SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 769

Programska podrška za planiranje i provođenje kibernetičkih napada na zadanom informacijskom sustavu

Darija Filipović

Zagreb, rujan 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 769

Pristupnica:	**Darija Filipović (0036523875)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Programska podrška za planiranje i provođenje kibernetičkih napada na zadanom informacijskom sustavu**

Opis zadatka:

Tijekom planiranja napada za potrebe provođenja kibernetičkih vježbi potrebno je imati definirano okruženje u kojemu se napad provodi, a također je potrebno voditi računa o raspoloživim resursima. Pri tome je izazov što tijekom planiranja napada potencijalno nisu poznati svi detalji okruženja već kako se napad razvija, tako napadač saznaje sve više o okolini u kojoj se nalazi i mijenja svoje ponašanje. Dodatno, moguće je da se planirane aktivnosti ne mogu provesti iz raznih razloga. U sklopu ovog završnog rada potrebno je ispitati mogućnost integracije alata za planiranje napada s podacima o informacijskom sustavu koji se napada. Nadalje, treba razraditi načine kako prikazati i upravljati alternativnim pravcima napada na informacijski sustav te kako upravljati planom napada u ovisnosti o novim informacijama koje se saznaju o informacijskom sustavu tijekom pada.

Rok za predaju rada: 10. lipnja 2022.

SADRŽAJ

1. Uvod

Penetracijsko je ispitivanje simuliranje napada na ranjivi sustav u svrhu definiranja i iskorištavanja ranjivosti. Pronađene ranjivosti tada je moguće ispraviti i sustav zaštititi od budućih pravih napada. Ovaj je proces, zbog opsega posla koji ispitivači moraju obaviti, između ostalog, vremenski zahtjevan i da bi se efikasno izveo iziskuje intuiciju koja se stječe provođenjem samih testiranja, ali i pomoćnim metodama kao što su kibernetičke vježbe. Kibernetičke vježbe simuliraju stvarne situacije bez stvarnog rizika i služe učenju penetracijskih ispitivača, ali i podizanju svijesti zaposlenika u tvrtkama o rizicima, ili simuliranju određene situacije IT odjelu tvrtke da bi se znali sami nostiti s istom kada se dogodi u stvarnosti. Kao podrška planiranju i penetracijskih testova i kibernetičkih vježbi, razvijeni su brojni alati koji određene dijelove provedbe sistematiziraju, olakšavaju i čak automatiziraju. Ideja rada je doprinijeti razvoju jednog takvog alata koji će omogućiti organizaciju vremena odnosno plan samog napada ili testa.

Rad se sastoji od pet poglavlja od kojih je prvo Uvod. U drugom je poglavlju opisan problem planiranja i provođenja penetracijskih testova, odnosno faze koje je potrebno proći prije izvedbe samog testiranja. Slijedi poglavlje s objašnjenjem kibernetičke vježbe, koja je svrha provedbe i koji su izazovi s kojima se susrećemo. Nakon toga slijedi poglavlje u kojem je opisana programska podrška razvijena u sklopu ovog rada, to jest pregled tehnologija korištenih za izradu rješenja, pregled funkcionalnosti i opisa njihove implementacije te primjer korištenja. Posljednje je poglavlje zaključak prethodno napisanoga.

2. Koraci penetracijskog testa

Penetracijski testovi sastoje se od širokog skupa aktivnosti koje zahtijevaju široko znanje i visoku razinu sistematičnosti. Zbog toga su neovisno osmišljene različite metode penetracijskog testiranja koje posao dijele u više faza. Jedan takav smjer postavili su autori Penetration Testing Execution Manuala (nadalje PTES), ali sličnu podjelu radi i Open Source Security Testing Methodology Manual (OSSTMM) Herzog (2010), Open Web Application Security Project (OWASP) OWA (2002) i Information Systems Security Assesment Framework (ISSAF) Roback (2000). Svaki priručnik tj. metodologija bira se po specifičnim kriterijima i pristupu svakog ispitivača i načinu usmjerenja koji mu je potreban (usmjerava li se posebna metodologija više na završni izvještaj ili na sam proces testiranja). PTES proces testiranja dijeli na 7 faza Nickerson et al. (2011):

- Interacije prije angažmana (eng. Pre-engagement Interactions)
- Prikupljanje podataka i znanja (eng. Intelligence Gathering)
- Modeliranje prijetnji (eng. Threat modeling)
- Analiza ranjivosti (eng. Vulnerability Analysis)
- Iskorištavanje ranjivosti/Faza napada (eng.Exploitation)
- Faza poslije napada (eng. Post Exploitation)
- Pisanje izvještaja (eng. reporting)

2.1. Interacije prije angažmana

Ovu fazu obilježavaju početni dogovori u više navrata. Dogovaraju se opseg posla, granice koje se ne smiju preći, tehnička izvedba, kojim informacijama ispitivač ima pristup, ciljevi odnosno zahtjevi klijenta i potrebni dogovori s trećim strankama koje sudjeluju i trebaju biti obavještene o provođenju testiranja.

2.2. Prikupljanje podataka i znanja

Nakon dogovorenog opsega posla, u ovoj se fazi skupljaju informacije o klijentu tj. ciljanoj žrtvi da bi se pronašle točke upada i ranjivosti koje će se iskoristiti u kasnijim fazama. Tijekom prikupljanja podataka bitno je voditi računa da se podaci steknu samo legalnim i s klijentom dogovorenim putem. Nakon što se steknu nove informacije, potrebno je revidirati uvjete i pravila angažmana koji su dogovoreni na prethodnim sastancima. Podaci koje se cilja pronaći su svi oni koji bi mogli omogućiti i olakšati neki smjer napada, a uključuju: informacije o fizičkim lokacijama, najmu, partnerima, klijentima, infrastrukturi, otvorenim pristupima i servisima pokrenutima na mreži, financijama, konkurenciji, sastancima, otvorenim natječajima, sudskim procesima, donacijama, adresama e-pošte, meta-podaci iz pronađenih dokumenata.

2.3. Modeliranje prijetnji

Prilikom ovog koraka definira se imovina i napadačka strana kako bi ispitivač odlučio prioritete napada i mogućnost postizanja tih ciljeva. Imovina se dijeli na fizičku imovinu tvrtke koja predstavlja cilj napada, kao baze podataka, izvoni kod, podaci o proizvodima, podaci za prijavu kao administrator i slično, i na procese poslovanja koji predstavljaju način napada kao što su tehnička infrastruktura, integracije s aplikacijama trećih strana, planovi za budućnost. Napadačka se strana dijeli na zajednice koje predstavljaju prijetnju, kao zaposlenici, uprava, otpušteni, vanjski i sposobnosti istih, što uključuje dostupnost podataka i procesa, sposobnost koirštenja potrebnih alata. Mapiranjem prijetnji u odnosu na imovinu slaže se model.

model? Išta o njemu?

2.4. Analiza ranjivosti

Analiza ranjivosti je korak za pronalazak ranjivosti na temelju prethodno definiranog modela prijetnji aktivnim i pasivnim testiranjem na kraju čega se provodi validacija. Aktivno testiranje provodi se direktnom interakcijom s komponentama koje se ispituju, skeniraju se otvoreni pristupi (eng. *port*), servisi i protokoli i testira se na generalne ranjivosti, a može se provoditi ručno i automatizirano, što ispitivaču ostavlja vrijeme da se orijentira na obradu podataka i zadatke koje je nužno obaviti ručno. Pasivno testiranje obuhvaća analizu metapodataka i praćenje prometa da bi se pronašli slučajno otkriveni podaci. Na kraju testiranja potrebno je provesti validaciju pronađenih ranjivosti provjeravanjem jesu li različiti alati za iste pretrage dali iste rezultate. Pronađene

ranjivosti potrebno je validirati tj. potvrditi kao ranjivosti i istražiti moguće načine iskorištavanja ispitivanjem i pretragom u bazama ranjivosti kao što su Bugtraq, Exploit Database, CVE i OSVDB. Putem se slaže stablo napada s mogućim smjerovima i načinima iskorištavanja ranjivosti.

2.5. Iskorištavanje ranjivosti/Faza napada

Tijekom napada iskorištavaju se pronađene ranjivosti zaobilaženjem sigurnosnih restrikcija. Pomoću materijala prikupljenih u prethodnim fazama i konstruiranog stabla napada odlučuje se koji je najučinkovitiji smjer napada te kako zaobići prepreke koje je klijent postavio u svrhu vlastite zaštite.

2.6. Faza poslije napada

Nakon napada se određuje vrijednost osvojenih uređaja i razina kontrole nad sustavom koju je napadač uspostavio, što se određuje na temelju osjetljivosti pronađenih podataka i pravila i prioriteta dogovorenih s klijentom u početnim fazama ispitivanja. Osim toga je potrebno sve izmijenjeno vratiti u prvobitni položaj tako de se klijenta ne ometa u daljnjem radu. Kako bi se to postiglo, potrebno je tijekom napada održavati popis poduzetih radnji i isti priložiti završnom izvješću. Budući da su se tijekom naoada saznale nove informacije koje nije bilo moguće prethodno istražiti, revidiraju se ranjivosti ponovnom analizom infrastrukture i predlažu novi napadi.

2.7. Pisanje izvještaja

U ovom se koraku priprema završno izvješće za klijenta u kojem se razlažu sve pronađene ranjivosti, poduzete mjere i smjernice za poboljšanje sigurnosti sustava.

3. Kibernetičke vježbe

Kibernetičke vježbe služe da bi se osigurala pripravnost i sposobnost branitelja za reakciju na događaje koji čine kibernetičku kriznu situaciju. Definicija plana kibernetičke vježbe korištena u ovom radu sastoji se od početnog i ciljnog stanja te međukoraka nužnih za dolazak u ciljno stanje, počevši od početnog stanja. Korak, odnosno stanje plana, sastoji se od akcije koja se tim korakom izvršava, preduvjeta za izvođenje te akcije, posljedica koje se ostvaruju izvođenjem tog koraka i trajanja izvođenja akcije. Svaki korak plana pretpostavlja određeno globalno stanje resursa, a prilikom provedbe mijenja stanje resursa. Nužan uvjet za ispravnost plana jest dosljedno upravljanje resursima, odnosno odabir takvog slijeda akcija u kojem za svaku akciju vrijedi da su zadovoljeni njezini preduvjeti i da njezine posljedice nisu kontradiktorne s preduvjetima sljedeće akcije. Nužan uvjet za izvedivost plana jest da je korake moguće izvoditi sekvencijalno, to jest da je svaki idući korak planiran nakon promatranog koraka u vremenskoj domeni. Planiranje kibernetičkih vježbi uključuje odabir ciljeva napada te definiranje akcija pomoću kojih se odabrani ciljevi mogu ostvariti, a potom i raspoređivanje definiranih akcija u vremenskoj domeni na način koji osigurava da su zadovoljeni nužni uvjeti za ispravnost i izvedivost plana. U kontekstu provođenja napada, koraci su dodatno prošireni trenutnim napretkom izvođenja koraka u vremenskoj domeni i indikatorom uspješnosti izvođenja akcije. Ručna izrada plana kibernetičke vježbe složen je i dugotrajan proces zbog potrebe za definiranjem alternativnog plana u slučaju potencijalnog neuspjeha pri izvođenju koraka plana, što rezultira stablom u kojem svako grananje stvara onoliko konkurentnih planova koliko grana postoji. Programska podrška za planiranje i provođenje kibernetičkih vježbi opisana u ovom radu za izračun plana koristi algoritam parcijalnog poretka, a za prikaz plana kibernetičke vježbe i praćenje provedbe koristi gantogram, graf pogodan za prikaz trajanja različitih aktivnosti u vremenu. Ulazni parametri nužni za planiranje kibernetičke vježbe pomoću navedene programske podrške su početno i ciljno stanje, definirani kao čvorovi u gantogramu, te akcije koje imaju definirane preduvjete i/ili posljedice i trajanje izvođenja akcije. Nakon definiranja ulaznih parametara korisnik okida izračun plana pritiskom na gumb.

Ukoliko postoji plan koji zadovoljava sve ciljeve i izvediv je u danom vremenskom intervalu, utoliko se takav plan prikazuje korisniku kao gantogram, a u suprotnom se ispisuje poruka o pogrešci. Korisnik potom može pratiti napredak izvođenja koraka u vremenu ili označiti neki korak kao neuspješan te na taj način okinuti ponovni izračun plana od tog koraka nadalje.

4. Opis programske podrške

Razvijena programska podrška opisana u ovom radu implementirana je kao web aplikacija zasnovana na arhitekturi klijent-poslužitelj. Klijent je izveden korištenjem razvojnog okvira React JavaScript. Sav pripadni izvorni kod za klijent nalazi u direktoriju `client`. Za izvedbu gantograma na klijentu korištena je DHTMLX Gantt biblioteka. Poslužitelj je implementiran korištenjem razvojnog okvira Flask. Za spremanje podataka korištena je baza podataka MongoDB. Izvorni kod poslužitelja nalazi se u direktoriju `server`. Klijent i poslužitelj komuniciraju preko aplikacijskog programskog sučelja (eng. *Application Programming Interface*, skraćeno API) za prijenos reprezentacijskog stanje (eng. *Representational State Transfer*, skraćeno REST). Kako bi se osigurali ispravan rad i isporuka (eng. *deploy*) aplikacije, korišten je alat Docker; klijent i poslužitelj imaju pripadne Docker datoteke, a za upravljanje s više Docker kontejnera (eng. *container*) korišten je alat Docker Compose.

4.1. MongoDB

MongoDB je nerelacijska baza podataka koja podatke sprema u različitim kolekcijama u BSON dataotekama, binarnim reprezentacijama JSON datoteka mon (2022). Umjesto tablica je svaki upis u kolekciji strukturiran na način da ključ poredstavlja "stupac" u kojem bi se unos nalazio, a vrijednost predstavlja samu vrijednost unosa, tj. upisana su samo pojedina polja koja su određena BSON datotekom. U bazi su stvorene kolekcije:

- `tasks`- u koju se pohranjuju zadaci koji čine korake plana i prikazuju se plavim pravokutnicima u sučelju,
- `links`- u koju se pohranjuju veze između zadataka koje se prikazuju narančastim strelicama u sučelju,
- `effects`- u koju se pohranjuju resursi stečeni obavljanjem zadataka koji se prikazuju u pregledu resursa na desnoj strani sučelja,

- `actions`, u koju se pohranjuju moguće akcije predajom JSON datoteke na pritisak gumba "Import Actions",
- `partial_plans` u koju se pohranjuju izračunati parcijalni planovi potrebni za izračun cjelokupnog plana,
- `counters`- koji služi spremanju posljednjeg i generiranju novog identifikacijskog broja za zadatke, poveznice, efekte, akcije i parcijalne planove i
- `hits` - u koju se pohranjuje broj otvaranja stranice za provjeru pravilnog spajanja servera i baze.

4.1.1. Struktura dokumenata

Zadaci su pojedini koraci plana, pohranjeni su u kolekciji `db.tasks` i sastoje se od sljedećih atributa:

- `task_id` - identifikator zadatka,
- `text` - tekst zadatka koji se prikazuje na gantogramu,
- `start_date` - datum i vrijeme početka izvođenja zadatka,
- `end_date` - datum i vrijeme kraja izvođenja zadatka,
- `action` - identifikator akcije koja se izvodi,
- `parent` - identifikator roditelja zadatka ako je zadatak ugniježđen,
- `failed` - zastavica koja označava je li zadatak proveden,
- `preconditions` - preduvjeti nužni za izvođenje zadatka,
- `effects` - rezultati tj. posljedice obavljanja zadatka,
- `color` - boja zadatka i
- `fail_handled` - zastavica koja označava je li zadatak obrađen ako nije proveden tj. ako je nakon izračuna određeno da je zadatak nepotreban, je li mu boja promijenjena i jesu li mu rezultati uklonjeni.

Poveznice označavaju slijed i odnose među zadacima, pohranjene su u kolekciji `db.links` i sastoje se od atributa:

- `link_id` - identifikator poveznice,
- `source` - identifikator izvorišnog zadatka iz kojeg poveznica potječe,
- `destination` - identifikator odredišnog zadatka na koji poveznica pokazuje,

- `type` - tip poveznice, uobičajena vrijednost je 0 što označava da poveznica ima izvor u završetku jednog zadatka, odredište u početku drugog zadatka i odredišni zadatak ne može započeti prije nego izvorišni završi. Ostali mogući tipovi su:
    - "1" koji izvire u početku jednog zadatka, ponire u početak drugog zadatka i označava da odredišni zadatak ne može započeti prije nego počne izvorišni,
    - "2" koji izvire u kraju jednog zadatka, ponire u kraj drugog zadatka i označava da odredišni zadatka ne može završiti prije nego izvorišni završi i
    - "3" koji izvire u početku jednog zadatka, ponire u kraj drugog zadatka i označava da odredišni zadatak ne može završiti prije nego izvorišni započne.

Akcije predstavljaju moguće odabire pri planiranju i stvaranju zadataka, pohranjene su u kolekciji `db.actions` i sastoje se od atributa:

- `action_id` - identifikator akcije,
- `name` - naziv akcije,
- `preconditions` - preduvjeti potrebni za izvršenje akcije,
- `posteffects` - rezultati tj. posljedice izvršenja akcije,
- `time` - vrijeme potrebno da se akcija izvrši i
- `price` - cijena izvršenja akcije koja u ovom slučaju nije iskorištena, ali pruža prostor za unapređenje programske podrške.

Efekti su posljedice izvršenja zadataka i akcija, pohranjeni su u kolekciji `db.effects` radi jednostavnijeg dohvaćanja, tj. izbjegavanja ponovnih poziva u bazu da bi se pronašli svi rezultati za pregled resursa, i sastoje se od atributa:

- `effect_id` - identifikator posljedice,
- `effect` - naziv posljedice,
- `task_id` - identifikator roditeljskog zadatka i
- `date_acquired` - datum i vrijeme kada je efekt postignut.

Parcijalni poredci stvaraju se pokretanjem izračuna potpunog plana kao rezultat metode `partial_order_planner`, pohranjeni su u kolekciji `db.partial_plans`, a sastoje se od atributa:

- `steps` koji je lista objekata koraka to jest zadataka,
- `ordering_constraints` - liste ograničenja pri rasporedu,
- `causal_links` - liste poveznica i
- `successful` - boolean vrijednosti uspijeha.

Svaki je korak zapravo pojednostavljena akcija - sastoji se od identifikatora, liste uvjeta, liste rezultata i vremena potrebnog za izvršenje. Stanje u bazi unutar docker kontejnera može se pregledati pokretanjem interaktivne sesije kontejnera naredbom

```
docker exec -it <ime_kontejnera_baze> bash
```

pri čemu je ime kontejnera baze u ovom slučaju "mongo". Izvršenjem naredbe u terminalu dobiva se korijenski pristup kontejneru "mongo" gdje se izvođenjem naredbe `mongo` otvara sučelje naredbenog retka za bazu čime se ona može ispitivati.

4.2. DHTMLX Gantt

DHTMLX Gantt je biblioteka korištena za prikaz gantograma DHTMLX (2022). Inicijalizacijom elementa `gantt` stvara se polje za prikaz gantograma te polje za prikaz imena pojedinih zadataka s pripadajućim gumbom za dodavanje zadataka. Konfiguracija elementa se postavlja mijenjanjem atributa `gantt.config`. Funkcije koje se pozivaju okidanjem određenog događaja dodaju se pomoću `gantt.attachEvent`. `gantt.locale.labels` mijenja nazive dijelova za unos zadatka koji su definirani putem `gantt.config.lightbox.sections`. Nakon postavljanja konfiguracije, `gantt.init` inicijalizira korištenje biblioteke unutar HTML kontejnera, a `gantt.load` učitava putanju do API metoda na poslužitelju potrebnih za dohvaćanje, dodavanje, izmjenu i brisanje zadataka i poveznica na gantogramu. `gantt.createDataProces` također učitava putanju do API metoda na poslužitelju da bi se stvorio procesor podataka koji se veže na sami element `gantt` i šalje podatke na odgovarajuću rutu.

4.3. Flask

Poslužitelj je izveden koristeći programski jezik Python i razvojni okvir Flask Pallets (2010). Izvorni kod poslužitelja nalazi se u datoteci `server.py`, a podijeljen je

na API metode za komunikaciju s klijentom i pomoćne metode za izvršavanje logike programskog rješenja.

4.3.1. API metode

Za pravilnu integraciju biblioteke DHTMLX Gantt s poslužiteljem definirane su `GET` (`get_tasks`, `get_links`), `POST` (`create_task`, `add_link`), `PUT` (`update_task`, `update_link`) i `DELETE` (`delete_task`, `delete_- link`) metode za dodavanje, dohvaćanje, izmjenu i brisanje pojedinačnih zadataka i poveznica između zadataka. Metoda `create_task` uz pohranjivanje novostvorenog zadatka u bazu sprema posljedicu izvršenja zadatka. Osim njih su definirane dodatne metode za izvođenje programske podrške.

`GET` metoda `get_effects` dohvaća spremljene posljedice zadataka iz baze.

`POST` metoda `import_actions` osigurava uvoz akcija predanih u obliku JSON datoteke u bazu.

`GET` metoda `get_actions` dohvaća spremljene akcije iz baze.

`POST` metoda `import_existing_project` uvozi već pripremljen raspored zadataka i poveznica za prikaz na gantogramu u bazu.

`GET` metoda `get_plan` izvršava planiranje scenarija kibernetičke vježbe.

Metoda `clear_gantt_chart` uklanja iz baze zadatke i poveznice prikazane na gantogramu.

4.3.2. Pomoćne metode

Pomoćne metode rasterećuju opseg posla API metoda i čine izvorni kod preglednijim.

Metoda `get_next_sequence` povećava vrijednost brojača za jedan i vraća traženi brojač, a služi generiranju i praćenju identifikatora zadataka, poveznica, akcija, efekata i parcijalnih planova.

Metoda `clear_chart` iz baze briše sve zadatke, poveznice, efekte i parcijalne planove te poništava identifikacijske brojeve u kolekciji `counters`, a zovu je API metode `import_existing_project` prethodno prikazu uvezenih zadataka i `clear_chart`.

Metoda `clean_previous_plan_if_exists`, jednaka je metodi `clear_chart`, osim što zadržava zadatke s identifikacijskim brojem manjim od tri, to jest početni i ciljni zadatak, te vraća brojač identifikatora zadataka na vrijednost 2. Metoda se koristi u API metodi `get_plan` prethodno metodi `plan_gantt_actions` da bi se prikaz vratio na početni problem za koji korisnik želi izračun plana.

Metoda `plan_gantt_actions` za argument uzima identifikator zadatka u gantogramu koji služi kao početna točka za računanje plana i priprema potrebne liste mogućih akcija, poduzetih koraka i ciljanih preduvjeta te poziva metodu `partial_order_planner`. Ako rezultat metode `partial_order_planner` kao vrijednost atributa `successful` ima `True`, poziva se metoda `construct_gantt_total_order_plan`.

Metoda `partial_order_planner` prima tri argumenta: objekt problema koji treba isplanirati, listu ciljeva i listu akcija te rekurzijom provodi algoritam planiranja parcijalnog poretka., a koristi pomoćne metode `where_contains_effect`, `check_if- _threat`, `conditions_intersection` da bi konačno vratila izmijenjen prvi argument.

Metoda `construct_gantt_total_order_plan` prima dva argumenta: objekt parcijalnog plana i identifikator početne akcije od koje se računa, odabire jedan potpuni poredak, raspoređuje korake u vremenu i konstruira zadatke za gantogram, vraća istinu ili laž, a za računanje koristi pomoćne metode `construct- _total_order` i `construct_final_order`.

Metoda `where_contains_effect` provjerava sadrži li akcija posljedicu jednakog naziva kao argument i, ako da, je li joj vrijednost jednaka onoj u argumentu.

Metoda `check_if_threat` provjerava jednakost vrijednosti uvjeta i posljedice predanih u argumentu. Ako vrijednosti nisu jednake, akcija s uvjetom se ne može nadovezati na akciju s posljedicom pa se ona označava kao prijetnja i vraća se istinita vrijednost.

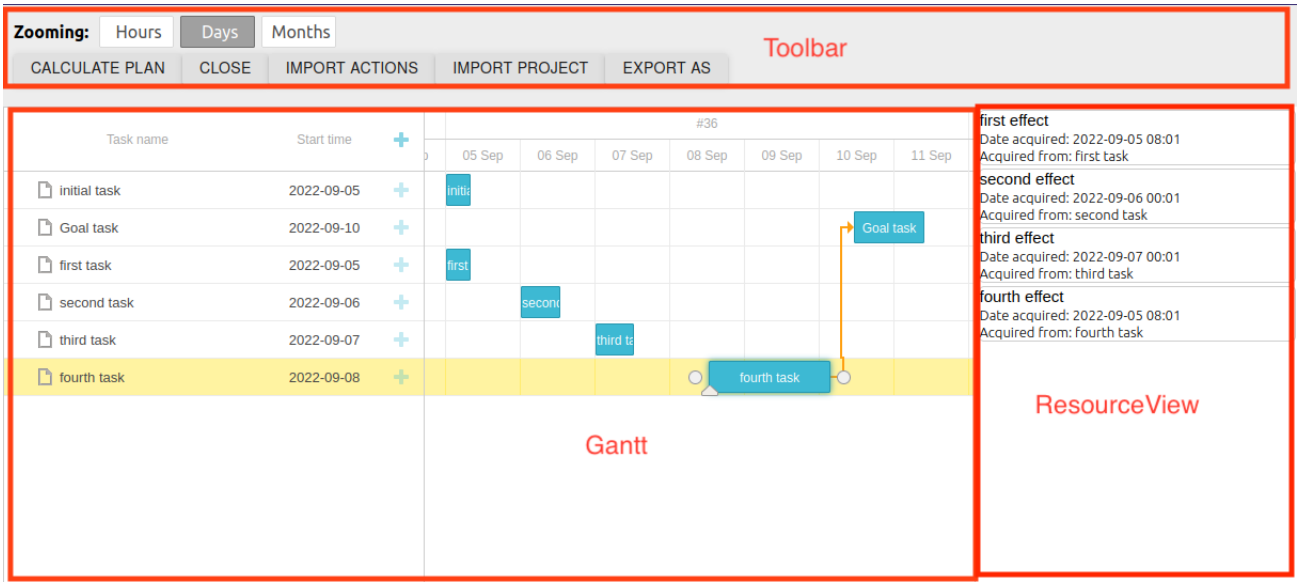
Metoda `conditions_intersection` provjerava jednakost dvaju uvjeta i vraća istinu ako su nazivi i vrijednosti uvjeta predanih u argumentima jednaki.

Metoda `construct_total_- order` prima objekt parcijalnog plana i bira jedan mogući potpuni poredak.

Metoda `construct_final_order` kao argument prima objekt potpunog poretka i preoblikuje ga u niz objekata pogodnijih za oblikovanje u metodi `construct_gantt_- total_order_plan` koja ju poziva.

4.4. React JavaScript

Komponente klijenta izvedene su pomoću web razvojnog okvira React JavaScript Inc. (2022c). U metodi `render` u vršnoj komponenti `App.js` inicijaliziraju se komponente `Toolbar`, `Gantt` i `ResourceView` koje su označene na fotografiji 4.1



Slika 4.1: Izgled sučelja aplikacije

Komponenta `Toolbar` tvori alatnu traku u gornjem dijelu ekrana i sadrži dva reda gumba. U prvom redu se nalaze tri gumba koja služe podešavanju vremenskog rasporeda u gantogramu. Moguće je odabrati između prikaza sati, dana i mjeseci. U drugom se redu nalaze gumbi :

– `CALCULATE PLAN` kojim se uzima trenutno stanje u gantogramu i poziva API metoda `get_plan` čiji je rad opisan u potpoglavlju 4.3 Flask,

– `CLOSE` kojim se poziva API metoda verb `clear_gantt_chart` i kojim se briše stanje u gantogramu,
– `IMPORT ACTIONS` koji poziva API metodu `import_actions` i kojim se uvoze nove akcije,
– `IMPORT PROJECT` koji poziva API metodu `import_existing_project` kojim se mijenja trenutni prikaz gantograma na onaj predan pritiskom na gumb i
– `EXPORT AS` koji otvara dijalog za odabir formata spremanja stanja, koji može biti JSON, Excel, PDF, JPG i iCal.

Komponenta `Gantt` sadrži prikaz zadataka kao popis i u gantogramu. Elementi potrebni za rad s gantogramom dostupni su u biblioteci DHTMLX Gantt čije se korištenje omogućava naredbom `import { gantt } from "dhtmlx-gantt"`. Element samog gantograma inicijalizira se u metodi životnog ciklusa `componentDidMount` naredbom `gantt.init(this.ganttContainer)`. Da bi se svi podaci pravilno slali na poslužitelj i spremali u bazu inicijaliziran je element `initGanttDataProcessor` koji se prosljeđuje elementu gantt naredbom `this.dataProcessor.init(gantt)`. Budući da se metoda `componentDidMount` poziva samo prvi put kad aplikacija poziva metodu `render`, inicijalizacija se obavlja samo jednom. Osim toga se postavlja konfiguracija gantograma.

Komponenta `ResourceView` sadrži prikupljene resurse tj. efekte zadataka.

Komponenta `Export` postaje vidljiva klikom na gumb `Export as` na komponenti `Toolbar` i sadrži gumbe kojima se pokreće izvoz gantograma u različitim foramtima i gumb za zatvaranje prozora.

4.5. Docker

Docker je alat koji olakšava prenosivost aplikacija kreiranjem slika koje se grade jednako bez ozbira na temeljnu konfiguraciju i operacijski sustav domaćinskog uređaja Inc. (2022a). Svaka Docker slika (eng. *image*) po pokretanju postaje kontejner. Programska podrška se sastoji od 3 Docker kontejnera. Klijent je izveden kontejnerom čija je konfiguracija prikazana izvodnim kodom 4.1.

Izvorni kod 4.1: Dockerfile za upravljanje kontejnerom za klijenta

```
1  FROM node:16.8-alpine
2
3  WORKDIR /client
4
5  ENV PATH /app/node_modules/.bin:$PATH
6
7  COPY package.json ./
8  COPY package-lock.json ./
9  RUN npm install
10
11 COPY ./public ./public
12 COPY ./src ./src
13
14 EXPOSE 3000
15
16 CMD npm start
```

Nareba FROM stvara sloj slike iz već postojeće Docker slike `node:16.8.-alpine`. Naredba `WORKDIR` postavlja `/client` kao radni direktorij u kojem se izvršavaju sve sljedeće naredbe. Naredba `ENV` postavlja varijablu okoline za putanju do mape `node_modules` da bi Docker znao koje ovisnosti mora razriješiti. Sljedeće dvije naredbe kopiraju sadržaje datoteka navedenih u prvom argumentu koje se nalaze lokalno na lokaciju navedenu u drugom argumentu na datotečnom sustavu Docker kontejnera. Naredba `RUN` pokreće naredbu navedenu u nastavku da bi se preuzeli svi potrebni paketi i razriješile ovisnosti. Naredba `EXPOSE` otvara pristup (eng. *port*) kojim se može pristupiti napravljenom kontejneru. Konačno se poziva naredba `CMD` koja pokreće klijent. U jednom Docker kontejneru mora biti točno jedna naredba `CMD`.

Izvorni kod Dockerfilea za izgradnju poslužitelja prilazan je izvornim kodom 4.2

Izvorni kod 4.2: Dockerfile za upravljanje kontejnerom za poslužitelja

```
1  FROM python:3.10.4-alpine
2
3  WORKDIR /server
4
5  COPY requirements.txt ./
6
7  RUN pip install --no-cache-dir -r requirements.txt
```

```
8
9  COPY server.py ./
10
11 EXPOSE 8080
12
13 CMD python server.py
```

Slično Dockerfileu klijenta, kontejner se gradi iz već postojeće slike `python:3.10.4.-alpine`. Naredbom `RUN` pokreće se instaliranje potrebnih paketa definiranih u datoteci `requirements.txt` putem alata pip, naredbom `EXPOSE` izlaže se se pristup 8080, a naredbom `CMD` pokreće poslužitelj.

Docker Compose je alat za definiranje i pokretanje Docker aplikacija s više kontejnera [Inc. (2022b)]. Konfiguracija se piše u datoteku `docker-compose.yml` koja je prikazana Izvornim kodom 4.3

Izvorni kod 4.3: docker-compose.yml

```
1  version: '1.0.0'
2  services:
3    mongo:
4      container_name: mongo
5      image: mongo:4.0-xenial
6      volumes:
7        - data:/data/db
8    flask:
9      container_name: flask
10     build: ./server
11     ports:
12       - "8080:8080"
13     depends_on:
14        - mongo
15
16   client:
17     container_name: client
18     build:
19       context: ./client
20       dockerfile: Dockerfile
21     volumes:
22       - '.:/app'
```

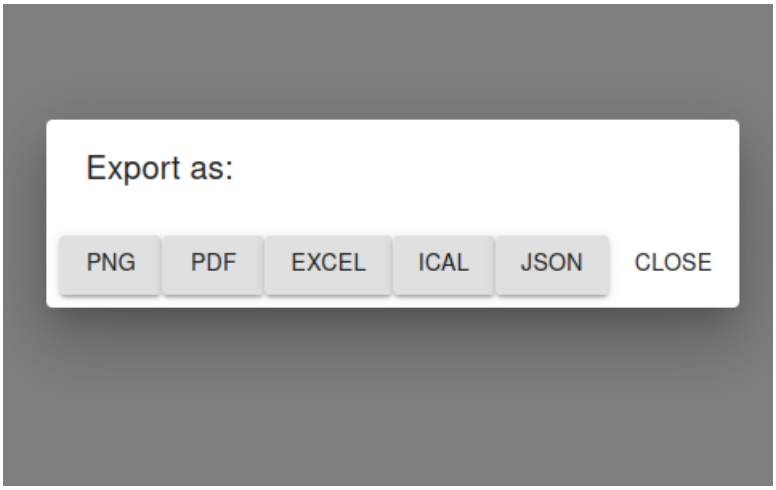
```
23        - '/app/node_modules'
24      ports:
25        - 3000:3000
26      environment:
27        - CHOKIDAR_USEPOLLING=true
28
29  volumes:
30    data:
```

Prva linija određuje verziju formata datoteke docker-compose.yml. Verzija određuje oblik i dopuštene naredbe u ostatku datoteke. U dijelu označenom `services` definirani su svi kontejneri koji se grade, a to su `mongo` koji služi kao baza podataka, `flask` na kojem se nalazi poslužitelj i `client` koji pokreće klijent. Za kontejner mongo se koristi već postojeća slika `mongo:4.0-xenial`, a lokacija pohrane podataka postavlja se naredbom `volumes`. Poslužitelj se gradi na temelju mape `server` u kojoj se nalazi datoteka `server.py` i pripadajući `Dockerfile` naredbom `build`. Naredbom `ports` mapira se lokalni pristup na računalu na pristup u kontejneru. Naredbom `depends_on` kontejneru se javlja ovisnost o drugoj komponenti, u ovom slučaju o prethodno definiranoj bazi. Klijent je definiran slično poslužitelju. Naredbom `volumes` definirano je gdje se na kontejneru sprema izvorni kod aplikacije i sadržaj paketa potrebnih za razrješavanje ovisnosti. Odjeljak `volumes` definira ime lokacije za pohranu sadržaja baze na kontejneru. Naredbom `docker compose build` grade se slike koje su određene u odjeljku `services`, odnosno u zasebnim Dockerfileovima. Nakon toga se može pozvati `docker compose up` kojom se gradi cjelokupna aplikacija.

4.6. Funkcionalnosti

4.6.1. Ispravna izgradnja Docker kontejnera

Alat je ispravno "dokeriziran" što znači da su Dockerfileovi i docker-compose.yml izmijenjeni tako da se kontejneri ispravno grade, s pravilno podešenim pristupima, ovisnostima i lokacijama spremanja podataka. Pregled istih je moguć uvidom u izvorne kodove 4.1, 4.2 i 4.3.



Slika 4.2: Dijalog za izvoz gantograma

4.6.2. Izvoz

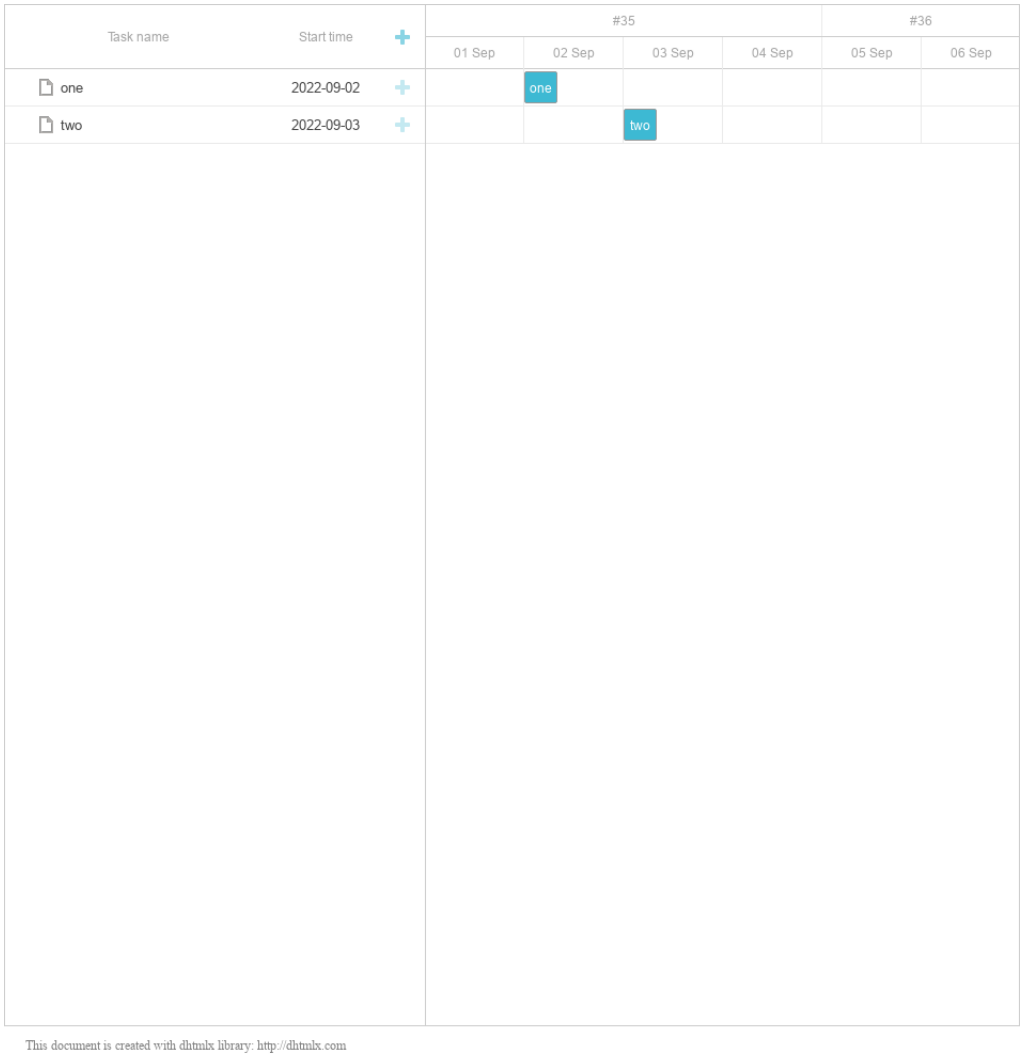
Na alatnoj traci iznad područja gantograma nalazi se gumb "Export as" kojim se otvara dijalog prikazan fotografijom 4.2 koji na izbor ostavlja izvoz gantograma u obliku PNG-a, PDF-a, Excel tablice, iCal i JSON datoteke. Izvoz je izveden korištenjem API metoda biblioteke DHTMLX Gantt: `gantt.exportToPNG()`, `gantt.exportToPDF()`, `gantt.exportTo-ICal()`, `gantt.exportToJSON({ raw: true })` dodavanjem `script` elementa u kontruktor komponente `Export` naredbama

```
const script = document.createElement("script");
script.async = true;
script.src = "http://export.dhtmlx.com/gantt/api.js";
script.id = "ganttApi";
```

Nabrojane metode se pozivaju klikom na odgovarajući gumb. Primjer izgleda izvezene PNG datoteke vidljiv je na slici 4.3. Budući da je izvoz temeljen na API funkcijama DHTMLX Gantt biblioteke, na izvezenim fotografijama, kalendarima i excel i pdf datotekama je moguće vidjeti samo prikaz gantograma odnosno samog zadatka i njegovog vremena početka i kraja, ali su izvozom u JSON tip datoteke vidljivi svi atributi jednog zadatka, svi atributi poveznica te konfiguracija gantograma.

4.6.3. Brisanje trenutnog prikaza i podataka u bazi

Obnovljena je funkcionalnost uklanjanja podataka iz baze i s gantograma u trenutnom projektu koja se postiže pritiskom na gumb `Close`. Kod je refaktoriran i prilagođen novijoj verziji `node`-a - 16.08.



Slika 4.3: Primjer izvezenog gantograma u png formatu

4.6.4. Uvoz postojećeg projekta

Ispravljen je i ponovno omogućen uvoz postojećeg projekta. Prima se JSON datoteka u kojoj se nalaze podaci o zadacima i poveznicama koje treba prikazati na gantogramu. Oblik datoteke vidljiv je na primjeru 4.4.

Primjer 4.4: Primjer izgleda JSON datoteke za uvoz postojećeg projekta

```
1  {"data":
2      [{"id":1,
3      "text":"t1",
4      "start_date":"2022-08-16 00:00",
5      "end_date":"2022-08-16 08:01",
6      "action":"2",
```

```
 7       "progress":0,
 8       "parent":"0",
 9       "failed":[],
10       "preconditions":"",
11       "effects":"",
12       "color":"rgb(61,185,211)"},
13       {"id":2,
14       "text":"t2",
15       "start_date":"2022-08-17 00:00",
16       "end_date":"2022-08-17 08:01",
17       "action":"6",
18       "progress":0,
19       "parent":"0",
20       "failed":[],
21       "preconditions":"",
22       "effects":"",
23       "color":"rgb(61,185,211)"}
24       ],
25   "links":[
26       {"id":1,"source":1,"target":2,"type":"0"}
27       ]
28   }
```

4.6.5. Uvoz i dodavanje akcija

Dodavanje akcija koje predstavljaju podlogu za zadatke omogućeno je klikom na gumb `Import actions` pri čemu se otvara dijalog za odabir datoteke jer je jedini trenutno mogući način dodavanja u obliku JSON datoteke sa svim potrebnim podacima. Oblik datoteke vidljiv je na primjeru 4.5. Trenutno je za uvoz jedino bitno ime akcije, ali ostali atributi ostavljaju prostor za poboljšanje aplikacije.

Primjer 4.5: Primjer izgleda JSON datoteke za uvoz akcija

```
1  {
2  "actions": {
3          "Cyber intelligence gathering": {
4                  "preconditions": "target.name in ['↩
                      Hospital', 'H2O-ST', 'HEP-ODS', 'Bank↩
                      ', 'Newspaper', 'Political party', '↩
```

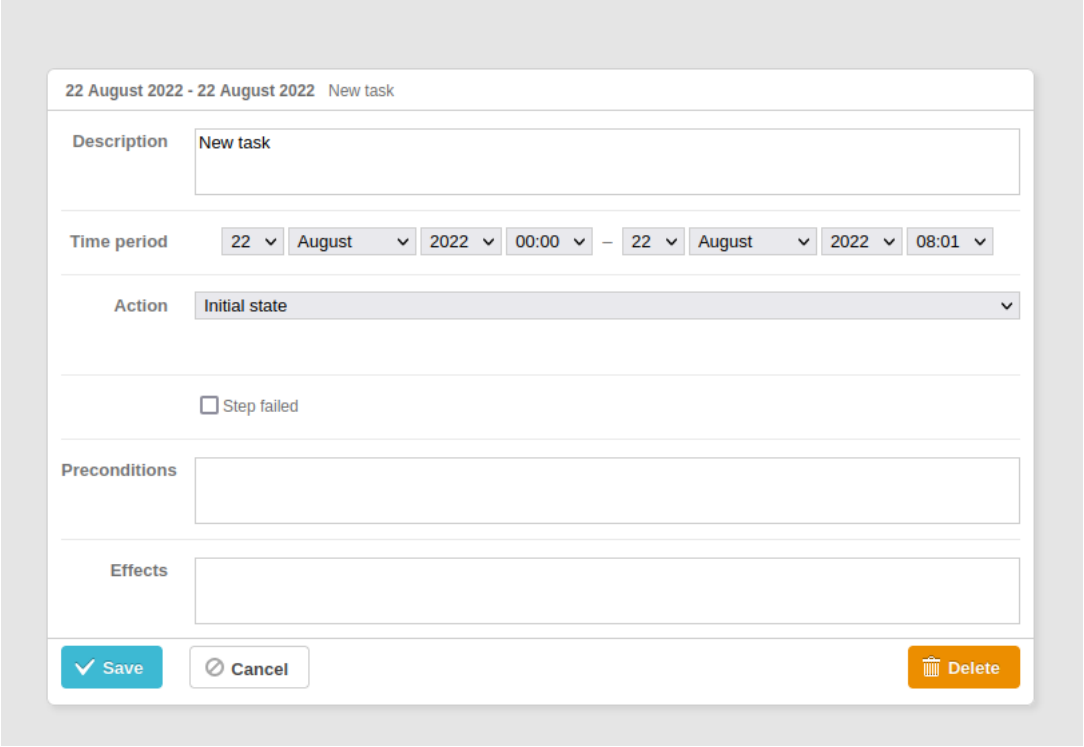
```
                    Social network', 'Politician']",
5                       "posteffects": "target.↩
                           intel_gathered = True",
6                   "action_inject": "inject.log_info = '↩
                    Intelligence gathering successfully ↩
                    finished on ' + target.name + '.'; ↩
                    inject.report = ''; inject.delay = ↩
                    randint(1,2)",
7                   "object_inject": "inject.log_info = ''; ↩
                    inject.report = ''; inject.delay = ↩
                    randint(20, 40)",
8                   "actors": ["Penetester"],
9                   "time": 100,
10                  "price": 100,
11                  "success_rate": 0.5
12          }
13  }
```

4.6.6. Pregled resursa

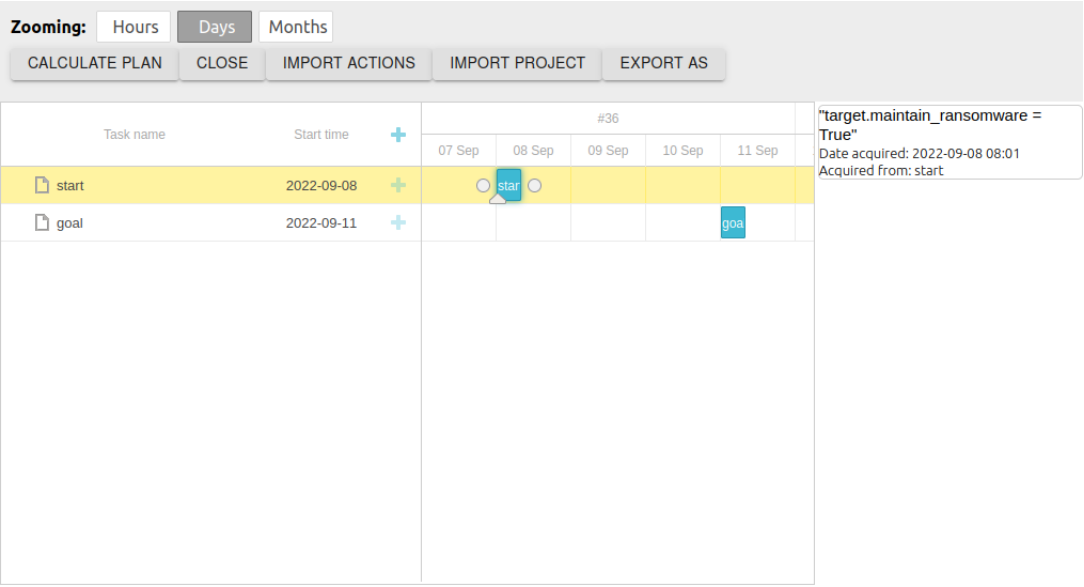
Na desnoj strani prozora nalazi se prikaz prikupljenih resursa. Svaki resurs je rezultat izvođenja zadatka odnosno akcije i predstavlja polje `postconditions` zapisa zadatka u kolekciji `db.tasks`. Svi rezulati dodatno su upisani u bazu u kolekciju `db.effects` radi smanjivanja nepotrebnih ulazaka u bazi pri svakom oslikavanju i svakoj promjeni ili dodatku zadatka. Za svaki se resurs prikazuje kada se stekao. Osim normalnog korištenja alata pri kojem se resursi prikazuju tek izračunom plana moguće je ručno postavljati zadatke i proizvoljne resurse te ih pratiti. Takvo korištenje prikazano je na fotografiji 4.1. Prikaz resursa je jednostavno prilagoditi u ovisnosti o atributima koje zadaci odnosno akcije imaju izmjenom komponente `ResourceView` i metoda `create_task` i `get_effects` u datoteci `server.py`.

4.7. Objašnjenje i primjer rada

Otvaranjem alata moguće je ručno dodavati zadatke i postavljati njihove parametre klikovima. Dijalogom za stvaranje zadatka u tekstualnom okviru moguće je postaviti naziv zadatka koji će se prikazati u gantogramu, padajućim izbornikom datum i



Slika 4.4: Dijalog za stvaranje zadatka



Slika 4.5: Početni izgled sučelja s dva kreirana zadatka

vrijeme početka i datum i vrijeme kraja te tip akcije, potvrdnim okvirom bira se je li zadatak uspješno izveden, u tekstualne okvire se unose uvjeti za izvođenje i razultati koji se postižu izvođenjem. Dijalog za stvaranje zadatka vidjliv je na slici 4.4 U slučaju postavljanja vremena kraja izvođenja zadatka prije vremena početka, vrijeme kraja se

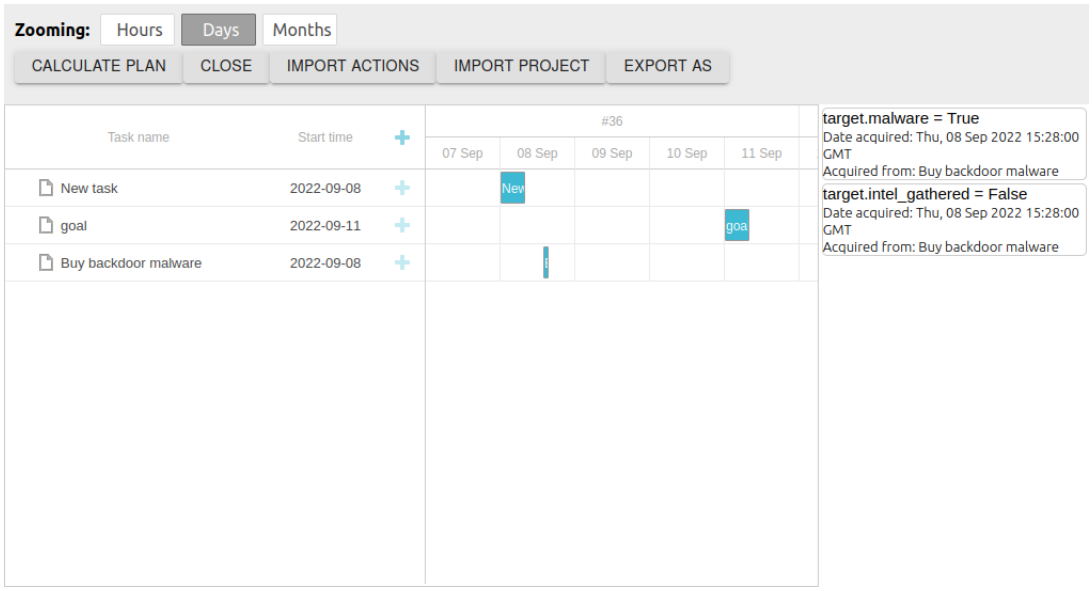
automatski postavlja na jednu minutu nakon odabranog vremena početka. Predviđeni oblik rezultata je

```
<tvrdnja> = <boolean>;
```

, na primjer

```
target.performed_malware_analysis  = True;
```

Predviđeni oblik je moguće jednostavno promijeniti ako se u budućnosti odredi pogodniji način ispitivanja uvjeta. Nakon ručnog stvaranja početnog i ciljnog zadatka, potrebno je uvesti moguće akcije klikom na gumb `Import Actions` i odabirom već pripremljene JSON datoteke napunjene mogućim akcijama. Izgled okvira nakon ovog koraka vidljiv je na slici 4.5. Klikom na gumb `Calculate plan` odredi se totalni plan i prikaže u gantogramu zajedno sa stečenim resursima u pregledu resursa, primjer čega je vidljiv na slici 4.6



Slika 4.6: Izgled sučelja nakon izračuna plana

5. Zaključak

U radu su objašnjeni dijelovi penetracijskog testa i razlozi provedbe kibernetičkih vježbi. Alat opisan u ovom radu olakšava vizualizaciju i planiranje tijeka napada, a dobro može služiti u penetracijskim testovima i kibernetičkim vježbama. Opisana je arhitektura programskog rješenja, popravljene su funkcionalnosti izgradnje kontejnera, uvoza projekta, uvoza akcija, brisanja prikaza i izračuna totalnog plana. Alat je nadograđen opcijom izvoza trenutnog prikaza gantograma i pregledom resursa stečenih izvođenjem zadataka.

Za pravilnu integraciju prethodno opisanog alata s podacima o informacijskom sustavu potrebno je odlučiti kojeg su točno oblika to jest postoji li pogodniji oblik i stvoriti odgovarajuće testne datoteke za uvoz akcija. Nakon toga i odluke što nam je bitno znati o resursima se i pregled resursa može prilagoditi. Osmišljeno programsko rješenje u trenutnom obliku ne dobiva puno dodatne vrijednosti od pregleda resursa budući da je alat namijenjen trenutnom računanju plana na temelju početnog i krajnjeg stanja gdje pregled resursa dobivenih i potrošenih obavljanjem svakog zadatka nije momentalno vidljiv pa ne pomaže odabiru akcija. Da bi pregled resursa imao smisla mogao bi se uvesti pregled potrebnog vremena i ukupna cijena svih izvedenih akcija. Više je preinaka potrebno da bi se korisniku dopustilo da sam bira parcijalni plan koji želi na temelju prikaza stečenih i izgubljenih resursa za svaki plan. Takva preinaka bi smanjila automatizacijsku narav i namjenu osmišljene programske potpore. Još jedan od mogućih smjerova u kojem se alat u budućnosti može prilagoditi je na način da doprinosti rekonstrukciji vremenske crte napada odnosno incidenta koji se već dogodio.

LITERATURA

Mongodb documentation, 2022. URL https://www.mongodb.com/docs/manual/.

DHTMLX. Dhtmlx gantt library documentation, 2022. URL https://docs.dhtmlx.com/gantt/.

Pete Herzog. *Open-Source Security Testing Methodology Manual*. ISECOM, version 3.0 izdanju, 2010. URL https://www.isecom.org/OSSTMM.3.pdf.

Docker Inc. Docker documentation, 2022a. URL https://docs.docker.com/.

Docker Inc. Docker compose documentation, 2022b. URL https://docs.docker.com/compose/.

Meta Platforms Inc. Reactjs documentation, 2022c. URL https://reactjs.org/docs/getting-started.html.

Chris Nickerson, Dave Kennedy, Chris John Riley, Eric Smith, Iftach Ian Amit, Andrew Rabie, Stefan Friedli, Justin Searle, Brandon Knight, Chris Gates, Joe McCray, Carlos Perez, John Strand, Steve Tornio, Nick Percoco, Dave Shackelford, Val Smith, Robin Wood, i Wim Remes. Penetration testing execution standard, 2011. URL http://www.pentest-standard.org/index.php/Main_Page.

OWASP Web Security Testing Guide. OWASP, 2002. URL https://owasp.org/www-project-web-security-testing-guide/.

Pallets. Flask framework documentation, 2010. URL https://flask.palletsprojects.com/en/2.2.x/.

E. Roback. Information systems security assessment framework, 2000.

Programska podrška za planiranje i provođenje kibernetičkih napada na zadanom informacijskom sustavu

Sažetak

Pregled sadržaja penetracijskog testa i svrha kibernetičkih vježbi. Programska podrška za planiranje i vizualizaciju tijeka izvođenja vježbe/napada gantogramom sastoji se od React klijenta, Flask + Python poslužitelja, MongoDB baze, svi su zasebni Docker kontejneri povezani Docker composeom. Prikaz gantograma na klijentu osigurava biblioteka DHMLX Gantt.

Ključne riječi: kibernetičke vježbe, planiranje napada, penetracijski test, programska podrška, React, Flask, gantogram, DHTMLX Gantt

Software support for planning and execution of cyber attacks on a given information system

Abstract

An overview of the contents of a penetratrion test and the purpose of cyber exercises. The software tool for planning and visualisation of an exercise/attack with a gantt chart consists of a React client, Flask server and MongoDB database, all in separate Docker containers connected by Docker compose. The view of the gantt chart on the client is provided by the DHTMLX Gantt library.

Keywords: cyber exercises, attack planning, penetration test, software, React, Flask, gantt chart, DHTMLX Gantt