SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2214

Programska podrška za planiranje i provođenje kibernetičkih napada

Maja Krmpotić Đurđević

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Kako biste uklonili ovu stranicu, obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Kibernetičke vježbe služe kako 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 modificira 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.

Manualna 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. 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.

Rad je strukturiran na sljedeći način; u prvom je poglavlju opisan problem planiranja i provođenja kibernetičkih vježbi, odnosno izazovi koje je potrebno riješiti u sklopu tog problema. Potom slijedi poglavlje u kojem je opisana programska podrška razvijena u sklopu ovog rada, to jest pregled tehnologija korištenih za izradu rješenja, pregled funkcionalnih zahtjeva i opisa njihove implementacije te primjer korištenja. Zatim slijede poglavlje u kojem je objašnjen algoritam parcijalnog poretka i njegova primjena na problem planiranja kibernetičkih vježbi i poglavlje koje opisuje odabir potpunog poretka i konstrukciju plana na temelju parcijalnog poretka generiranog korištenjem algoritma parcijalnog poretka. Iduće poglavlje opisuje na koji je način u programskom rješenju podržana modifikacija plana u slučaju neuspjeha pri izvođenju koraka plana.

2. Planiranje i provođenje kibernetičkih vježbi

2.1. Kibernetičke vježbe

Kibernetičke vježbe su vježbe koje organiziraju tvrtke, države ili savezi s ciljem uvježbavanja sudionika za reakciju na kibernetičke krizne situacije. Cilj reakcije na kibernetičku kriznu situaciju jest minimizacija štete ili ograničavanje napada. Ostali ciljevi kibernetičkih vježbi uključuju, primjerice, jačanje svijesti o sigurnosti korisnika kibernetičkog prostora i sigurnosti kritičnih infrastruktura, poticanje rasprave o smjernicama za postupanje na strateškoj razini i ukazivanje na važnost strateške komunikacije u kriznim situacijama. Kibernetičke vježbe mogu se provoditi u obliku raspravljačkih ili funkcionalnih vježbi [13]. Raspravljačke vježbe zasnivaju se na hipotetskoj situaciji te potiču raspravu kojom sudionici odlučuju o odgovoru na hipotetsku kriznu situaciju. Funkcionalne vježbe simuliraju napade u simulacijskim okruženjima te testiraju odgovor sudionika, ispravnost opreme i komunikacijske vještine sudionika.

Primjer kibernetičke vježbe jest "Cyber Europe" [5], vježba organizirana na razini Europske unije. U vježbi mogu sudjelovati privatni i javni sektor zemalja članica Europske unije i zemalja članica Europske slobodne trgovinske zone. Vježbe "Cyber Europe" su simulacije kibernetičkih incidenata izrađene prema scenarijima zasnovanim na stvarnim događajima, s ciljem edukacije sudionika. Na razini ministara obrane država članica Europske unije u Estoniji se održala vježba "EU CYBRID 2017", a za zemlje saveznice Sjevernoatlanskog saveza održala se vježba "Cyber Coalition 2019"[1] [11], također u Estoniji. U Republici Hrvatskoj 2018. godine održana je kibernetička vježba Koordinacije za sustav domovinske sigurnosti "Kibernetički štit 2018" [10].

Organizacija kibernetičke vježbe započinje formiranjem tima za planiranje kibernetičke vježbe, definiranjem opsega vježbe i određivanjem ciljeva vježbe. Tim za planiranje vježbe identificira potrebu za provođenjem vježbe i bira vrstu vježbe koja će biti provedena. Potom započinje izrada realističnog scenarija koji će biti predstavljen sudionicima vježbe, a koji se

[1] Godine 2019. održana je dvanaesta iteracija vježbe "Cyber Coalition".

sastoji od općenite ideje i posebnih ideja. Općenita ideja može biti prezentirana sudionicima prije početka vježbe i služi kao uvod u tematiku vježbe. Posebne ideje opisuju stanje i događaje prije, za vrijeme i nakon incidenta. Za provođenje plana bira se voditelj vježbe, osoba odgovorna za predstavljanje scenarija sudionicima i održavanje rasprave među sudionicima za vrijeme vježbe. Po završetku vježbe vrši se evaluacija ishoda vježbe u odnosu na zadane ciljeve.

2.2. Planiranje kibernetičkih vježbi

Planiranje kibernetičkih vježbi odvija se u pet koraka i traje nekoliko mjeseci [8]. Prvi korak pri planiranju kibernetičke vježbe jest sastanak za razvoj koncepta vježbe. Na sastanku za razvoj koncepta vježbe izrađuje se inicijalna, općenita skica scenarija vježbe, izabire se vrsta vježbe koja će se provoditi (raspravljačka ili funkcionalna), određuju se ciljevi i definiraju se očekivani ishodi vježbe. Također, identificiraju se organizacije koje će morati sudjelovati na sastanku za inicijalno planiranje, logističke potrebe i potencijalno potrebni resursi. Dva do osam tjedana nakon sastanka za razvoj koncepta vježbe održava se sastanak za inicijalno planiranje na kojem sudjeluju svi interni ili eksterni partneri koji sudjeluju u planiranju vježbe. Na sastanku za inicijalno planiranje radi se osvrt na sastanak za razvoj koncepta vježbe te se određuju ciljevi vježbe, definira scenarij vježbe i određuje koja će organizacije koordinirati logistički plan i resurse. Potom slijedi sastanak za planiranje liste događaja cjelokupnog scenarija koji rezultira skicom cjelokupnog scenarija, dodjeljivanjem akcijskih stavki sudionicima i određivanjem rokova. Na idućem sastanku finalizira se scenarij vježbe, izrađuje se skica logističkog plana i određuje se koja je organizacija zadužena za razvoj materijala za pripremu sudionika vježbe. Sastanak za konačno planiranje vježbe održava se mjesec dana prije provođenja kibernetičke vježbe i rezultira dovršenim scenarijem vježbe, logističkim planom, materijalima za pripremu sudionika vježbe i planom za dostavu resursa na vježbu.

2.3. Planiranje scenarija kibernetičkih vježbi

Scenarij kibernetičke vježbe sastoji se od niza koraka čijim se izvođenjem ostvaruju zadani ciljevi vježbe. Svaki korak scenarija pretpostavlja određeno stanje resursa potrebno za izvođenje tog koraka te po uspješnom izvođenju modificira stanje resursa. U slučaju neuspješnog izvođenja koraka plana ne primjenjuju se modifikacije resursa koje bi uzrokovao taj korak pa se daljnje izvođenje plana nastavlja koracima koji ne pretpostavljaju da su resursi u stanju u kojem bi bili po primjeni te modifikacije. Kako bi scenarij kibernetičke vježbe bio prikladan za korištenje pri izvedbi kibernetičke vježbe nužno mora imati definirane konkuretne planove kojima su pokriveni svi ishodi izvođenja nekog koraka.

U ovom je radu opisana programska podrška za planiranje kibernetičkih vježbi koja u inicijalno modeliranom scenariju pretpostavlja da će svaki korak scenarija biti uspješno izvršen, međutim, u slučaju neuspjeha, odbacuje postojeći dio scenarija koji slijedi nakon neuspješno izvršenog koraka te ga ponovno modelira uzevši zadnji uspješno izvršeni korak kao novo početno stanje. Za modeliranje plana korištenjem opisane programske podrške moraju biti zadovoljene sljedeće pretpostavke: definirano je početno stanje, definirano je ciljno stanje i uvezene su sve atomarne akcije koje je moguće koristiti za modeliranje scenarija. Početno stanje resursa definira se kao posljedica izvođenja početnog stanja, a vrijeme završetka izvođenja početnog koraka određuje početak scenarija. Krajnje ciljeve scenarija potrebno je opisati kao stanje resursa kakvo se očekuje po uspješnom završetku scenarija, a završetak scenarija određen je vremenom početka ciljnog koraka.

Za najveću kontrolu nad tokom scenarija preporuča se definiranje atomarnih akcija, to jest takvih akcija koje se ne mogu smisleno razdijeliti na niz koraka za izvođenje koji utječu na resurse. Ukoliko je moguće promatranu akciju podijeliti na niz koraka za izvođenje, potrebno je svaki od tih koraka definirati kao akciju. U suprotnom bi se u slučaju neuspješnog izvođenja akcije moralo ručno kontrolirati stanje resursa prilikom ponovnog modeliranja dijela scenarija, što zahtjeva domensko znanje osobe koja provodi vježbu i povećava vjerojatnost pojave logičke pogreške.

Kada su zadovoljeni preduvjeti za modeliranje scenarija korištenjem programske podrške korisnik okida modeliranje scenarija pritiskom na gumb. Za planiranje scenarija kibernetičke vježbe programska podrška koristi algoritam planiranja parcijalnog poretka. Odabir akcija implementiran je nedeterministički, dakle svako pokretanje modeliranja scenarija s istim ulaznim parametrima rezultirat će, ako postoji više mogućih scenarija, različitim scenarijem. Parcijalni poredak određuje samo nužna ograničenja kako bi plan bio ispravan te sadrži jedan ili više potpunih poredaka. Na temelju konstruiranog parcijalnog poretka potom se odabire potpuni poredak, također nedeterministički. Nakon odabira potpunog poretka vrši se raspoređivanje akcija u vremenu što rezultira gotovim scenarijem kibernetičke vježbe.

2.4. Provođenje kibernetičkih vježbi

Prilikom provođenja kibernetičke vježbe formiraju se četiri tima. Kibernetičku vježbu vodi grupa za kontroliranje vježbe (engl. *Exercise Control Group*, skraćeno ECG). Tim ECG za vrijeme trajanja vježbe zadaje zadatke sudionicima vježbe koje se trenira (engl. *Training Audience*) i upravlja crvenim timom (engl. *Red Team*, skraćeno RT), zaduženim za provođenje koraka scenarija (engl. *inject*). Nadziranje vježbe provodi tim promatrača (engl. *Observers*) koji na temelju prikupljenih informacija daje timu ECG povratnu informaciju o tijeku vježbe. Ukoliko timovi ECG ili RT primijete poteškoće pri provođenju vježbe, utoliko

provode modifikacije scenarija kako bi provođenje vježbe bilo što efikasnije. Po završetku vježbe, u roku od dvadeset i jednog dana, sudionici koji su planirali vježbu dužni su napisati osvrt na vježbu (engl. *After Action Review*, skraćeno AAR). Dokument AAR služi kako bi se zabilježile naučene lekcije te se temeljem njega definiraju željeni ishodi nadolazećih vježbi.

3. Opis programske podrške za planiranje i provođenje kibernetičkih napada

3.1. Struktura programskog rješenja

Programsko rješenje za planiranje i provođenje kibernetičkih vježbi opisano u ovom radu implementirano je kao web aplikacija zasnovana na arhitekturi klijent-poslužitelj. Klijent je implementiran korištenjem React JavaScript razvojnog okvira te se sav pripadni izvorni kod klijenta nalazi u direktoriju `client`. Prilikom implementacije klijenta korištena je DHTMLX Gantt biblioteka za prikaz gantograma. Poslužitelj je implementiran korištenjem razvojnog okvira Flask, a za spremanje podataka korištena je baza podataka MongoDB. Izvorni kod poslužitelja nalazi se u direktoriju `server`. Klijent i poslužitelj komuniciraju preko aplikacijskog programskog sučelja (engl. *Application Programming Interface*, skraćeno API) za reprezentacijsko stanje prijenosa (engl. *Representational State Transfer*, skraćeno REST). Kako bi se osigurali ispravan rad i isporuka (engl. *deploy*) aplikacije korišten je alat Docker. Klijent i poslužitelj imaju svoje Docker datoteke, a za upravljanje s više Docker kontejnera (engl. *container*) korišten je alat Docker Compose, odnosno definirana je YAML datoteka na razini projekta. Pomoću YAML datoteke određeni su i pristupi (engl. *port*) na kojima slušaju klijent i poslužitelj.

3.2. Tehnologije korištene za izradu programskog rješenja

3.2.1. Docker

Alat Docker služi za izradu, isporuku i pokretanje aplikacije korištenjem kontejnera. U direktoriju `client` nalazi se Docker datoteka čiji je sadržaj prikazan izvornim kodom 3.1.

```
1 FROM node:10.15-alpine
2
```

```
3  WORKDIR /client
4
5  COPY package.json ./
6
7  RUN apk add yarn
8  RUN yarn install
9  RUN yarn global add serve
10
11 COPY ./public ./public
12 COPY ./src ./src
13
14 EXPOSE 5000
15
16 CMD yarn build && serve -s build
```

Izvorni kod 3.1: Dockerfile za upravljanje kontejnerom za klijenta

Naredbom `FROM` u prvoj liniji datoteke stvara se sloj slike (engl. *image layer*) iz Docker slike `node:10.15-alpine`. Naredbom `WORKDIR` određuje se radni direktorij, a naredbama `COPY` dodaju se nove datoteke s izvorišta, to jest prve navedene putanje, u kontejner. Naredbom `RUN` inicira se instalacija navedene ovisnosti (engl. *dependency*) aplikacije. Naredbom `EXPOSE 5000` određuje se da kontejner sluša na pristupu 5000, a naredbom `CMD` određena je pretpostavljena naredba za izvršavanje Docker slike. Analogno opisanoj Docker datoteci na klijentu, definirana je i Docker datoteka u `server` direktoriju.

Alat Docker Compose korišten je za upravljanje klijentskim i poslužiteljskim kontejnerima te se pomoću njega, korištenjem naredbe `docker-compose up -build` izvršava izgradnja (engl. *build*) slike programskog rješenja na temelju postojećeg izvornog koda. Jednom kada je slika izrađena kontejneri koji čine programsko rješenje pokreću se naredbom `docker-compose up`. Datoteka `docker-compose.yml` prikazana je izvornim kodom 3.2.

```
1  version: "2.4"
2
3  services:
4    mongo:
5      image: mongo:4.0-xenial
6      volumes:
7        - data:/data/db
8    flask:
9      build: ./server
10     ports:
11       - "8080:8080"
12     depends_on:
13       - mongo
```

```
14  react:
15    build: ./client
16    stdin_open: true
17    ports:
18      - "5000:5000"
19    depends_on:
20      - flask
```

Izvorni kod 3.2: Sadržaj datoteke `docker-compose.yml`

U datoteci `docker-compose.yml` definirani su korišteni kontejneri korištenjem ključne riječi `services`. Ključnom riječi `image` određuje se Docker slika koja se koristi za izgradnju kontejnera, a ključna riječ `build` koristi se za definiranje direktorija u kojem se nalazi navedeni kontejner, odnosno njegova Docker datoteka. Ključnom riječi `ports` određuju se pristupi na kojima kontejner sluša, a ključnom riječi `depends_on` definiraju se međuovisnosti među navedenim kontejnerima.

3.2.2. MongoDB

Upravljanje podacima, odnosno spremanje podataka generiranih korištenjem programskog rješenja vrši se korištenjem sustava za upravljanje bazama podataka MongoDB. Baza podataka MongoDB je nerelacijska baza podataka, zasnovana na modelu dokumenata, koja podatke sprema u BSON formatu. Pristup bazi podataka s poslužitelja ostvaren je korištenjem PyMongo biblioteke.

Strukture dokumenata

Entiteti korišteni pri radu s programskom podrškom opisanom u ovom radu su zadaci, poveznice, akcije i parcijalni poretci. Zadatak predstavlja korak plana enkapsuliran u zadatak koji se prikazuje u gantogramu. Atributi (engl. *property*) relevantni za gantogram su `start_date` (početno vrijeme izvođenja zadatka), `end_date` (vrijeme završetka izvođenja zadatka), `taskid` (identifikacijski broj zadatka u gantogramu), `progress` (trenutni napredak izvođenja zadatka) i `duration` (trajanje izvođenja zadatka u minutama). Atributi kojima je modeliran korak scenarija su `action` (identifikacijski broj akcije koja se izvodi ovim korakom), `preconditions` (preduvjeti za izvođenje koraka plana), `effects` (posljedice izvođenja koraka plana) i `failed` (zastavica koja indicira neuspješno izvođenje koraka plana).

Poveznice služe kako bi se modelirao odnos među zadacima. Poveznica sadrži atribute `link_id` (identifikacijski broj poveznice u gantogramu), `source` (identifikacijski broj zadatka koji je izvorište poveznice), `target` (identifikacijski broj zadatka koji je odredište poveznice) te `type` (oznaka vrste poveznice). Vrsta poveznice u ovom je rješenju nužno

end_to_start, čija je oznaka "0", koja označava da poveznica izvire iz završetka izvorišnog zadatka, a završava u početku odredišnog zadatka.

Akcije su određene atributima action_id (identifikacijski broj akcije), name (naziv akcije), preconditions (preduvjeti za izvođenje akcije), posteffects (posljedice izvođenja akcije) i time (vrijeme potrebno za izvođenje akcije u minutama).

Parcijalni poredak jest entitet koji čuva podatke korištene prilikom planiranja parcijalnog poretka, a to su steps (koraci parcijalnog poretka), causal_links (ograničenja nad kojima se može pojaviti prijetnja), ordering_constraints (ograničenje poretka parova koraka) i successful (zastavica koja indicira uspješno konstruiranje ispravnog plana).

Za održavanje sekvencijalnih identifikacijskih brojeva koristi se pomoćni counters dokument, unutar kojeg se pamti zadnji dodijeljeni identifikacijski broj za zadatke, poveznice i akcije.

3.2.3. Flask

Za implementaciju poslužitelja korišten je web razvojni okvir Flask i programski jezik Python. Poslužitelj se sastoji od API metoda za komunikaciju s klijentom i od metoda za izvršavanje poslovne logike programskog rješenja. Izvorni kod poslužitelja nalazi se u datoteci server.py.

Aplikacijsko programsko sučelje

Aplikacijsko programsko sučelje poslužitelja sastoji se od metoda propisanih u dokumentaciji biblioteke DHTMLX Gantt nužnih za integraciju s poslužiteljem te dodatnih metoda specifičnih za opisanu programsku podršku. Biblioteka DHTMLX Gantt propisuje GET, POST, PUT i DELETE metode za dohvat, dodavanje, uređivanje i brisanje zadataka i poveznica. Navedene metode implementirane su na način da traženu modifikaciju izvedu u bazi podataka. Također, postoji i POST metoda import_actions za uvoz akcija koja u tijelu zahtjeva prima podatke o akcijama u JSON formatu te GET metoda get_actions za dohvat akcija koje se nalaze u bazi podataka. Postojeće zadatke i poveznice prikazane u gantogramu moguće je obrisati korištenjem metode clear_gantt_actions. Uvoz postojećeg projekta spremljenog u JSON formatu može se izvršiti pomoću metode import_existing_project. Planiranje scenarija kibernetičke vježbe izvršava se pozivom metode get_plan.

Metode za izvršavanje poslovne logike

Poziv API metode get_plan okida izvršavanje poslovne logike aplikacije vezane za planiranje scenarija kibernetičke vježbe. Izvršavanje metode get_plan započinje pozivom metode clean_previous_plan_if_exists koja osigurava da će biti obrisano sve

osim početnog i ciljnog stanja. Potom se poziva metoda `plan_gantt_actions` koja kao ulazni parametar prima identifikacijski broj akcije koja označava početno stanje. Kada se metoda `plan_gantt_actions` poziva u metodi `get_plan` kao ulazni parametar prosljeđuje joj se broj jedan jer postoje mehanizmi koji osiguravaju da će to biti identifikacijski broj akcije `Initial state`. Metoda `plan_gantt_actions` priprema strukturu plana parcijalnog poretka, konstruira početni i ciljni korak, priprema listu ciljeva i listu mogućih akcija te poziva metodu `partial_order_planner`. Metoda `partial_order_planner` izvršava algoritam planiranja parcijalnog poretka te vraća objekt koji sadrži sve podatke o parcijalnom poretku. Ukoliko je pronalazak parcijalnog poretka bio neuspješan, utoliko se klijentu vraća odgovor da pokušaj planiranja scenarija nije bio uspješan. Ako je parcijalni poredak uspješno pronađen poziva se metoda `construct_gantt_total_order_plan` koja na temelju parcijalnog poretka odabire jedan potpuni poredak, raspoređuje korake u vremenu i konstruira zadatke za prikaz scenarija gantogramom. Odabir potpunog poretka izvršava se pozivom metode `construct_total_order`. Ukoliko je moguće rasporediti korake potpunog poretka u raspoloživom vremenu, utoliko se spremaju zadaci za gantogram i vraća klijentu odgovor o uspješnom planiranju scenarija kibernetičke vježbe. U suprotnom, klijentu se vraća odgovor o neuspješnom planiranju scenarija s danim ulaznim parametrima.

Osim navedenih metoda postoje još i pomoćne metode poput metoda za utvrđivanje međuovisnosti koraka poretka `get_recursive_predecessors`, koja vraća sve prethodnike danog koraka i prethodnike njihovih prethodnika, i `get_recursive_successors`, koja vraća sve sljedbenike danog koraka i sljedbenike njegovih sljedbenika. U slučaju kada klijent označi izvođenje nekog koraka neuspješnim poziva se metoda `recalculate_plan` koja radi analogno metodi `get_plan` uz razliku da briše samo dio scenarija koji dolazi nakon neuspješno izvedenog koraka te kao početno stanje uzima zadnji uspješno izvedeni korak.

3.2.4. Razvojni okvir React JavaScript

Za implementaciju klijenta u opisanom je programskom rješenju korišten razvojni okvir React JavaScript. Aplikacija klijenta stvorena je korištenjem alata `Create React App` koji konfigurira projekt i brine se o ovisnostima (engl. *dependency*) projekta. Vršna komponenta jest `App`, a u njezinoj `render` metodi inicijaliziraju se još i komponente `Toolbar` i `Gantt`. Komponenta `Toolbar` predstavlja alatnu traku nad gantogramom te nudi opcije postavljanja željene granulacije vremena u gantogramu, uvoza akcija, uvoza postojećeg projekta, brisanja postojećeg gantograma, okidanja planiranja scenarija i izvoza postojećeg scenarija u nekoliko različitih formata. Komponenta `Gantt` služi kao enkapsulacija elementa za rad s gantogramima dostupne u DHTMLX Gantt biblioteci, što se postiže naredbom `im-`

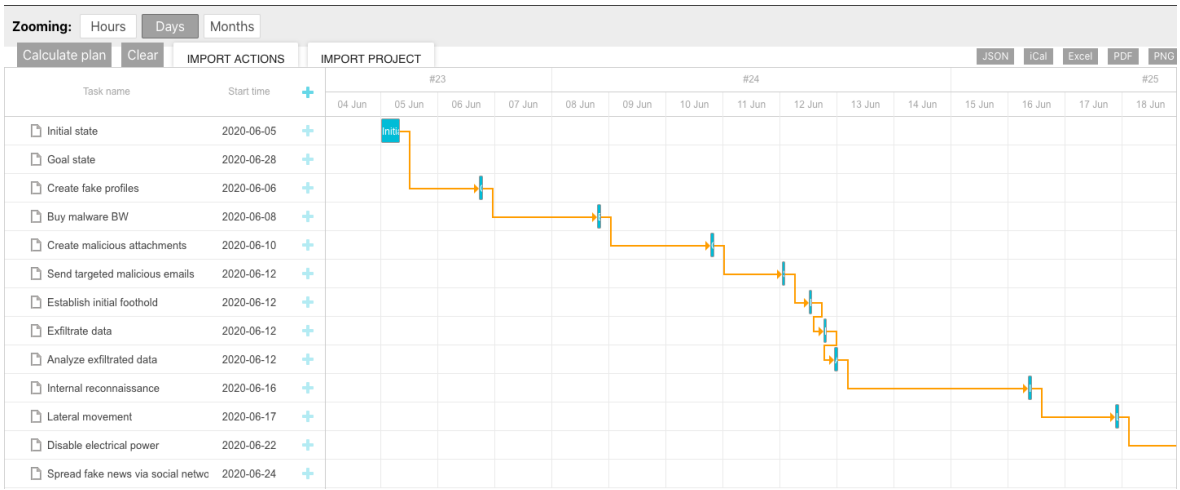
`port { gantt } from 'dhtmlx-gantt'`. Inicijalizacija elementa koji prikazuje gantogram ostvarena je u metodi životnog ciklusa komponente (engl. *lifecycle*) `componentDidMount` korištenjem naredbe `gantt.init(this.ganttContainer)`. Integracija s poslužiteljem izvršena je odmah po inicijalizaciji elementa pozivom metode definirane u `Gantt` komponenti, `initGanttDataProcessor`, koja inicijalizira element za obradu podataka s informacijama o URL adresi poslužitelja, pristupu i načinu komunikacije. Inicijalizirani element za obradu podataka potom se prosljeđuje elementu `gantt` naredbom `this.dataProcessor.init(gantt)`. U metodi `componentDidMount` vrši se i konfiguracija gantograma i pripadajućih formi za rad s elementima gantograma.

3.2.5. Biblioteka DHTMLX Gantt

Za prikaz gantograma na korisničkom sučelju korištena je JavaScript biblioteka DHTMLX Gantt. Atributi `gantt` elementa korišteni pri izradi programskog rješenja opisanog u ovom radu su `gantt.config`, korišten za konfiguraciju gantograma, `gantt.attachEvent`, korišten za dodavanje funkcija koje se izvršavaju kada se dogodi određeni događaj, atribut `gantt.config.lightbox.sections`, korišten za definiranje atributa zadataka i `gantt.locale.labels`, korišten za dodavanje naziva elemenata. Za omogućavanje izvoza gantograma u PDF, Excel, PNG, iCal ili JSON datoteku korištena je dodatna, eksterna skripta dostupna na URL adresi `http://export.dhtmlx.com/gantt/api.js`.

3.3. Funkcionalni zahtjevi

Korisničko sučelje programske podrške opisane u ovom radu vidljivo je na slici 3.1.



Slika 3.1: Korisničko sučelje programske podrške za planiranje i provođenje kibernetičkih napada

3.3.1. Osnovne funkcionalnosti za rad s gantogramom

Dodavanje zadatka

Dodavanje zadatka u gantogram vrši se klikom na ikonu u obliku znaka plus u zaglavlju popisa zadataka vidljivom na slici 3.1. Pritiskom na ikonu otvara se prozor s označnim mjestima za unos potrebnih podataka prikazan na slici 3.2. Prvo polje forme, naziva `Description`, odnosi se na naziv ili opis zadatka. Korisnik zatim odabire datum početka zadatka i datum dovršetka zadatka u polju naziva `Time period`. Potom slijedi odabir akcije u padajućem izborniku naziva `Action` i definiranje preduvjeta i posljedica koraka u poljima `Preconditions` i `Effects` u formatu *"naziv prvog preduvjeta ili posljedice = True ili False; naziv drugog preduvjeta ili posljedice = True ili False; ..."*. Ne preporuča se korištenje polja s nazivom `Step failed` prilikom dodavanja zadatka zbog pretpostavke da su dodani zadaci planirani za izvođenje u budućnosti te kao takvi ne mogu biti neuspješno izvršeni.

Slika 3.2: Prozor za dodavanje i uređivanje zadataka

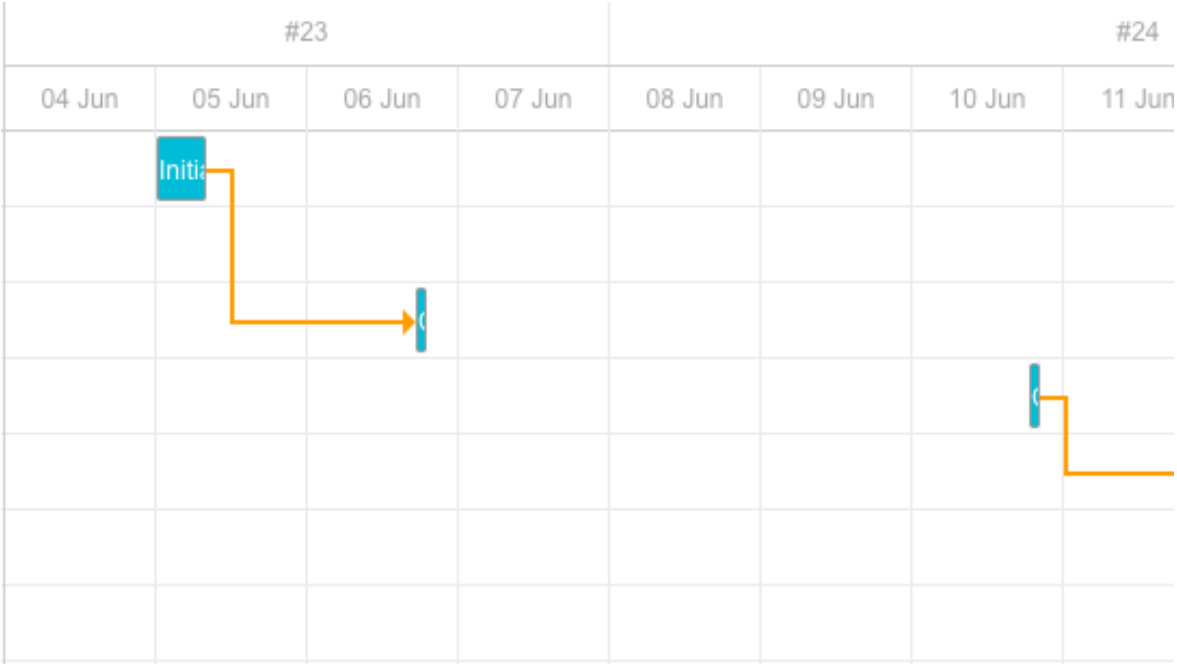
Uređivanje zadatka

Uređivanje zadatka započinje dvostrukim klikom na zadatak s popisa ili gantograma koji se želi urediti. Tada se otvara forma prikazana na slici 3.2 gdje je moguće uređivati sve podatke vidljive na formi. Kada su uređeni svi podaci čije je uređivanje zamišljeno promjene se

pohranjuju pritiskom na gumb `Save`. Ukoliko je označeno polje `Task failed` izvršit će se ponovno planiranje dijela scenarija između neuspješno izvršenog zadatka i ciljnog zadatka.

Brisanje zadatka

Brisanje zadatka vrši se otvaranjem forme za uređivanje zadatka te pritiskom na gumb `Delete`, vidljivom na slici 3.2. Uspješno brisanje zadatka rezultira uklanjanjem zadatka iz gantograma kao što je vidljivo na slici 3.3.



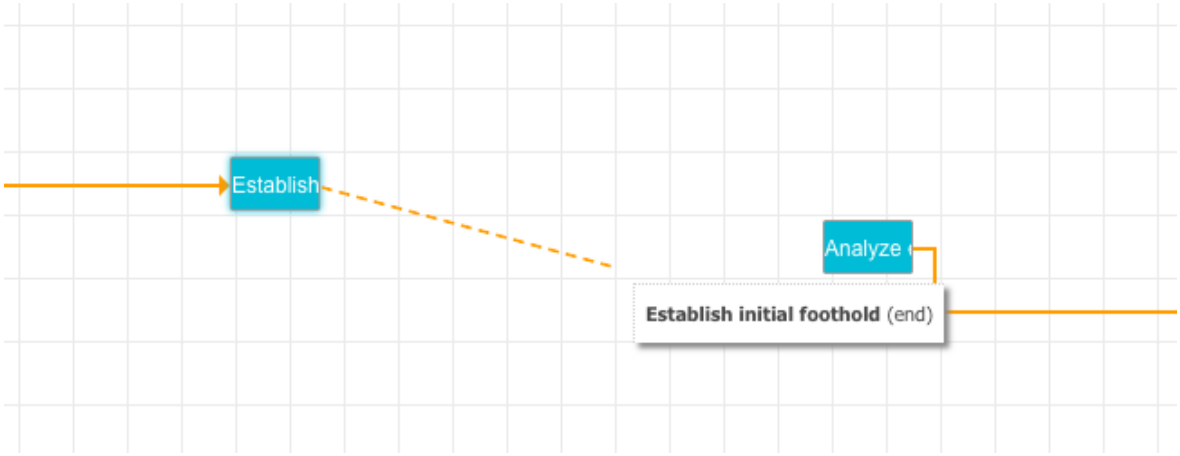
Slika 3.3: Prikaz gantograma nakon brisanja zadatka

Dodavanje poveznica između zadataka

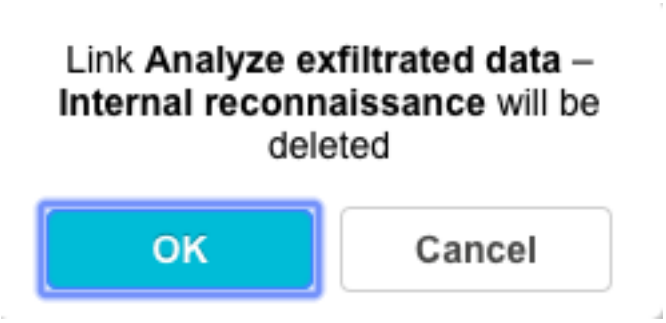
Dodavanje nove poveznice u gantogram vrši se na način da se prijeđe kursorom preko kraja željenog izvorišnog zadatka te se po pojavi kružića pritiskom na kružić povuče poveznica do odredišnog zadatka. Dodavanje poveznice prikazano je na slici 3.4.

Brisanje poveznice između zadataka

Brisanje poveznice između zadataka vrši se dvostrukim klikom na poveznicu koju treba obrisati. Potom se otvara prozor prikazan na slici 3.5 gdje je potrebno kliknuti na gumb `OK` za potvrdu namjere brisanja poveznice, ili `Cancel` za odustajanje od brisanja poveznice.



Slika 3.4: Prikaz dodavanja poveznice među zadacima



Slika 3.5: Prikaz prozora za potvrdu brisanja poveznice među zadacima

Vremenska granulacija prikaza vremena

Programska podrška opisana u ovom radu nudi tri opcije prilikom odabira granulacije prikaza vremena u gantogramu, a željena se opcija odabire u izborniku `Zooming` dostupnom u alatnoj traci. Alatna traka gantograma prikazana je na slici 3.6. Ponuđene opcije za granulaciju vremena su sati (engl. *hours*), dani (engl. *days*) i mjeseci (engl. *months*) te svaka ima pripadajući gumb u izborniku. Granulacija vremena po satima prikazana je na slici 3.7, po danima na slici 3.8, a po mjesecima na slici 3.9.

Slika 3.6: Prikaz alatne trake

Slika 3.7: Granulacija vremena u satima

Slika 3.8: Granulacija vremena u danima

3.3.2. Funkcionalnosti za izradu i modifikaciju scenarija kibernetičke vježbe

Uvoz akcija

Za uvoz akcija potrebno je pripremiti JSON datoteku koja sadrži sve potrebne informacije za uvoz akcija. Primjer JSON datoteke koja ilustrira ispravan format datoteke prikazana je u nastavku. Pritiskom na gumb `Import actions` vidljiv na slici 3.6 otvara se izbornik u kojem je potrebno odabrati pripremljenu JSON datoteku. Kada su akcije uspješno uvezene bit će vidljive u padajućem izborniku `Action` vidljivom na slici 3.2.

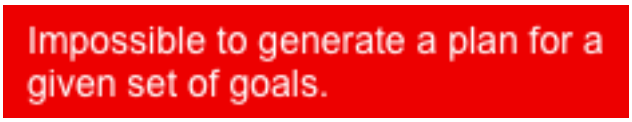
```
{
  "actions": {
    "Cyber intelligence gathering": {
      "preconditions": "",
      "posteffects": "intel_gathered = True",
      "time": 100,
      "price": 100
    },
    "Create ransomware WC2": {
      "preconditions": "intel_gathered = True",
      "posteffects": "ransomware = True; intel_gathered =
        False",
      "time": 100,
      "price": 100
    }
  }
}
```

Izvorni kod 3.3: Primjer datoteke za uvoz akcija

Slika 3.9: Granulacija vremena u mjesecima

Planiranje i provođenje scenarija

Nakon što su uvezene sve potrebne akcije i definirani početni i ciljni zadatak pokreće se planiranje scenarija kibernetičke vježbe pritiskom na gumb `Calculate plan`, vidljiv na slici 3.6. Ukoliko programska podrška nije uspjela napraviti plan scenarija koji ispunjava sve ciljeve u zadanom vremenskom intervalu, utoliko se u gornjem desnom kutu ispisuje poruka o neuspjehu vidljiva na slici 3.10. Ako je scenarij uspješno isplaniran prikazuje se u obliku zadataka i poveznica u gantogramu, što je vidljivo na slici 3.1. Nakon što je scenarij uspješno generiran provođenje vježbe vrši se uređivanjem napretka zadataka.



Slika 3.10: Poruka o neuspješnom pokušaju planiranja scenarija kibernetičke vježbe

Ponovno modeliranje scenarija u slučaju neuspješnog izvođenja koraka

Ako voditelj kibernetičke vježbe prilikom provođenja vježbe procijeni da neki zadatak nije uspješno izvršen zbog prikladne reakcije sudionika vježbe, označava taj zadatak oznakom `Step failed` čime se okida ponovno planiranje scenarija vježbe od tog zadatka nadalje. Za vrijeme trajanja planiranja novog scenarija na poslužitelju, na korisničkom se sučelju u gornjem desnom kutu ispisuje poruka koja o tome obavještava krajnjeg korisnika, to jest voditelja vježbe.

3.3.3. Dodatne funkcionalnosti

Izvoz gantograma

Na alatnoj traci prikazanoj na slici 3.6 nalaze se, s desne strane, gumbi koji služe za izvoz gantograma u brojnim formatima. Prvi gumb služi za izvoz gantograma u JSON formatu, koji se potom može ponovno uvesti u aplikaciju u bilo kojem trenutku. Za prikaz zadataka gantograma u kalendaru gantogram se može izvesti u iCal formatu, a za izvoz grafičkog prikaza gantograma moguće je koristiti izvoz u PDF ili PNG formatu. Zadatke je moguće izvesti i u Excel formatu. Gantogram se u odabranom formatu izvozi pritiskom na gumb čiji je naziv ujedno i naziv odgovarajućeg formata.

Uvoz postojećeg projekta

Postojeći projekt, pohranjen u JSON formatu, moguće je uvesti u aplikaciju pritiskom na gumb `Import project` vidljiv na slici 3.6. Odabirom opcije za uvoz postojećeg projekta otvara se izbornik u kojem je moguće odabrati datoteku koja sadrži projekt, a odabir se potvrđuje pritiskom na gumb `Open`. Po uspješnom uvozu projekta briše se trenutni gantogram i prikazuju se zadaci i poveznice iz uvezenog projekta.

4. Primjena algoritma planiranja parcijalnog poretka za pronalazak plana kibernetičke vježbe

4.1. Automatizirano planiranje i raspoređivanje

Automatizirano planiranje i raspoređivanje je grana umjetne inteligencije koja se bavi pronalaskom slijeda akcija čijim se izvođenjem postiže zadani cilj. Prilikom odabira algoritma za planiranje potrebno je definirati problem planiranja koji se rješava. Problem planiranja opisan u ovom radu ima konačan broj početnih stanja, diskretna stanja i determinističke akcije s određenim trajanjem izvođenja akcije. Akcije se ne izvode konkuretno već jedna iza druge, a konstruiranje plana vrši se za samo jednog agenta. Cilj primjene automatiziranog planiranja na navedeni problem jest postizanje zadanog cilja. Opisanom problemu planiranja odgovara primjena algoritma planiranja parcijalnog poretka, uz naknadno raspoređivanje akcija u vremenu.

4.2. Algoritam planiranja parcijalnog poretka

Planiranje parcijalnog poretka je vrsta automatiziranog planiranja čiji se mehanizam rada sastoji od pronalaženja parcijalnih poredaka među akcijama i obvezivanjem na odabir poretka samo kada je to neizbježno (engl. *least commitment*). Algoritam planiranja parcijalnog poretka je algoritam koji kao ulazne parametre prima početno stanje, ciljno stanje i moguće akcije, a rezultira konstruiranjem parcijalnog poretka i potragom za potpunim rješenjem. Algoritam je ispravan (engl. *sound*) i potpun (engl. *complete*). Ispravnost algoritma označava svojstvo da se u skupu rješenja nužno nalaze samo ispravna rješenja, a svojstvo potpunosti garantira da će se u skupu rješenja nalaziti sva ispravna rješenja.

Tijekom rada algoritam planiranja parcijalnog poretka održava strukturu plana parcijalnog poretka koja se sastoji od skupa akcija, skupa poredaka parova akcija, skupa kauzalnih poveznica i skupa nezadovoljenih preduvjeta. Skup akcija jest skup akcija koje su odabrane

za konstrukciju parcijalnog poretka. Skup poredaka parova akcija čuva podatke o poretku parova akcija koji mora biti zadovoljen u konačnom parcijalnom poretku. Skup kauzalnih poveznica sastoji se od podataka o potencijalno ugroženim vezama između dvije akcije i preduvjetu koji ih čini ranjivima. Ukoliko je dodavanjem nove akcije u skup akcija ugrožena neka postojeća kauzalna poveznica, utoliko se provodi razrješavanje prijetnje primjenom jednog od mehanizama razrješavanja prijetnje. Mehanizmi razrješavanja prijetnje su razrješavanje prijetnje promocijom (engl. *promotion*) i razrješavanje prijetnje degradacijom (engl. *demotion*). Razrješavanje prijetnje promocijom nalaže da se akcija koja ugrožava poveznicu smješta u poretku nakon odredišne akcije ugrožene poveznice. Razrješavanje prijetnje degradacijom nalaže da se akcija koja ugrožava poveznicu smješta u poretku prije izvorišne akcije ugrožene poveznice. Skup nezadovoljenih preduvjeta, to jest skup ciljeva sadržava informaciju o svim nezadovoljenim preduvjetima u trenutnom parcijalnom poretku. Algoritam završava s radom kada je skup nezadovoljenih preduvjeta prazan. Tijek planiranja parcijalnog poretka prikazan je algoritmom 1.

4.3. Primjena algoritma planiranja parcijalnog poretka za planiranje kibernetičkih vježbi

Problem planiranja kibernetičke vježbe opisan u ovom radu započinje definiranjem početnog stanja, završnog stanja i mogućih akcija te je na njega primjenjiv algoritam planiranja parcijalnog poretka čiji su to ulazni parametri. Modifikacija algoritma planiranja parcijalnog poretka prilikom prilagodbe algoritma na primjenu u domeni planiranja kibernetičkih vježbi uzrokovala je gubitak svojstva potpunosti. Slučaj u kojem je narušeno svojstvo potpunosti nastupa kada se nakon konstrukcije parcijalnog poretka utvrdi da nije moguće konstruirati potpuni poredak koji bi bio izvršiv u zadanom vremenskom intervalu, a postoji neki drugi parcijalni poredak koji to ograničenje ne bi narušio. Iako time nije narušena potpunost samog algoritma, narušena je potpunost planiranja kibernetičke vježbe. Algoritam je u programskom rješenju implementiran unutar metode `partial_order_planner`. Potpis metode i implementacija prvog koraka algoritma, provjera postoji li nezadovoljeni cilj u skupu ciljeva, prikazani su izvornim kodom 4.1.

```
1 def partial_order_planner(plan_problem, goals, actions):
2     # 1. if G is empty terminate and return plan
3     if len(goals) == 0:
4         plan_problem['successful'] = True
5         return plan_problem
```

Izvorni kod 4.1: Isječak izvornog koda metode `partial_order_planner` kojim je implementiran prvi korak algoritma

Algoritam 1 Algoritam planiranja parcijalnog poretka

A – skup mogućih akcija
S – skup koraka
O – ograničenja poretka
L – kauzalne poveznice
Izlaz: $plan$ – parcijalni poredak
repeat
  if G := $\emptyset$ **then**
    return plan
  end if
  izaberi {c,s} $\in$ G
  if postoji $s_i \xrightarrow{\text{e, } \neg \text{ c}}$ s **then**
    plan.successful := false
    return plan
  end if
  nedeterministički izaberi s $\in$ S ili a $\in$ A $\rightarrow$ s_s s efektom e
  if s_s ne postoji **then**
    plan.successful := false
    return plan
  else
    L' := L $\cup$ $\{s_s \xrightarrow{\text{e,c}} \text{S}\}$
    O' := O $\cup$ $\{s_s < \text{S}\}$
    G' := G - {c,s}
    S' := S $\cup$ $\{s_s\}$
  end if
  c_s := preduvjeti od s_s
  for all $c \in c_s$ **do**
    G := G $\cup$ {c, s_s}
  end for
  for all $l \in L$ **do**
    if s_s prijeti l **then**
      primijeni promociju ili degradaciju
    end if
  end for
until G != $\emptyset$

Izvorni kod 4.2 prikazuje implementaciju drugog koraka algoritma, to jest odabir preduvjeta koji će se zadovoljiti u ovoj iteraciji algoritma i provjera postoji li konflikt zbog kojeg je taj preduvjet nemoguće zadovoljiti. Navedeno je ostvareno nasumičnim odabirom cilja iz ciljnog skupa te provjerom postoji li kauzalna poveznica s odredištem u stanju trenutnog cilja, a posljedicom suprotnom preduvjetu stanja trenutnog cilja. Ukoliko takva kauzalna posljedica postoji, utoliko metoda označava plan neuspješnim i vraća rezultat.

```
    # 2. select {c,S} e G
    current_goal = random.choice(goals)
    current_goal_precondition = current_goal['c']

    # 2.a if there's a link "S[i] -(e, not c)-> S" in causal links L fail -> IMPOSSIBLE
    PLAN
    causal_links = plan_problem['causal_links']
    contains_contradiction = False
    for link in causal_links:
        if link['target']['step_id'] == current_goal['S']['step_id'] \
                and link['c']['name'] == current_goal_precondition['name'] \
                and link['c']['value'] != current_goal_precondition['value']:
            contains_contradiction = True

    if contains_contradiction:
        print("Contains contradiction.", flush=True)
        plan_problem['successful'] = False
        return plan_problem
```

Izvorni kod 4.2: Isječak izvornog koda metode `partial_order_planner` kojim je implementiran drugi korak algoritma

Treći i četvrti korak algoritma, to jest nasumičan odabir akcije, ažuriranje problema planiranja i ažuriranje ciljeva, u metodi su ostvareni izvornim kodom 4.3. Prvo se određuju akcije čije posljedice odgovaraju preduvjetima trenutnog cilja, to jest ispravne akcije. Ukoliko je skup ispravnih akcija prazan, utoliko se plan označava neuspješnim i vraća se rezultat. Ako postoje ispravne akcije, nasumično se odabire jedna te se provjerava postoji li odabrana akcija u postojećim koracima parcijalnog poretka. Ako akcija ne pripada niti jednom koraku, definira se novi korak i dodaje u skup koraka parcijalnog poretka. Potom se vrši ažuriranje problema planiranja parcijalnog poretka: trenutni cilj izbacuje se iz skupa ciljeva, dodaju se novi ciljevi za svaki preduvjet trenutne akcije, dodaje se poredak koji nalaže da se trenutni korak izvodi prije trenutnog ciljnog koraka i dodaje se kauzalna poveznica koja indicira da su trenutno stanje i trenutno ciljno stanje povezane preduvjetom trenutnog cilja.

```
    # 3. nondeterministically select step S or action with effect e
    #   if there is no such action fail -> IMPOSSIBLE PLAN
    #   otherwise update planning problem
    valid_actions = where_contains_effect(actions, current_goal_precondition['name'],
    current_goal_precondition['value'])

    valid_action_found = False
    while not valid_action_found:
```

```
        if len(valid_actions) == 0:
            print("No valid actions.", flush=True)
            plan_problem['successful'] = False
            return plan_problem

        selected_action = random.choice(valid_actions)

        if selected_action is None:
            print("Selected action is None.", flush=True)
            plan_problem['successful'] = False
            return plan_problem

        selected_step = next((x for x in plan_problem['steps'] if x['step_id'] ==
    selected_action['action_id']), None)
        if selected_step is None:
            selected_step = {
                'step_id': selected_action['action_id'],
                'preconditions': selected_action['preconditions'],
                'posteffects': selected_action['posteffects'],
                'time': selected_action['time']
            }
            plan_problem['steps'].append(selected_step)

        # Update G
        goals.remove(current_goal)
        new_goals = []
        for precondition in selected_step['preconditions']:
            new_goal = {
                'c': precondition,
                'S': selected_step
            }
            goals.append(new_goal)
            new_goals.append(new_goal)

        # Update O
        new_order = {
            'predecessor': selected_step,
            'successor': current_goal['S']
        }
        plan_problem['ordering_constraints'].append(new_order)

        # Update L
        new_causal_link = {
            'source': selected_step,
            'c': current_goal_precondition,
            'target': current_goal['S']
        }
        plan_problem['causal_links'].append(new_causal_link)
```

Izvorni kod 4.3: Isječak izvornog koda metode `partial_order_planner` kojim su implementirani treći i četvrti korak algoritma

Po ažuriranju parcijalnog poretka vrši se razrješavanje prijetnji prikazano izvornim kodom 4.4. Identifikacija prijetnji obavljena je iteriranjem po skupu kauzalnih poveznica i provjerom prijete li posljedice trenutnog koraka kauzalnoj poveznici. Ukoliko nije pronađena

niti jedna prijetnja, utoliko se prelazi u sljedeću iteraciju algoritma za planiranje parcijalnog poretka. Ako je nad nekom kauzalnom poveznicom pronađena prijetnja prvo se pokušava razriješiti degradacijom. Prijetnju je moguće razriješiti degradacijom ako preduvjeti trenutnog koraka ne ovise o posljedicama izvorišnog koraka kauzalne posljedice, u suprotnom se problem rješava promocijom. Međutim, ako odredišni korak kauzalne posljedice prijeti preduvjetima trenutnog koraka, navedenu prijetnju nije moguće razriješiti. Ako prijetnju nije moguće uspješno razriješiti, poništava se efekt ažuriranja problema parcijalnog poretka te se bira neka od preostalih ispravnih akcija za ispunjenje trenutnog cilja. Ukoliko sve ispravne akcije rezultiraju nerješivim prijetnjama, utoliko se plan proglašava neuspješnim i vraća se rezultat. U suprotnom, nastavlja se s izvođenjem algoritma kako bi se zadovoljili preostali preduvjeti.

```
        # 4. causal link protection
        selected_step_posteffects = selected_step['posteffects']
        all_causal_links_protected = True
        for causal_link in causal_links:
            condition = causal_link['c']
            source = causal_link['source']
            is_threat = check_if_threat(condition, selected_step_posteffects)
            if is_threat:
                # case 1: DEMOTION -> is threat, but does not depend on source
                dependencies = conditions_intersection(source['posteffects'], selected_step
    ['preconditions'])
                if len(dependencies) == 0:
                    protection_order = {
                        'predecessor': selected_step,
                        'successor': causal_link['target']
                    }
                    plan_problem['ordering_constraints'].append(protection_order)
                    continue

                # case 2: PROMOTION -> is threat, depends on source, target does not
    threaten it
                causal_link_target = causal_link['target']

                constraint_threat = {
                    'name': condition['name'],
                    'value': not condition['value']
                }

                target_threats_step = any(effect['name'] == constraint_threat['name']
                                          and effect['value'] == constraint_threat['value']
    for effect in
                                          causal_link_target['posteffects'])

                if not target_threats_step:
                    protection_order = {
                        'predecessor': causal_link['target'],
                        'successor': selected_step
                    }
                    plan_problem['ordering_constraints'].append(protection_order)
                    continue
```

```
38
39                  # case 3: IMPOSSIBLE PLAN
40                  all_causal_links_protected = False
41                  break
42
43          if all_causal_links_protected:
44              valid_action_found = True
45          else:
46              valid_actions.remove(selected_action)
47              plan_problem['steps'].remove(selected_step)
48              goals.append(current_goal)
49              for goal in new_goals:
50                  goals.remove(goal)
51              plan_problem['ordering_constraints'].remove(new_order)
52              plan_problem['causal_links'].remove(new_causal_link)
53
54      # 5. recursively call PoP
55      return partial_order_planner(plan_problem, goals, actions)
```

Izvorni kod 4.4: Isječak izvornog koda metode `partial_order_planner` kojim je implementirano razrješavanje prijetnji

5. Odabir potpunog poretka i generiranje scenarija kibernetičke vježbe

Kako bi se na temelju parcijalnog poretka izradio scenarij kibernetičke vježbe potrebno je odabrati bilo koji potpuni poredak sadržan u parcijalnom poretku. Potpuni poredak jest poredak u kojem su uređene sve veze između koraka plana.

5.1. Odabir potpunog poretka

Metoda za odabir potpunog poretka, `construct_total_order`, prikazana je izvornim kodom 5.1. Ulazni parametar metode je generirani parcijalni poredak. Odabir potpunog poretka izveden je iteriranjem po skupu koraka parcijalnog poretka pri čemu se prvi korak samo dodaje u listu potpunog poretka, a za svaki sljedeći korak odabire se odgovarajuća pozicija u listi potpunog poretka. Odabir odgovarajuće pozicije u listi potpunog poretka započinje rekurzivnim dohvatom svih prethodnika i sljedbenika trenutnog koraka. Potom se pronalazi prethodnik koji se pojavljuje najkasnije u listi potpunog poretka i njegov indeks označava najmanji indeks na kojem može biti smješten trenutni korak. Analogno odabiru najmanjeg indeksa, odabire se najranije pojavljivanje sljedbenika koraka te se njegov indeks uzima kao najveći mogući indeks trenutnog koraka. Ukoliko je najmanji indeks veći od najvećeg indeksa, utoliko je došlo do pogreške i nije moguće odabrati potpuni poredak. Ako su odabrani indeksi ispravni, slučajno se bira pozicija u listi između najmanjeg i najvećeg indeksa te se tamo smješta trenutni korak. Po završetku iteriranja po koracima gotov je odabir potpunog poretka te ga metoda vraća kao rezultat.

Kada je odabran potpuni poredak poziva se metoda `construct_final_order` koja priprema strukturu podataka prikladnu za generiranje elemenata gantograma. Izlaz metode `construct_final_order` je lista parova koraka čime su predstavljene potrebne poveznice za prikaz generiranog potpunog poretka.

```
1 def construct_total_order(partial_order):
2     total_order = []
3     i = 0
4     for step in partial_order['steps']:
5         if len(total_order) < 2:
```

```
            total_order.append(step['step_id'])
            continue
        if step['step_id'] in total_order:
            continue
        predecessors = get_recursive_predecessors(partial_order['ordering_constraints'], step['step_id'])
        successors = get_recursive_successors(partial_order['ordering_constraints'], step['step_id'])
        min_i = max_predecessor(total_order, predecessors)
        max_i = min_successor(total_order, successors)
        if min_i > max_i:
            print("There has been an error!", flush=True)
            return []
        index = 0
        if min_i == max_i or min_i+1==max_i:
            index = max_i
        else:
            index = random.randint(min_i + 1, max_i)
        total_order.insert(index, step['step_id'])

    return total_order
```

Izvorni kod 5.1: Izvorni kod metode `construct_total_order`

5.2. Raspoređivanje koraka u vremenskoj domeni

Za potpuno određen scenarij kibernetičke vježbe nužno je definirati početak i kraj izvođenja svakog koraka potpunog poretka. Konstruiranje gantograma na temelju potpunog poretka vrši se u metodi `construct_gantt_total_order_plan`, a isječkom metode prikazanim izvornim kodom 5.2 izvodi se raspoređivanje koraka u vremenu. Raspoređivanje koraka u vremenu započinje pronalaskom vremena završetka prethodnog koraka i vremenom početka ciljnog koraka. Potom se računa udio vremena potrebnog za izvođenje trenutnog koraka u vremenu potrebnom za izvođenje svih preostalih koraka te se na temelju tog udjela računa trenutak u kojem trenutni korak najkasnije smije završiti. Potom se u vremenskom intervalu između završetka prethodnog koraka i odabranog trenutka umanjenog za trajanje trenutnog koraka odabire trenutak početka izvođenja trenutnog koraka.

```
        step_duration = step['time']
        step_duration_proportion = step_duration/total_steps_duration
        total_steps_duration -= step_duration

        start_limit = int(plan_duration_minutes*step_duration_proportion) - step_duration
        start_minutes = random.randint(0, start_limit)

        start_date = start + timedelta(seconds=start_minutes*60)
        end_date = start_date + timedelta(seconds=step_duration*60)

        start = end_date
        plan_duration_minutes = (end - start).total_seconds() / 60.0
```

Izvorni kod 5.2: Isječak izvornog koda metode `construct_gantt_total_order_plan` za raspoređivanje zadataka u vremenu

Ako se dogodi da je vrijeme potrebno za izvođenje svih koraka plana veće od predviđenog vremena trajanja scenarija, planiranje scenarija smatra se neuspješnim te se obavještava korisnika o neuspjehu. Međutim, navedena situacija ne znači nužno da ne postoji scenarij koji odgovara ulaznim parametrima, već da ne postoji ispravan scenarij temeljen na generiranom parcijalnom poretku.

5.3. Generiranje zadataka za prikaz scenarija pomoću gantograma

Kada su koraci scenarija uspješno raspoređeni u vremenu pripremaju se zadaci i poveznice za prikaz gantograma. Zadaci se pripremaju iteriranjem po listi potpunog poretka, a stvaranje objekta i spremanje zadatka u bazu podataka prikazano je izvornim kodom 5.3. Identifikacijski broj zadatka određuje se sekvencijalno, a atributi koji označavaju ime, identifikacijski broj akcije, preduvjete, posljedice i trajanje određuju se na temelju akcije koja se izvodi trenutnim korakom. Vrijeme početka izvođenja i vrijeme završetka izvođenja određeni su prilikom raspoređivanja koraka u vremenskoj domeni. Identifikacijski broj zadatka roditelja uzima se s početnog zadatka jer pripadaju istom projektu, a atributi koji se odnose na indikaciju uspješnog izvođenja zadatka i boju zadatka u gantogramu postavljaju se na pretpostavljene vrijednosti.

```
        task = {
            'taskid': get_next_sequence("taskId"),
            'text': step_action['name'],
            'start_date': str(start_date),
            'end_date': str(end_date),
            'action': str(int(step_action['action_id'])),
            'progress': 0.0,
            'parent': initial_task["parent"],
            'duration': step['time'],
            'preconditions': step_action["preconditions"],
            'effects': step_action["posteffect"],
            'failed': False,
            'color': 'rgb(61,185,211)',
            'fail_handled': False
        }
        db.tasks.insert(task)
```

Izvorni kod 5.3: Isječak izvornog koda metode `construct_gantt_total_order_plan` za spremanje zadatka u bazu podataka

Kako bi gantogram bio potpun, dodaju se i poveznice između spremljenih zadataka. Poveznice se generiraju na temelju strukture stvorene izvođenjem metode `construct_final_order`, što je prikazano izvornim kodom 5.4.

```
    for order in final_order:
        action_from = order['predecessor']
        action_to = order['successor']
        task_from = db.tasks.find_one({'action': str(int(action_from))})
        task_to = db.tasks.find_one({'action': str(int(action_to))})
        link = {
            'link_id': get_next_sequence('linkId'),
            'source': int(task_from['taskid']),
            'target': int(task_to['taskid']),
            'type': "0"
        }
        db.links.insert(link)
```

Izvorni kod 5.4: Isječak izvornog koda metode `construct_gantt_total_order_plan` za spremanje poveznica u bazu podataka

6. Ponovni izračun plana u slučaju neuspješnog izvođenja koraka plana

6.1. Podrška za rekalkulaciju plana na klijentu

Korisnik označava neuspješno izvođenje koraka plana uređivanjem pripadajućeg zadatka tako da postavlja indikator neuspješnog izvođenja. Kako bi navedena promjena okinula ponovni dohvat podataka s poslužitelja definirana je funkcija `taskUpdateHandler` koja se poziva kada se dogodi događaj (engl. *event*) `onAfterTaskUpdate`. Funkcija prvo čeka dvjesto milisekundi, koliko je dovoljno da na poslužitelj stigne prethodni zahtjev za uređivanje zadatka. Nakon dvjesto milisekundi postavlja vrijednost stanja (engl. *state*) komponente `planRecalculated` na istinu. Modifikacija stanja okida poziv metode životnog ciklusa komponente `shouldComponentUpdate` koja određuje da se komponenta mora ažurirati u slučaju da je stanje `planRecalculated` istinito. Po završetku ažuriranja komponente poziva se metoda životnog ciklusa `componentDidUpdate` koja postavlja stanje `planRecalculated` na neistinu, briše prikaz gantograma te šalje zahtjev za dohvat podataka poslužitelju, što je prikazano isječkom izvornog koda 6.1. U trenutku kada je poslan zahtjev za dohvat podataka potencijalno još traje rekalkulacija plana, stoga su API metode poslužitelja proširene semaforom i zahtjev za dohvat podataka čeka da bude do kraja obrađen zahtjev za uređivanjem podataka. Jednom kada je novi plan konstriuran, klijent uspješno dohvaća nove podatke i prikazuje ih na gantogramu.

```
1            if (this.state.planRecalculated){
2                this.setState({
3                    planRecalculated: false
4                });
5            }
6
7            gantt.init(this.ganttContainer);
8            gantt.clearAll();
```

```
9                gantt.load("http://localhost:8080/gantt");
```

Izvorni kod 6.1: Isječak izvornog koda metode `componentDidUpdate` za dohvat rekalkuliranog plana

6.2. Podrška za rekalkulaciju plana na poslužitelju

Rekalkulacija plana na poslužitelju izvodi se ako je u zadatku koji se nalazi u tijelu zahtjeva za uređivanjem atribut `step_failed` označen kao istinit. Obrada takvog zahtjeva započinje promjenom boje zadatka iz plave u crvenu i postavljanjem zastavice `task_failed` na istinitu vrijednost. Potom se ažurira zadatak u bazi podataka i poziva metoda `recalculate_plan` s vrijednosti ulaznog parametra postavljenom na identifikacijski broj uređenog zadatka. Time se trenutni, neuspješno izvedeni zadatak proglašava početnim stanjem te se ignoriraju svi prethodni, već izvršeni zadaci. Rekalkulacija odabranog dijela plana započinje brisanjem zadataka koji se nalaze nakon trenutnog zadatka, a prije ciljnog zadatka. Potom se izvodi algoritam planiranja parcijalnog poretka i konstrukcija potpunog poretka novih koraka za prikaz gantogramom.

7. Zaključak

Planiranje i provođenje kibernetičkih vježbi nužno je kako bi se osigurala spremnost ključnih institucija za ispravnu i pravodobnu reakciju na kibernetičke krize. Planiranje scenarija kibernetičke vježbe složen je i dugotrajan proces koji zahtjeva opširno domensko znanje o kibernetičkoj sigurnosti. Automatizacija planiranja scenarija kibernetičke vježbe ubrzava proces planiranja vježbe te omogućava bolji uvid u moguće konkuretne scenarije za ostvarenje zadanog, istog cilja. Također, omogućava i brzu rekalkulaciju plana u slučaju da za tim pojavi potreba uslijed interakcije branitelja.

U ovom je radu opisana programska podrška za planiranje i provođenje kibernetičkih vježbi koja za planiranje scenarija kibernetičke vježbe koristi algoritam planiranja parcijalnog poretka, a za prikaz scenarija na korisničkom sučelju i za praćenje provođenja scenarija koristi alat za upravljanje projektima, gantogram. U prvom je dijelu rada objašnjeno što su kibernetičke vježbe, kako se provode te koja je motivacija za njihovo provođenje. Potom su opisani tehnički alati korišteni za izradu programskog rješenja i navedeni funkcionalni zahtjevi za programsko rješenje. U radu je dan i primjer korištenja programske podrške za planiranje i provođenje kibernetičke vježbe te je opisan algoritam korišten za implementaciju rješenja.

Razvijeno rješenje nužno pronalazi scenarij koji odgovara ulaznim parametrima ako on postoji, međutim ako isti nije izvediv u zadanom vremenskom intervalu ne prikazuje ga krajnjem korisniku. Ako je korisnik siguran da postoji valjani scenarij za zadane ulazne parametre, može okinuti ponovni izračun plana jer je odabir akcija za izvođenje slučajan i postoji vjerojatnost da će ponovni izračun rezultirati valjanim scenarijem.

LITERATURA

Daniel S. Weld Anthony Barrett. *Partial-Order Planning: Evaluating Possible Efficiency Gains*. University of Washingon, 1993. URL https://homes.cs.washington.edu/~weld/papers/tr92-05-01a.pdf.

J. Scott Penberthy Daniel S. Weld Anthony Barrett, Keith Golden. *UCPOP User's Manual*. 1994.

Council Conclusions on Cyber Diplomacy. Council of the European Union, 2015. URL http://data.consilium.europa.eu/doc/document/ST-6122-2015-INIT/en/pdf.

DHTMLX Gantt Documentation. DHTMLX, 2020. URL https://docs.dhtmlx.com/gantt/.

Cyber Europe Programme. European Union Agency for Cybersecurity, 2020. URL https://www.enisa.europa.eu/topics/cyber-exercises/cyber-europe-programme.

React JavaScript Documentation. Facebook Inc, 2020. URL https://reactjs.org/docs/getting-started.html.

Daniel S. Weld J. Scott Penberthy. *UCPOP: A Sound, Complete, Partial Order Planner for ADL*. IBM T.J. Watson Research Center, University of Washington, 1992. URL https://www.act.nato.int/articles/exercise-cyber-coalition-2019-concludes-estonia.

Jason Kick. *Cyber Exercise Playbook*. The MITRE Corporation, 2014. URL https://www.mitre.org/sites/default/files/publications/pr_14-3929-cyber-exercise-playbook.pdf.

Paolo Traverso Malik Ghallab, Dana Nau. *Automated Planning Theory and Practice*. Morgan Kaufmann Publishers, 2004.

Kibernetički štit 2018. Ministarstvo obrane Republike Hrvatske, 2018. URL https://www.morh.hr/o-vjezbi/.

Exercise Cyber Coalition 2019 Concludes in Estonia. NATO, 2019. URL https://www.act.nato.int/articles/exercise-cyber-coalition-2019-concludes-estonia.

Cyber Scenario Planning Handbook. Singapore Exchange, 2019. URL https://api2.sgx.com/sites/default/files/2019-07/SGX\%20RegCo\%20Cyber\%20Scenario\%20Planning\%20Handbook_0.pdf.

A guide to cyber exercises. Victoria State Government, 2019. URL https://www.vic.gov.au/sites/default/files/2019-08/Vic-Gov-Cyber-Exercise-guide.pdf.

Daniel S. Weld. *An Introduction to Least Commitment Planning*. University of Washingon, 1994. URL https://homes.cs.washington.edu/~weld/papers/pi.pdf.

Programska podrška za planiranje i provođenje kibernetičkih napada

Sažetak

Ovaj diplomski rad bavi se razvojem programske podrške za planiranje i provođenje kibernetičkih vježbi. U radu su opisani procesi planiranja i provođenja kibernetičkih vježbi te je dan pregled funkcionalnih zahtjeva za programsku podršku za automatizaciju planiranja scenarija kibernetičkih vježbi. Također, opisani su i tehnički detalji implementacije programske podrške te je dan primjer korištenja. Za automatizaciju planiranja scenarija kibernetičke vježbe korišten je algoritam planiranja parcijalnog poretka te je u radu objašnjeno kako je prilagođen domeni planiranja kibernetičkih vježbi.

Ključne riječi: Kibernetičke vježbe, automatizirano planiranje, algoritam planiranja parcijalnog poretka, programska podrška, gantogram

Software support for planning and execution of cyber attacks

Abstract

This thesis is about development of a software for automated planning and execution of cyber exercises. It contains a description of cyber exercise planning and execution processes and it demonstrates functional requests for given software. Also, it contains a description of implementation technical details and a demonstration example. Cyber exercise scenario planning was automated using partial order planning algorithm so the thesis contains an explanation of how it was adapted to cyber exercise planning domain.

Keywords: Cyber exercises, automated planning, partial order planning algorithm, software, gantogram