elements = decoded_flow_body.find_all(attrs=attributes)
  if inject_range is not None:
    j = random.randint(inject_range[0], inject_range[1])
    for i,tmp in enumerate(found_elements):
      if i == j:
        tmp.replaceWith(BeautifulSoup(data, 'html.parser'))
        break
      flow.response.content = str(decoded_flow_body).encode("utf8")
    elif len(found_elements) != 0:
      tmp = found_elements[0]
      tmp.replaceWith(BeautifulSoup(data, 'html.parser'))
      flow.response.content = str(decoded_flow_body).encode("utf8")
  return flow
```

Sučelje ne definira jednoznačno niti jednu drugu metodu ubacivanja ili manipulacije sadržaja radi raznovrsnosti prikaza tih sadržaja na internetu.

4.4. Klasa GoogleSearchInjector

Klasa *GoogleSearchInjector* definira ubacivanje sadržaja u pretraživanja preko *google* tražilice. Klasa nasljeđuje sučelje *Injector* te implementira vlastite metode ubacivanja sadržaja i preusmjeravanja prometa na drugi poslužitelj.

Inicijalizacija klase traži i sprema *query* ako on postoji u zahtjevu prema poslužitelju. Nakon inicijalizacije *query* atributa poziva se metoda inicijalizacije natklase. Izvorni kod koji to opisuje je sljedeći:

```
def __init__(self, flow):
  self.query = None
  if 'q' in flow.request.query.keys():
    self.query = flow.request.query['q']
  super().__init__(flow)
```

4.4.1. Ubacivanje rezultata pretraživanja

Metoda koja izvodi akciju ubacivanja rezultata pretraživanja je *inject_content*. Poziva se prilikom obrade odgovora od poslužitelja (poglavlje 4.2.3). U nastavku je prikazan izvorni kod te metode.

```
def inject_content(self, flow, decoded_flow_body, data, keywords):
  if self.check_query(keywords):
    return self.replace_content(flow, decoded_flow_body, data,
        attributes={"class": "g"}, inject_range=(0,4))
  return flow
```

Početni uvjet je ispitivanje sličnosti trenutne vrijednosti korisničkog upita sa svakom ključnom riječi definiranom unutar liste ključnih riječi trenutne komponente. Za ispitivanje sličnosti se koristi vanjska biblioteka *SequenceMatcher*. Dobivene sličnosti sa svakom pojedinačnom ključnom riječi se silazno sortiraju te se vraća istinita vrijednost ako je najveća sličnost veća od parametra 0.67 koji je određen eksperimentalnom metodom.

```
def check_query(self, keywords):
  if self.query is None:
    return False
  ratios = [(v, self.similarity(self.query.lower(), v.lower())) for
      v in keywords]
  sorted_ratios = sorted(ratios, reverse = True, key=lambda x: x[1])
  return sorted_ratios[0][1] > 0.67
```

Izračunavanje sličnosti je jednolinijska metoda koja izgleda ovako:

```
def similarity(self, a, b):
  return SequenceMatcher(None, a, b).ratio()
```

Metoda natklase *replace_content* odrađuje zamjenu sadržaja. Varijabla *j* poprima nasumičnu vrijednost od 0 do 3 te označava redno mjesto koji rezultat pretraživanja će biti zamijenjen. Granice vrijednosti parametra su također eksperimentalno određene radi silazne relevantnosti svakog sljedećeg rezultata koji se prikazuju korisniku.

Parametar *decoded_body_flow* je sadržaj odgovora poslužitelja obrađen preko *BeautifulSoup* paketa. Ugrađenom metodom *find_all* u ovoj metodi se pronalaze svi dijelovi *HTML* sadržaja koji su definirani s klasom "g". Analizom stvarnih *HTML* odgovora od *google* poslužitelja se utvrdilo da svaki segment rezultata pretraživanja počinje s tom komponentom u sadržaju.

Nakon što se pronađu svi rezultati pretraživanja jedan od prva 4 se zamjenjuje sa sadržajem definiranim parametrom *data*. Rezultat zamjene pohranjen u varijabli *decoded_body_flow* se preslikava u varijablu od stvarnog toka podataka tako da se izvrši enkodiranje podataka po standardu koji *Google* koristi te se pretvori u tip *string*. Vraćen tok podataka sada sadrži jedan ubačeni rezultat pretraživanja.

4.4.2. Preusmjeravanje korisničkog upita

Ako prilikom korisničkog upita prema *Google* poslužitelju se nađe aktivna akcija preusmjeravanja tada se poziva metoda *redirect* koja na jednostavan način preusmjerava korisnički upit na drugi poslužitelj. Preusmjeravanje se radi tako da se preko ugrađene *python* biblioteke *re* zamjenjuje trenutni poslužitelj s novim poslužiteljem koji je definiran parametrom *new_site* Prilikom preusmjeravanja se svi parametri upita zadržavaju pa tako sam upit se izvršava na drugom poslužitelju.

```
1 def redirect(self, flow, _, new_site):
2   flow.request.url = re.sub(r"www.google.com", r"%s" % new_site,
        flow.request.url)
3   return flow
```

4.5. Klasa NewsInjector

Namjena klase *NewsInjector* je ujedinjenje svih klasa koje se bave ubacivanjem sadržaja na novinske portale. Metode klase su privremeno prazne i namijenjene za implementaciju zajedničkih funkcionalnosti za novinske portale u budućnosti.

U prototipu je implementirana potklasa *_24SataInjector* koja ubacuje novinski članak u taj poslužitelj. Ubacivanje samog sadržaja je pojednostavljeno zbog dobre struk-

ture stvarnog odgovora od poslužitelja preko koje se jasno definiraju blokovi koji opisuju pojedini članak. Samim time ubacivanje drugog članka je jednolinijska naredba koja koristi zajedničku metodu definiranu u natklasi *Injector*.

```
def inject_content(self, flow, decoded_flow_body, data, keywords):
 return self.replace_content(flow, decoded_flow_body, data,
    attributes={"class": "card__article cf"}, inject_range=(0,4))
```

4.6. Klasa BlogInjector

Kao i u prethodnom potpoglavlju, klasa *BlogInjector* je postavljena za buduće implementiranje funkcionalnosti zajedničke za sve blogove.

U prototipu je implementirana potklasa *SophosInjector*. Za razliku od ubacivanja novinskih članaka ova klasa definira i ubacivanje čitavih blogova.

```
def inject_content(self, flow, decoded_flow_body, data, keywords):
 blog_url = None
 if len(keywords) > 1:
  blog_url = keywords[1]
 if re.match(r".*nakedsecurity.sophos.com\/$", flow.request.url)
    and "article" in keywords:
 return self.replace_content(flow, decoded_flow_body, data,
    attributes={"class": "card-article"}, inject_range=(0,2))
 elif blog_url and re.match(r".*nakedsecurity.sophos.com\/%s\/" %
    blog_url, flow.request.url) and "blog" in keywords:
  return self.inject_blog_content(flow, decoded_flow_body, data,
     keywords)
 else:
  return None
```

Lista *keywords* u ovom slučaju predstavlja dva elementa: prvi definira koja vrsta ubacivanja se mora izvesti a drugi definira fiksnu poveznicu na kojoj će ubacivanje biti odrađeno. Moguće su dvije vrste ubacivanja sadržaja: Ubacivanje blog članka i ubacivanje blog sadržaja. Te dvije akcije su međusobno povezane zato što se klikom na članak otvara poveznica koja vodi do ubačenog bloga. Poveznica koja je definirana drugim parametrom iz liste se ispituje u uvjetu za ubacivanje čitavog bloga. Metoda koja vrši ubacivanje čitavog bloga poziva zajedničku metodu kao i sve prethodne metode ubacivanja s parametrima koji su navedeni na kodu ispod:

```
def inject_blog_content(self, flow, decoded_flow_body, data,
    keywords):
  return self.replace_content(flow, decoded_flow_body, data,
      attributes={"class": "content-panel"})
```

4.7. Uočeni problemi kod razvoja

Prilikom istraživanja i implementacije prototipa uočeno je nekoliko potencijalnih prepreka u održavanju sustava. Jedan od glavnih problema je što sadržaj odgovora od poslužitelja nije statičan. Pretpostavka je da *HTML* odgovori neće imati uvijek istu strukturu pa kada se ona promijeni potrebno je ponovo analizirati strukturu i definirati blokove koji opisuju željene komponente koje će se izmijeniti. Osim pretpostavljene dinamične *HTML* strukture, neki poslužitelji sadržaj dohvaćaju iz više posrednika te znatno otežavaju manipulaciju tog sadržaja. U ovom radu se taj problem uočio kod pokušaja implementacije ubacivanja sadržaja u *CNN* novinski portal.

Kod implementacije ubacivanja sadržaja u *Facebook* društvenu mrežu se uočio problem višestrukog preklapanja *HTML* blokova koji označavaju željenu komponentu. Kompleksnost takvog sadržaja otežava detekciju *HTML* koda koji se treba zamijeniti. Konkretno na primjeru *Facebook* mreže se pokušao zamijeniti rezultat pretraživanja osobe na tražilici što je rezultiralo gubitkom dijela informacija zbog narušene strukture odgovora poslužitelja.

Posljednji problem je loše implementirana struktura odgovora kod poslužitelja. Iako kod implementacije prototipa ovaj problem nije izravno uočen, nailazak na takav tip sadržaja može predstavljati podjednako težak zadatak definiranja traženih komponenti kao i prethodno navedeni problemi.

Osim problema tehničke prirode treba napomenuti da kreiranje samog sadržaja jedne vježbe zahtjeva velike količine utrošenog vremena. Osim definiranja priče koja mora biti logički ispravno povezana potrebno je sadržaj ubaciti u samu vježbu. Izmjena *HTML* komponenti može sadržavati više dijelova koje treba izmijeniti za svaki zasebni slučaj pa zbog toga taj postupak nije moguće u potpunosti automatizirati.

Rješenje nabrojanih problema nije jednoznačno. Svaka vrsta prepreke na koju se naiđe se rješava na drugačiji način. Detaljnijom analizom se mogu riješiti problemi loše napisanih i kompleksnih struktura odgovora poslužitelja. Dinamičko mijenjanje struktura zahtjeva održavanje sustava u češćim vremenskim intervalima.

Problem stvaranja vježbi se svodi na problem nedostatka vremena koji se može riješiti povećanjem broja ljudi koji stvaraju te vježbe.

Problem dohvaćanja sadržaja s više različitih poslužitelja se ne može riješiti na jednostavan način. U toj situaciji se preporučuje napraviti analizu koliko je bitno taj specifični poslužitelj implementirati u projekt te pronaći poslužitelja koji nudi sličnu uslugu s jednostavnijim dohvatom odgovora. Zaobilaženje takvih poslužitelja se trenutno čini kao najbolje rješenje tog problema.

5. Simulacija vježbe

Testiranje ispravnosti implementiranog rješenja se provelo na jednom pojednostavljenom scenariju napada. Kao pozadinska priča vježbi se koristio stvaran napad od grupe *Koobface*.

5.1. Koobface grupa

"*Koobface Gang*" ili "*Ali baba & 4*" je bila grupa malicioznih programera koji su 2008. stvorili crva koji je napadao operacijske sustave *Windows*, *Linux* i *MacOS*. Meta napada su bile društvene mreže poput *Facebooka* i *Skypea* te servisi elektroničke pošte poput *Gmaila*, *Yahoo Maila* i slično. Napadi su bili aktualni u 2009. i 2010. godini, a odvijali su se tako da je osoba dobila od prijatelja na društvenoj mreži poruku s poveznicom na video. Ta poveznica je vodila na lažnu *YouTube* stranicu na kojoj se tražilo ažuriranje *Flash* programa. Pokretanjem ažuriranja se zapravo pokrenula instalacija samog crva. Crv je na zaraženom računalu u vremenskim intervalima uspostavljao konekciju s glavnim serverom te tako slao osjetljive podatke ili primao naredbe što napraviti sljedeće. Predviđa se da je grupa iza tog crva zaradila preko 2 milijuna eura kroz infekciju računala i krađu privatnih podataka. U Siječnju 2012. godine istraživač Jan Droemer zajedno sa sveučilištem u Alabami objavljuje rezultate istraživanja tog crva te razotkriva pojedince iza svega toga [18] [14].

Scenarij vježbe za potrebe prototipa je neovisan pozadinskoj priči o grupi *Koobface* no koristi ime te grupe i imena osoba povezanih s tom grupom u svrhu realnijeg opisa priče te samim time bolje edukacije analitičara.

5.2. Simulirani scenario: Novi napad grupe Koobface

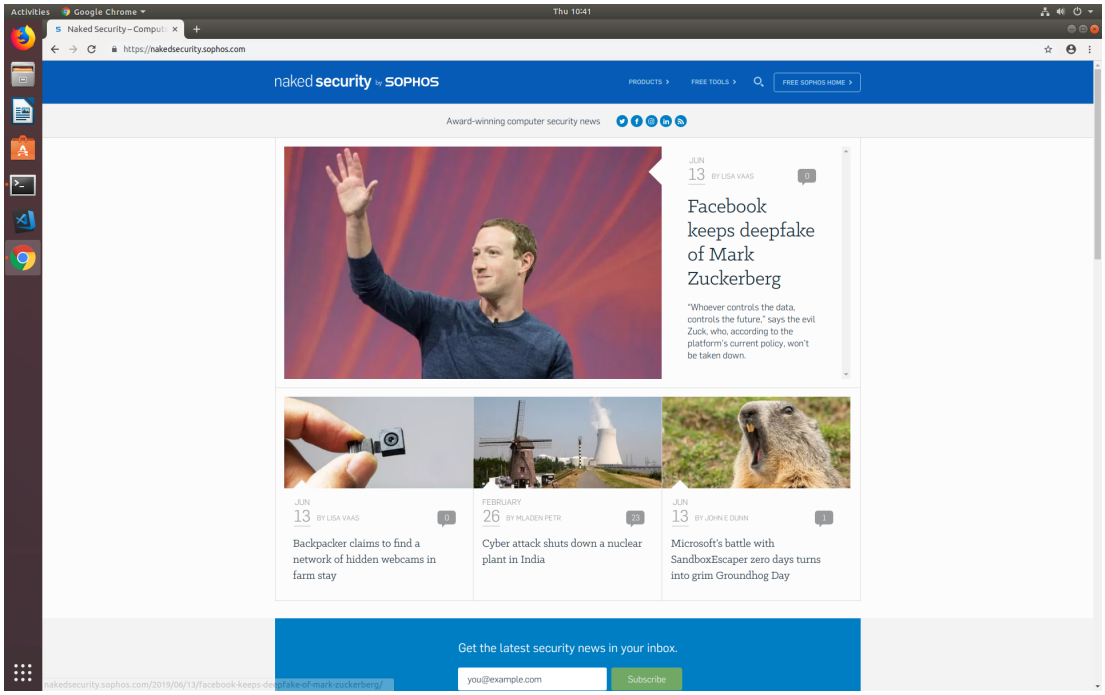
Tema simuliranog scenarija je bila kiber napad na nuklearno postrojenje u Indiji. Uz generirane vijesti, blogove i rezultate pretraživanja, analitičar ima zadatak da uz informacije dostupne od tih izvora napiše kratki izvještaj o grupi ili pojedincima koji su

izveli spomenuti napad.

Početak simuliranog scenarija može započeti na više načina. Jedan mogući način je imati početnu web stranicu, gdje preglednik uvijek usmjeri nakon paljenja programa, na kojoj je napisan zadatak koji analitičar mora obaviti. Drugi način je spremiti scenario u jednoj tekstualnoj datoteci u kojoj se opisuje problem te definira željeni ishod vježbe. Osim toga moguće je korisnika jednostavno uputiti na jedan od poslužitelja koji će ubaciti trag s kojim istraga može započeti. Način početka zadatka se planira jednoznačno odrediti u budućim inačicama ovog rada.

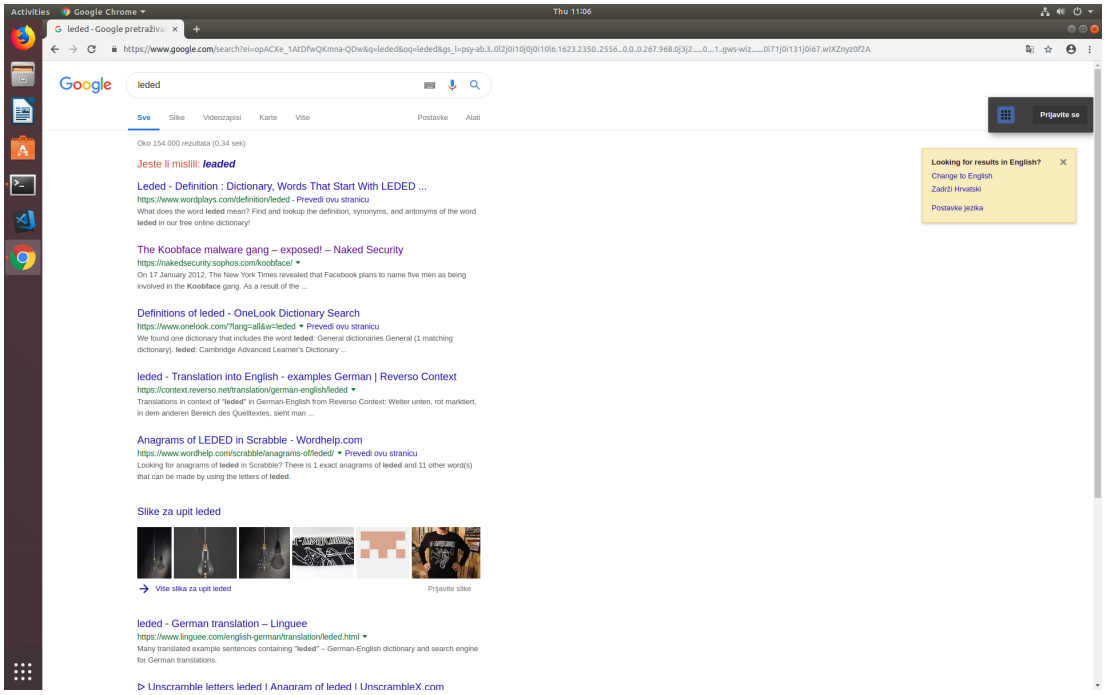
U primjeru testnog scenarija vježbe ako korisnik nakon paljenja programa posjeti stranicu poslužitelja *Sophos* otvorit će početnu stranicu na kojoj su prikazani najnoviji blogovi od raznih istraživača. Međutim jedan od blogova će biti zamijenjen s komponentom vježbe koja se aktivirala preko uvjeta zadanih kod stvaranja scenarija. Na slici 5.1 je ubačen blog o napadu na nuklearno postrojenje. Klikom na taj blog se otvara njegov sadržaj koji je druga komponenta scenarija. U trenutnom scenariju ta komponenta se aktivirala tek nakon što je bio ubačen sam članak odn. prva komponenta je bila roditelj drugoj. Prikaz sadržaja ubačenog bloga je na slici 5.2.

Iz sadržaja bloga se mogu izvući pseudonimi pronađeni u malicioznim datotekama koje su nađene od strane drugih istražitelja. Pretragom tih pseudonima na *Google* tražilici se pri vrhu pojavljuje stvarni blog koji opisuje maliciozne radnje grupe *Koobface*. Na slici 5.3 je prikazan primjer pretraživanja za pseudonim "*LeDeD*".



Slika 5.1: Primjer ubačenog sadržaja na poslužitelj *Sophos*

Slika 5.2: Primjer ubačenog blog sadržaja na poslužitelj *Sophos*



Slika 5.3: Primjer ubačenog rezultata pretraživanja u *Google* servis

6. Zaključak

U ovom radu je prikazano jedno rješenje kojim se može poboljšati način edukacije analitičara kibernetičkih prijetnji. Radi delikatne i kompleksne prirode posla bitno je da ljudi koji ulaze u ovo područje odmah dobiju što realniji uvid u stvarne probleme s kojima će se susretati. Trenutne vrste edukacije analitičara koje se nude na internetu se temelje na video materijalima s usputnim kratkim vježbama i simuliranim sustavima na kojima se prikazuju simulirani scenariji. Zbog količine podataka koja je svakim danom sve veća održavanje takvih sustava tako da relativno realno simuliranju stvarne situacije je vrlo teško.

Najveća prednost implementacije ovog rada je upravo rješenje problema popratnog "nebitnog" sadržaja kod simulacije scenarija zato što taj sadržaj čini sustav realnim. S relativno jednostavnom implementacijom budućih zamjena sadržaja te mogućnosti modularnog razvijanja budućih poslužitelja ovaj rad ima potencijal za brzo širenje među trenutnim i budućin analitičarima kibernetičkih prijetnji.

S druge strane dvije najveće mane su količina utrošenog vremena na stvaranje dovoljno realnih i logički povezanih vježbi te činjenica da prototip nije u kontroli sadržaja kojim rukuje. Unatoč tome s timom od nekoliko ljudi je moguće efikasno vremenski rasporediti zadatke daljnjeg razvijanja i održavanja sustava te kreativnog stvaranja vježbi koje će sigurno biti od koristi analitičarima koji će ju koristiti.

LITERATURA

[1] What is cyber threat intelligence, and why do you need it? URL www.businessnewsdaily.com/11141-cyber-threat-intelligence.html.

[2] Wikipedia, the free encyclopedia. URL www.circleid.com/posts/20190530_a_quick_look_at_the_4_most_prevalent_types_of_threat_intelligence.

[3] Mitmproxy certificates, . URL docs.mitmproxy.org/stable/concepts-certificates/.

[4] Mitmproxy addons, . URL docs.mitmproxy.org/stable/addons-events/.

[5] Mitmproxy documentation, . URL docs.mitmproxy.org/stable/tools-mitmweb/.

[6] Manual:pywikibot/mitmproxy, . URL www.mediawiki.org/wiki/Manual:Pywikibot/mitmproxy.

[7] Pypi mitmproxy project, . URL pypi.org/project/mitmproxy/.

[8] Sans course: Cyber threat intelligence. URL www.sans.org/course/cyber-threat-intelligence.

[9] Nnt log tracker enterprise. URL www.newnettechnologies.com/event-log-management.html.

[10] URL en.wikipedia.org/wiki/Human_intelligence_(intelligence_gathering).

[11] Violet Blue. New threat intelligence report skewers industry confusion, charlatans, 2015. URL www.zdnet.com/article/new-threat-

intelligence-report-skewers-industry-confusion-charlatans/.

[12] Jeff Compton. The cti process lifecycle: Achieving better results through execution, 2017. URL www.fireeye.com/blog/products-and-services/2017/10/cti-process-lifecycle.

[13] Recorded Future. Threat intelligence blog. URL www.recordedfuture.com/threat-intelligence/.

[14] Dirk Kollberg Jan Drömer. The koobface malware gang – exposed! URL nakedsecurity.sophos.com/koobface/.

[15] Tyson Macaulay. System and method for generating and refining cyber threat intelligence data, Kolovoz 25 2015. US Patent 9,118,702.

[16] Scott Pelley. Fbi director on threat of isis cybercrime, 2014. URL www.cbsnews.com/news/fbi-director-james-comey-on-threat-of-isis-cybercrime/.

[17] Patricia Pickett. Cyber intelligence analyst career overview, 2019. URL www.thebalancecareers.com/cyber-intelligence-analyst-2071296.

[18] Nart Villeneuve. Koobface: Inside a crimeware network, nov 2010. URL web.archive.org/web/20120914015420/www.infowar-monitor.net/reports/iwm-koobface.pdf.

[19] Douglas L. Wheeler. A guide to the history of intelligence, 2012. URL www.afio.com/publications/Wheeler_Hist_of_Intel_1800-1918_in_AFIO_INTEL_WinterSprg2012.pdf.

Dodatak A
Upute za korištenje prototipa

Implementirano rješenje se sastoji od ukupno nekoliko datoteka koje funkcioniraju kao dodatak za alat *mitmproxy*. Instalacija rješenja se sastoji od ispravnog postavljanja navedenog alata u način za razvijanje novih programskih dodataka te postavljanja korištenih vanjskih biblioteka programskog jezika.

A.1. Postavljanje okruženja

Rješenje je napisano u programskom jeziku *Python* u inačici 3.6. Instalacija programskog jezika je različita ovisno o operacijskom sustavu za koji se postavlja. Za *debian* sustave naredba za dohvat i instalaciju *Python* inačice 3.6 je:

```
>> sudo apt-get install python3.6
```

Za dohvat raznih paketa se koristi upravljač paketima *Pip* te ga je potrebno nadograditi na posljednju verziju. Njegova instalacija i nadogradnja za *debian* sustave se vrše preko sljedećih naredbi:

```
>> sudo apt-get install python3-pip
>> sudo pip3 install -U pip
```

Trenutna verzija rješenja koristi vanjsku biblioteku za parsiranje *HTML* odgovora *BeautifulSoup* koju je potrebno instalirati preko *Pip* alata ako nije instalirana unutar samog paketa programskog jezika.

Idući korak je preuzimanje izvornog koda alata *mitmproxy* sa sustava *Git* naredbom *clone* kao što je prikazano u nastavku:

```
>> git clone https://github.com/mitmproxy/mitmproxy.git
```

Automatizirana instalacija preuzetog alata se može izvršiti pokretanjem skripte *dev.sh* koja se nalazi unutar preuzetog paketa. Nakon što skripta završi svoje izvođenje, pokretanje alata se može napraviti preko definiranog virtualnog okruženja koje se poziva pokretanjem nove skripte na relativnoj putanji *venv/bin/activate* u odnosu na korijenski direktorij.

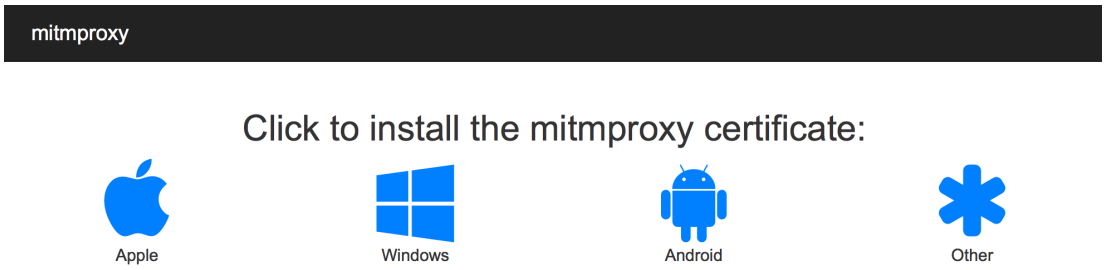
Unutar virtualnog okruženja se može pozvati bilo koja od izvršnih datoteka obrađenih u poglavlju 3.1. Nakon pokretanja alata sav dolazni i odlazni mrežni promet prolazi kroz alat koji pruža mogućnost modificiranja dobivenih zahtjeva i odgovora. U završnom koraku, da bi se omogućilo uspješno presretanje prometa, potrebno je ispravno konfigurirati certifikate alata što je opisano u nastavku.

A.2. Konfiguracija certifikata

Alat *mitmproxy* može modificirati mrežni promet ako mu računalo, na kojem je alat pokrenut, vjeruje. Ručnom instalacijom certifikata se daje dozvola alatu da po potrebi izmjenjuje odlazni i dolazni promet.

Prije postavljana certifikata je potrebno u pregledniku postaviti ručni *proxy* poslužitelj na adresu 127.0.0.1 na vratima 8080. Ovime će alat *mitmproxy* kod pokretanja preuzeti ulogu lokalnog *proxy* poslužitelja kojim će izvršavati vlastite funkcionalnosti.

Instalacija certifikata se obavlja nakon što se obave prethodni koraci, pokrene *mitmproxy* alat te posjeti stranica *mitm.it* u pregledniku koja sadrži upute za automatiziranu instalaciju certifikata za pojedine operacijske sustave. Ako je certifikat već postavljen tada se prikazuje prikladna poruka koja to naznačuje. Izgled stranice u situaciji kada certifikat nije instaliran je prikazan na slici A.1.



Slika A.1: Izgled stranice *mitm.it* ukoliko certifikat nije ispravno postavljen [3]

A.3. Stvaranje i uređivanje vježbi

Programsko rješenje se nalazi u direktoriju *addons* unutar preuzetog alata. Sve stvorene vježbe se prema konfiguraciji rješenja stavljaju u poddirektorij *exercises*. Struktura i programsko rješenje vježbi su opisani u poglavljima 3.2.1 i 4.1. Svaka nova vježba se stvara u obliku *JSON* datoteke sa strukturom navedenom u poglavlju 4.1 u kojem je naveden vremenski interval aktivnosti te definirane komponente koje će se ubacivati. Alat *mitmproxy* je potrebno ponovno pokrenuti ako je pokrenut u trenutku kada se doda nova vježba.

Prototip sustava za uvježbavanje analitičara za kibernetičke prijetnje

Sažetak

U radu je opisano što su kibernetička sigurnost i kiber inteligencija, koje vrste i razine kiber inteligencije postoje te zašto je važna pravilna edukacija iz tog područja. Opisana je pozicija analitičara kibernetičkih prijetnji. Definirana je arhitektura predloženog sustava za edukaciju analitičara kibernetičkih prijetnji i opisano programsko rješenje tog sustava. Provedeno je testiranje sustava nad jednostavnim scenarijem.

Ključne riječi: kibernetička sigurnost, analitičar kibernetičkih prijetnji, edukacija

Prototype of a traininig system for cyber threat analysts

Abstract

This work describes what cyber security and cyber threat intelligence are, names and describes levels and types of cyber threat intelligence and analyses why education in this field is important. It covers the role of a cyber threat analyst. It proposes an education platform for cyber threat analyst, defines its architecture and describes its implementation. At the end it tests the implementation using a simple education scenario.

Keywords: cyber security, cyber threat intelligence analyst, education