SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3051

NAPADI NA PSD2 API

Joško Šestan

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 3051

NAPADI NA PSD2 API

Joško Šestan

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 3051

Pristupnik:	**Joško Šestan (0036504353)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	doc. dr. sc. Stjepan Groš

Zadatak: **Napadi na PSD2 API**

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. U sklopu diplomskog rada potrebno je definirati scenarije napada koji se mogu provesti putem izloženog API-ja. Za svaki scenario potrebno je opisati štete koje mogu nastati te razviti zaštitne metode koje se mogu koristiti. Korištenjem nekakvog programskog alata implementirati napade. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 27. lipnja 2022.

SADRŽAJ

1. Uvod

Porastom popularnosti i razvoja internet bankarstva, financijska i bankovna industrija prate trend te razvijaju nove i unapređuju već postojeće usluge koje pružaju. Pojavom PSD2 direktive, Europska Unija ubrzava modernizaciju i unapređenje usluga koje banke pružaju. PSD2 pruža sigurnije, jednostavnije i učinkovitije obavljanje plaćanja te dohvaćanja resursa u bankama. Zbog nove direktive, banke moraju otvoriti svoje API-je kako bi treće strane, kao pružatelji bankovnih usluga, mogli pristupiti resursima korisnika u banci. Dok će ovo imati mnoge pozitivne strane, kao postojanje novih usluga i jednostavnost obavljanja plaćanja te pristupa resursima u banci, imati će i neke negativne.

Iako je zaštita korisnika u prvom planu PSD2 direktive, otvaranje API-ja trećima stranama samim time povećanje pružatelja usluga koji će moći pristupiti računima korisnika u banci povećati će i količinu mogućih napada te otvoriti novi vektore napada. Financijska je industrija zanimljiva napadačima zbog podataka i resursa kojima mogu pristupiti. Potrebno je poduzeti određene mjere zaštite kako bi se spriječili mogući napadi i prijetnje. Fokus ovog rada biti će upravo napadi na API-je banaka te načini kako se od njih zaštititi, kako bi podaci korisnika ostali sigurni.

Rad je strukturiran na sljedeći način. U 2. poglavlju predstavljen je uvid u PSD2 direktivu. Na čemu je naglasak direktive, tko su sudionici te na koji način funkcionira. Isto tako predstavljen je tok OAuth2 protokola koji je jedan od fokusa ovog rada. 3. poglavlje sadrži opisane scenarija napada koji prijete API-ima banaka. Napadi su podijeljeni u dvije kategorije. Jedan dio napada vezan je uz OAuth2 protokol i njegove parametre, dok su drugi napadi vezani direktno za API. U 4. poglavlju, pomoću programskog alata, prikazana je implementacija napada. Poglavlje 5. sadrži cjelokupan zaključak rada.

2. *Revisited Payment Services Directive*

Revisited Payment Services Directive (PSD2) je direktiva Europske Unije administrirana od strane Europske Komisije u svrhu reguliranja platnih usluga na području Europske Unije. Glavni naglasak PSD2 direktive je na modernizaciji i poboljšanju izvođenja platnih usluga u platnoj industriji te povećanje sigurnosti obavljanja platnih usluga i pouzdana autentifikacija klijenta (*Strong Customer Authentication*, SCA). Kod pouzdane autentifikacije klijenta misli se na autentifikaciju korisnika koji obavlja platne usluge u banci. Korisnici se u kontekstu PSD2 nazivaju *Payment Service Users* (PSU)

PSD2 omogućuje *Account Servicing Payment Service Providers* (ASPSP) da dopuste *Third Party Providers* (TPP) obavljanje platnih usluga za korisnike. ASPSP je zapravo banka, te će u ostatku teksta biti tako adresirano. TPP se još naziva i *Payment Service Provider* (PSP) pošto oni pružaju platne usluge. Postoje dvije vrste TPP-a: *Payment Initiation Service Providers* (PISP) koji će moći provoditi plaćanja za korisnika i *Account Information Service Providers* (AISP) koji će moći pristupiti resursima korisnika u banci te prikazati informacije vezane uz račun korisnika. Banka će morati TPP-ima pružiti Aplikacijska programska sučelja (*Application Programming Interface*, API) kako bi mogli obavljati navedene usluge. API omogućuje komuniciranje servisa TPP-a sa servisima aplikacije banke. Na primjer, API će omogućiti da TPP obavi plaćanje preko aplikacije banke ili da pristupi resursima iz baze podataka banke.

TPP su tvrtke koje se bave financijskim tehnologijama (*fintech*). Kako bi TPP mogao pružati usluge korisnicima, PSD2 zahtjeva da TPP sadrži određenu identifikaciju u obliku kvalificiranih certifikata (*qualified certificates*) [4, 11, 6]. Kvalificirani certifikati služe za osiguranje sigurnih transakcija u Europskoj uniji pod regulacijom *electronic IDentification, Authentication and trust Services* (eIDAS). Kvalificirane certifikate regulirane od eIDAS izdaju *Qualified Trust Service Providers* (QTSP). Europska Unija predstavlja i održava listu QTSP-a [11, 6]. Kod PSD2 regulacije postoje dva tipa kvalificiranih certifikata: *Qualified Website Authentication Certificate* (QWAC) i

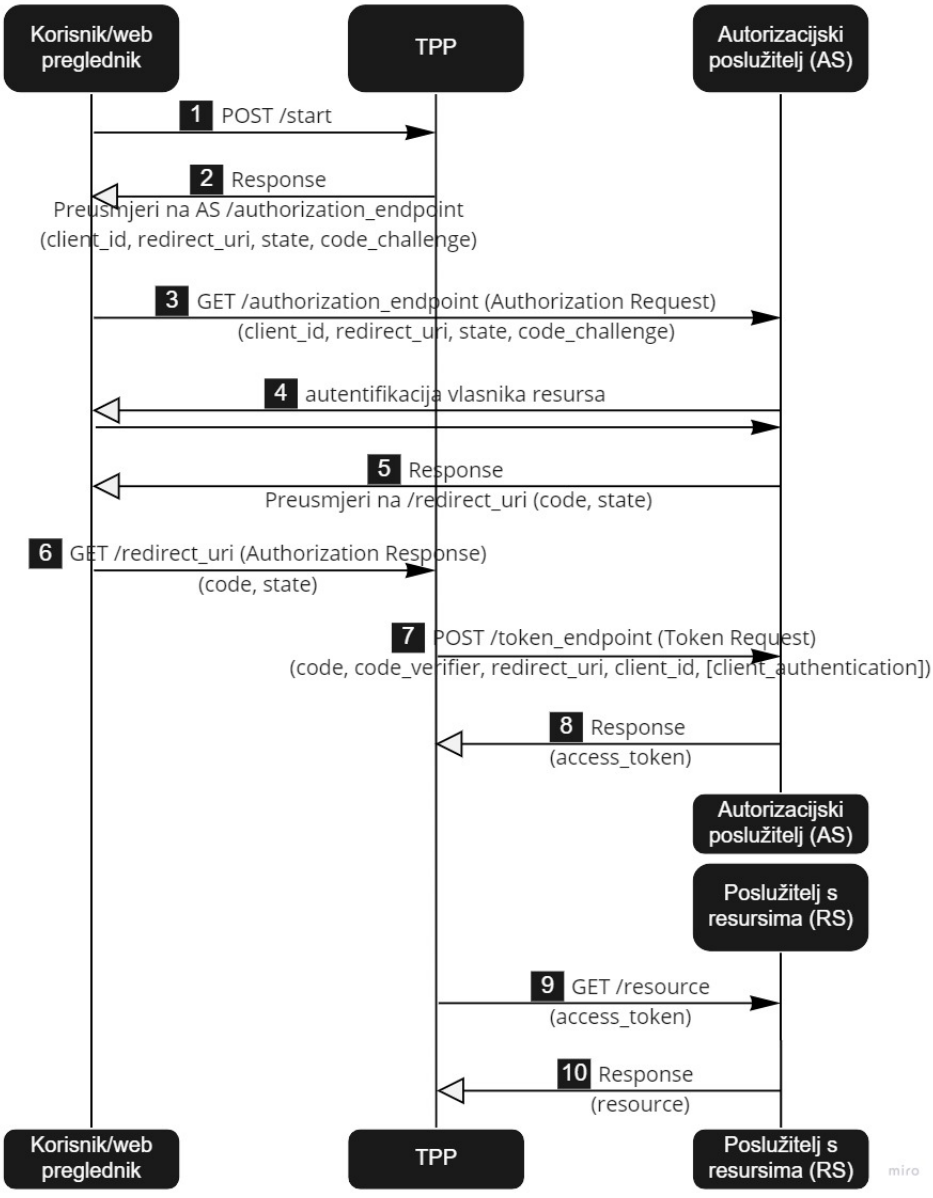
Qualified Certificate for Electronic Seal (QSealC) [4, 11, 6]. Obje vrste certifikata koriste se za autentifikaciju TPP-a kod banke. QWAC se koristi za zaštitu podataka u prijenosu te omogućuje uspostavljanje *Transport Layet Security* (TLS) konekcije, dok QSealC identificira od kuda su podaci došli, odnosno identificira TPP-a. Više od TLS-u i TLS konekciji u poglavlju 2.1. Postoji mnogo pristupa prema SCA [11]. Jedan od pristupa je korištenje *Open Authorization* (OAuth) protokola, preciznije OAuth2 protokola. OAuth2 kao sredstvo autentifikacije korisnika definira i kako će teći cijeli tok kod obavljanja platnih usluga korisnika od strane TPP-a. Objašnjenje OAuth2 toka nalazi se u poglavlju 2.1.

2.1. OAuth2 tok u PSD2

OAuth2 protokol koristi se kao način autentifikacije korisnika od strane banke i time korisnik dopušta TPP-u da pristupi njegovim resursima. OAuth2 dopušta TPP-ima pristup podacima na način da korisnik ne dijeli svoje vjerodajnice s TPP-om nego se autentificira direktno kod banke. Detaljna objašnjenja OAuth2 toka nalaze se u [8, 12, 11].

U OAuth2 toku prisutni su: korisnik koji želi obaviti određene usluge u banci, TPP koji će pristupit resursima u banci te za korisnika obaviti usluge i banka koju čine dva poslužitelja kojima se pristupa preko API-ja. Dva poslužitelja banke su autorizacijski poslužitelj, na kojemu se vrši autentifikacija korisnika te autoriziranje TPP-a kako bi mogao pristupiti resursima korisnika u svrhu obavljanja usluge, i poslužitelj s resursima kojem pristupa prethodno ovlašteni TPP u svrhu dobavljanja resursa. OAuth2 tok prikazan je na slici 2.1. te on funkcionira na sljedeći način.

Korisnik želi pristupiti svojim resursima u banci, odnosno obaviti neku uslugu. Korisnik će preko svojega web preglednika doći na web stranicu TPP-a, odabrati će banku kojoj želi pristupiti te će na taj način započeti tok (korak 1 na slici 2.1). Poslat će se zahtjev *POST /start* prema TPP-u. Nakon što TPP zaprimi zahtjev šalje odgovor nazad prema korisniku koji će ga preusmjeriti na autorizacijski poslužitelj banke, odnosno na krajnju točku autorizacije, kako bi se korisnik mogao autentificirati (korak 2 na slici 2.1). U odgovoru se nalaze parametri *client_id*, *redirect_uri*, *scope*, *state* i *code_challenge*. U ovom slučaju klijent banke je TPP, pa *client_id* sadrži identifikator TPP-a koji nije tajan ali je jedinstven kod autorizacijskog poslužitelja banke. Parametar *redirect_uri* sadržavati će URI TPP-a na koji će autorizacijski poslužitelj banke



Slika 2.1: OAuth2 tok

preusmjeriti korisnika nakon autentifikacije. Nadalje, *scope* definira kojim resursima i u koju svrhu TPP smije pristupiti. Za obranu od CSRF napada, objašnjenog u poglavlju 3.1.5, koristi se parametar *state* koji je jednokratna vrijednost definirana od strane TPP jedinstvena za sjednicu s korisnikom. Parametar *code_challenge* vezan je uz *Proof Key for Code Exchange* (PKCE). PKCE je proširenje OAuth2 protokola koje služi kao dodatna zaštita od napada, s naglaskom na zaštitu od korištenja ukradenog autorizacijskog koda (*authorization code*). TPP kreira nasumičnu vrijednost koja se zove *code_verifier* te sažima tu vrijednost u *code_challenge*. Autorizacijski će poslužitelj vezati autorizacijski kod uz *code_challenge*.

Preusmjeravanje korisnika će se izvršiti slanjem autorizacijskog zahtjeva (*Authorization request*) *GET /authorization_endpoint* s istim parametrima koji su bili prisutni u odgovoru TPP-a (korak 3 na slici 2.1). Autorizacijski poslužitelj banke sada autentificira korisnika, npr. tako da korisnik unese korisničko ime i lozinku (korak 4 na slici 2.1). Nakon uspješne autentifikacije korisnika autorizacijski će poslužitelj banke poslati odgovor (korak 5 na slici 2.1) koji će, uz parametar *state*, sadržavati parametar *code* u koji je uključen kreirani autorizacijski kod te će preusmjeriti korisnika nazad na TPP-a.

TPP će kod banke već imati definirane URI-je za preusmjeravanje korisnika. Parametrom *redirect_uri* TPP će definirati koji URI će banka onda koristiti za preusmjeravanje. Preusmjeravanje nazad na TPP vrši se slanjem zahtjeva *GET /redirect_uri (Authorization Response)* u koji će onda biti uključen autorizacijski kod (korak 6 na slici 2.1). TPP će onda slati zahtjev *POST /token_endpoint (Token Request)* na krajnju točku za dohvaćanje pristupnog tokena (*access token*) (korak 7 na slici 2.1). U zahtjev će uključiti autorizacijski kod koji će moći razmijeniti za pristupni token nakon što se autentificira na autorizacijskom poslužitelju banke. No kako bi mogao razmijeniti autorizacijski kod za pristupni token, TPP u zahtjev mora uključiti *code_verifier* pošto je autorizacijski kod vezan uz *code_challenge*. Isto tako, autorizacijski će poslužitelj banke provjeravati parametar *redirect_uri* koji će sadržavati URI TPP-a na koji je autorizacijski poslužitelj preusmjerio korisnika nakon autentifikacije.

Svaka komunikacija između TPP-a i poslužitelja banke mora biti zaštićena TLS protokolom. TLS je kriptografski protokol koji osigurava sigurnost komunikacije na računalnoj mreži. TLS se postavlja od strane TPP-a. Uz uspostavljanje TLS veze TPP se autentificira poslužitelju banke korištenjem kvalificiranih certifikata izdanih od eIDAS. Kod OAuth2 protokola TPP se autentificira korištenjem *OAuth 2.0 Mutual TLS for Client Authentication and Certificate Bound Access Tokens* (mTLS) u spoju s kvalifi-

ciranim certifikatima [3]. Kod mTLS-a TPP će se spojiti na poslužitelj koji će TPP-u predstaviti svoj TLS certifikat. Nakon što TPP verificira certifikat od poslužitelja, TPP predstavlja svoj TLS certifikat koji poslužitelj mora verificirati. Nakon verifikacije, TPP-u je dopušten pristup autorizacijskom poslužitelju banke. Time se TPP autentificirao kod poslužitelja te će mu poslužitelj zamijeniti autorizacijski kod za pristupni token. *Token binding* je povezivanje pristupnog tokena na TLS certifikat TPP-a tako da ga samo on može iskoristiti na poslužitelju s resursima banke, u svrhu dohvaćanja resursa i konačno obvaljanja usluge.

Nakon uspješne autentifikacije, autorizacijski poslužitelj banke odgovara TPP-u te u odgovoru šalje pristupni token (korak 8 na slici 2.1). Pristupni će token biti povezan na *scope* kako bi TPP mogao pristupiti samo onome što je korisnik zatražio. Nakon dohvaćanja pristupnog tokena TPP šalje zahtjev *GET /resource*, prema poslužitelju s resursima banke, u koji uključuje pristupni token. Poslužitelj s resursima sada provjerava je li taj pristupni token vezan upravo za tog TPP-a. Ako je, TPP pristupa resursima korisnika (korak 9 na slici 2.1). Poslužitelj s resursima odgovara TPP-u te u odgovoru uključuje resurse koje je TPP za tražio. Na taj će način korisnik preko TPP-a dobiti što je i zatražio te tok tu završava.

3. Napadi

Dolaskom PSD2 direktive banke dopuštaju TPP-ima pristup API-ima banaka u svrhu obavljanja platnih usluga za korisnike. Na taj se način otvaraju novi napadački vektori [25]. U sljedećim potpoglavljima predstavljeni su napadi, odnosno scenariji napada, te načine kako se od njih može zaštititi. Potpoglavlje 3.1 biti će fokusirano na napade vezane uz iskorištavanje propusta vezanih uz OAuth2 protokol, a potpoglavlje 3.2 na napade vezane uz propuste na samom API-ju.

3.1. Napadi vezani uz OAuth2

U ovom poglavlju biti će navedeni napadi i prijetnje koje bi OAuth2 protokol trebao spriječiti. Naime, lošom implementacijom ili nedostatkom određenih parametara i mehanizama zaštite kod OAuth2 protokola, napadač će biti u mogućnosti pristupiti API-ju banke te samim time i resursima korisnika.

3.1.1. Nedovoljna validacija URI-ja za preusmjeravanje

Parametar *redirect_uri* prvi se puta pojavljuje u OAuth2 toku kada TPP preusmjerava korisnika na krajnju točku za autorizaciju (korak 2 na slici 2.1). Autorizacijski poslužitelj banke mora validirati URI za preusmjeravanje, odnosno parametar *redirect_uri*, kako bi bi bio siguran da se radi o jednom od URI-ja prethodno registriranih kod njega od strane TPP-a. Kod registracije URI-ja za preusmjeravanje, neki će autorizacijski poslužitelji dopuštati TPP-ima registraciju obrazaca URI-ja a ne kompletne URI-je. Što znači da će autorizacijski poslužitelj onda validirati, odnosno uspoređivati, parametar *redirect_uri* s registriranim obrascima. Prilikom korištenja obrazaca javlja se prijetnja od curenja autorizacijskog koda ili pristupnog tokena.

Napadač će moći doći do autorizacijskog koda ili pristupnog tokena na način da po-

šalje korisnika na URI koji se nalazi pod kontrolom napadača. Za objašnjenje scenarija ovog napada uvedene su neke pretpostavke [19]. Pretpostavka je da obrazac URL-a, odnosno lokacija na koju se preusmjerava korisnik, izgleda nešto kao *https://*.somesite.example/** te je on registriran od strane TPP-a s nekim *client_id*-om. Svrha takvog obrasca je da omogući kreiranje valjanih URI-ja za preusmjeravanje s poddomenom *somesite.example*, npr. *https://app.somesite.example/redirect*. Naime, znak * bi se na autorizacijskom poslužitelju mogao interpretirati na krivi način. Mogao bi se interpretirati kao bilo što, a ne nešto što bi bilo valjano za određenu domenu. Napad bi krenuo tako što bi korisnik, nakon što pokrene tok dohvaćanja resursa ili informacija o svome računu u banci, poslije koraka gdje ga TPP želi preusmjeriti na krajnju točku autorizacije (korak 2 na slici 2.1), otvorio web stranicu koja je pod kontrolom napadača na način da ga je napadač na to natjerao ili mu podvalio. Otvaranjem te stranice korisnik će biti preusmjeren na konačnu točku autorizacije. U zahtjevu koji je poslan prema konačnoj točki autorizacije nalazit će se *client_id* od legitimnog TPP-a, koji je došao u posjed napadača kada je korisnik otvorio njegovu stranicu, ali i *redirect_uri* koji je napadač promijenio. Pošto se koriste obrasci, napadač je s lakoćom kreirao svoj URI za preusmjeravanje te će on, usporedbom obrazaca, na autorizacijskom poslužitelju banke biti prihvaćen. U odgovoru autorizacijskog poslužitelja prema korisniku šalje se autorizacijski kod te se korisnika preusmjerava na zadani URI za preusmjeravanje. Ako korisnik nije vidio URI za preusmjeravanje velika je mogućnost da neće ni biti svjestan napada [19]. Ovim preusmjeravanjem napadač će doći posjed autorizacijskog koda te postoji mogućnost da ga zamijeni za pristupni token korisnika.

Najbolja zaštita od ovog tipa napada je korištenje potpunih URI-ja za preusmjeravanje kod registracije kod autorizacijskog poslužitelja gdje će ih on onda uspoređivati s isto tako potpunim URI-jima za preusmjeravanje sadržanim u parametru *redirect_uri* [19].

3.1.2. Curenje vjerodajnica putem *Referer* zaglavlja

Referer zaglavlje unutar HTTP zahtjeva sadrži apsolutnu ili parcijalnu adresu od web stranice koja je taj zahtjev napravila. U koracima 3 i 6 u OAuth2 toku (slika 2.1) šalje se zahtjev za autorizaciju prema autorizacijskom poslužitelju banke i zahtjev prema TPP-u kao odgovor autorizacijskog poslužitelja banke. Ti zahtjevi sadrže neke važne parametre kao *code* i *state*. Može se dogoditi da do tih vrijednosti dođe napadač putem *Referer* HTTP zaglavlja nakon što zahtjevi iscure na web stranici TPP-a [19].

Curenje zahtjeva može se dogoditi na nekoliko načina. Moguće je da web stranica TPP-a sadrži poveznice na neke stranice koje su pod kontrolom napadača. Kada bi korisnik kliknuo na neku od tih poveznica napadač bi dobio zahtjev, u kojem je sadržan odgovor banke poslije autentifikacije korisnika. Iz zahtjeva bi onda bio u mogućnosti izvući parametre *code* i *state*. Ako web stranica TPP-a sadrži neke sadržaje trećih strana kao reklame ili slike može se dogoditi ista stvar. Sada napadač može potencijalno izmijeniti autorizacijski kod kako bi dobio pristupni token. Ako napadač zna vrijednost parametra *state* gubi se CSRF zaštita.

U [19] postoji nekoliko rješenja za ovaj problem. Prva i osnovna zaštita bila bi da web stranica TPP-a ne smije sadržavati sadržaje trećih strana ili bilo kakve poveznice na vanjske web stranice. Nadalje, još jedan način zaštite je da se može upotrijebiti *Referrer-Policy* HTTP zaglavlje koje će definirati koliko i šta točno može biti uključeno u zahtjevu. Moguće je još i ograničiti broj korištenja autorizacijskog koda u svrhu dobivanja pristupnog tokena. Ako se korištenje koda ograniči na jednostruko te ako je kod već iskorišten prije nego ga napadač uspije iskoristiti, neće ga ni moći iskoristiti. Isto tako, *state* vrijednost se može poništiti od strane TPP-a nakon preusmjeravanje korisnika s autorizacijskog poslužitelja banke jer se tu događa provjera *state* vrijednosti te nakon toga *state* više nije potreban. Ako se *state* poništi nakon provjere TPP-a, ta se ista vrijednost više neće moći iskoristiti od strane napadača.

3.1.3. *Mix-up* napad

U ovom su napadu prisutna dva autorizacijska poslužitelja te je napad izveden tako da napadač manipulira zahtjevima i određuje koji će autorizacijski poslužitelj, i kada, biti korišten [7]. Postoje određene pretpostavke, odnosno preduvjeti koji moraju biti zadovoljeni kako bi se napad mogao izvršiti. Napadač mora biti u mogućnosti presresti i manipulirati zahtjevima između korisnika i TPP-a, konkretno zahtjevi u koracima 1 i 3 OAuth2 toka na slici 2.1. Nadalje, TPP mora moći koristiti više autorizacijskih poslužitelja. Napadač može koristiti dinamičku registraciju TPP-a kako bi ga registrirao na svom autorizacijskom poslužitelju [19]. I posljednja pretpostavka je to da TPP koristi jednaki URI za preusmjeravanje za svaki autorizacijski poslužitelj. Jedan autorizacijski poslužitelj je u posjedu napadača, dok je drugi autorizacijski poslužitelj pošten, odnosno u posjedu banke.

Korisnik želi dohvatiti neki resurs iz banke te preko TPP-a odabire svoju banku, odnosno pokreće se autentifikacija na autorizacijskom poslužitelju banke (korak 1 na

slika 2.1). Napadač presreće zahtjev koji korisnik šalje prema TPP-u te ga modificira na način da umjesto autorizacijskog poslužitelja banke, definira da se autentikacija korisnika izvrši na autorizacijskom poslužitelju napadača. TPP će odgovoriti korisniku slanjem zahtjeva koji će ga preusmjeriti na krajnju točku za autorizaciju na autorizacijskom poslužitelju napadača (korak 2 na slika 2.1). Napadač također presreće i ovaj zahtjev te radi izmjenu da se korisnik preusmjeruje na krajnju točku autorizacije autorizacijskog poslužitelja banke, a ne napadača. Isto tako, napadač će promijeniti *client_id* TPP-a kod napadačkog autorizacijskog poslužitelja s *client_id* TPP-a kod autorizacijskog poslužitelja banke. *Client_id* TPP-a kod autorizacijskog poslužitelja banke je javno dostupan pa je to moguće učiniti. Korisnik se sada autentificira na autorizacijskom poslužitelju banke te autorizira TPP-a da može pristupiti njegovim resursima, odnosno korisnik se preusmjeruje nazad kod TPP-a te TPP dobiva autorizacijski kod u zahtjevu (korak 6 na slika 2.1). TPP će sada razmijeniti autorizacijski kod kako bi dobio pristupni token. Pošto TPP i dalje misli da je autorizacijski kod izdan od strane autorizacijskog poslužitelja napadača, on će izmjenu autorizacijskog koda za pristupni token vršiti upravo na autorizacijskom poslužitelju napadača. Na taj će način napadač doći u posjed autorizacijskog koda. Autorizacijski se kod može onda izmijeniti za pristupni token ili se može izvršiti napad ubacivanjem autorizacijskog koda koji je opisan u poglavlju 3.1.4.

Različiti načini zaštite navedeni u [19, 7]. Jedan način kako spriječiti ovaj napad bio bi da TPP koristi različiti URI za preusmjeravanje korisnika, na krajnju točku autorizacije, za različite autorizacijske poslužitelje jer kada bi se URI razlikovao napadač ne bi mogao manipulirati zahtjeve tako da mijenja na koji se autorizacijski poslužitelj preusmjeruje korisnik. Naime, neki TPP-i ne mogu imati različite URI-je za preusmjeravanje za različite autorizacijske poslužitelje. U tom slučaju trebao bi se dodati dodatni parametar na URI za preusmjeravanje korisnika nazad kod TPP-a od strane autorizacijskog poslužitelja. Taj bi parametar sadržavao identitet autorizacijskog poslužitelja te bi tada TPP mogao provjeriti da li je preusmjeravanje korisnika, odnosno slanje zahtjeva koji će preusmjeriti korisnika nazad TPP-u, učinjeno od strane autorizacijskog poslužitelj od kojeg je TPP to i očekivao.

3.1.4. Napad ubacivanjem autorizacijskog koda

Pomoću nekih od prethodnih napada, napadač može doći u posjed autorizacijskog koda od nekog korisnika. Autorizacijski se kod tada može iskoristiti u svrhu generiranja

pristupnog tokena za pristup resursima legitimnog korisnika, kojem je ukraden autorizacijski kod, te posljedično i samom pristupu resursima korisnika u banci. U napadu ubacivanjem autorizacijskog koda cilj napadača je iskoristiti autorizacijski kod na način da će ga ubaciti u svoju vlastitu sesiju s nekim legitimnim TPP-om.

Napad započinje tako da napadač započne sjednicu s TPP-om te zatražuje resurse iz banke koja je izdala autorizacijski kod prethodno ukraden od nekog korisnika. OAuth2 tok teče normalno sve do koraka kada se korisnik, odnosno u ovom slučaju napadač, preusmjeruje nazad na TPP-a nakon obavljene autentifikacije kod autorizacijskog poslužitelja banke (korak 6 na slika 2.1). Naime, kako autorizacijski poslužitelj banke šalje autorizacijski odgovor nazad napadaču, on je u mogućnosti taj odgovor izmijeniti korištenjem određenih alata te u njega ubaciti autorizacijski kod kojeg je prethodno ukrao legitimnom korisniku [19]. Preusmjeravanjem napadača nazad na TPP, TPP dobiva zahtjev koji sadrži autorizacijski odgovor, odnosno autorizacijski kod. TPP će tada taj autorizacijski kod zamijeniti za pristupni token što će omogućiti napadaču da pristupi resursima nekog korisnika u banci. Autorizacijski će poslužitelj autentificirati TPP-a, odnosno provjeravati ako je kod izdan TPP-u koji ga pokušava i zamijeniti te ako URI za preusmjeravanje odgovara parametru *redirect_uri*. Pošto napadač sve izvršava uz pomoć legitimnog TPP-a, sve provjere će uspjeti. Drugim riječima TPP ne bi mogao zamijeniti autorizacijski kod za pristupni token u slučaju da napadač koristi autorizacijski kod koji je izdan za nekog drugog TPP-a, odnosno za neki drugi *client_id*, te ako je napadač ukrao autorizacijski kod manipulacijom URI-ja za preusmjeravanje jer se tada taj URI neće poklapati s parametrom *redirect_uri*.

Zaštita od ovog napada proizlazi iz korištenja *PKCE challenge*-a [19]. Iako će TPP imati odgovarajući *code_verifier*, autorizacijski je kod povezan s *code_challenge*-om koji ne odgovara ovom *code_verifier*-u već onom koji je definiran u sjednici gdje je sam kod i u ukraden. Postoji i nemogućnost izmijene autorizacijskog koda za pristupni token ako je autorizacijski kod već iskorišten kada je to zahtijevao legitimni korisnik, a ima ograničen broj korištenja postavljen na jedan.

3.1.5. *Login Cross Site Request Forgery*

U *Login* tipu *Cross Site Request Forgery* (CSRF) napada korisnik će pristupati resursima napadača u banci. Napadač će smjestiti korisniku da se prijavi na njegov vlastiti račun u svrhu dobivanja određenih informacija od korisnika. Na primjer, korisnik bi mogao dodati neke privatne podatke, kao što je broj kartice, direktno na račun na-

padača. Nakon toga napadač se može prijaviti u svoj račun u banci te pristupiti tim podacima [16].

Napad se odvija tako da je korisnik prevaren, odnosno na neki način manipuliran, da pristupi web stranici koja je pod kontrolom napadača. Korisnik će misliti kako je pokrenuo OAuth2 tok za pristupanje svojim resursima. Naime, napadač će preko svoje stranice pokrenuti legitimno dohvaćanje autorizacijskog koda za pristup vlastitim resursima (koraci 1-5 na slika 2.1). Korisnik će biti preusmjeren nazad na web stranicu TPP-a no umjesto svojeg autorizacijskog koda, njegov će zahtjev sadržavati autorizacijski kod napadača. TPP će dalje razmijeniti taj autorizacijski kod napadača za pristupni token. Na taj način će korisnik biti prijavljen u bankovni račun napadača. Posljedično, sve informacije koje korisnik ostavi na računu napadača biti će dostupne samom napadaču te ih on može iskoristiti u vlastitu svrhu.

Rješenja kako se zaštiti od ovog napada navedene su u [8, 19, 18]. Zaštita od ovog napada proizlazi iz *state* parametra. Parametar *state* se prvi put pojavljuje u koraku OAuth2 toka kada TPP šalje odgovor korisniku na njegov zahtjev za pristup resursima u banci (korak 2 na slici 2.1). Prema OAuth2 specifikaciji korisnik, odnosno web preglednik korisnika, mora biti u stanju sačuvati vrijednost parametra *state* na lokaciji koja će biti dostupna samo TPP i korisniku. Nakon autentifikacije korisnika, slijedi preusmjeravanje nazad na TPP od strane banke (koraci 5 i 6 na slici 2.1). Kada je korisnik preusmjeren nazad na TPP, TPP će iz toga zahtjeva uzeti parametar *state* te provjeriti odgovara li ta vrijednost parametra onoj vrijednosti parametra *state* koju je TPP uključio u odgovor na zahtjev korisnika (korak 2 na slici 2.1). To će provjeriti tako da će gledati sjednicu s korisnikom . Ako vrijednosti nisu jednake TPP će pomisliti da se radi o CSRF napadu te će prekinuti tok. Isto tako zaštita od ovog napada pružena je korištenjem PKCE-a.

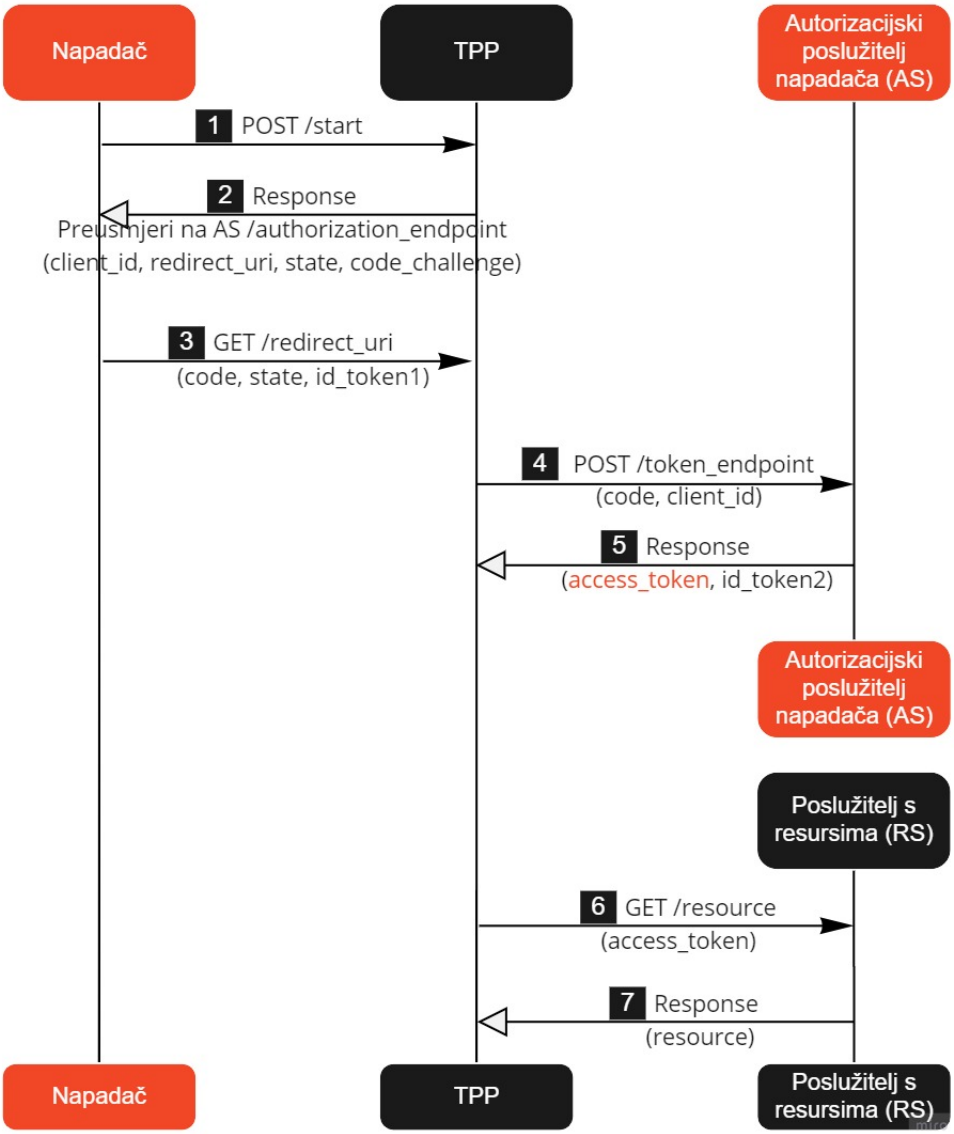
3.1.6. Napad ubacivanjem pristupnog tokena

Pristupni bi token mogao doći u ruke napadača ako napadač izvede uspješan *phishing*. *Phishing* je način dolaska do podataka ili informacija koje su od velike važnosti, npr. lozinke i kreditne kartice ili u ovom slučaju pristupni token za pristup resursima u banci. Ostvaruje se tako da su žrtve *phishing*-a kontaktirane putem elektroničke pošte, telefona ili telefonske poruke od strane nekog tko se predstavlja kao legitimna institucija ili zaposlenik legitimne institucije [17]. Žrtva *phishing*-a je tada prevarena u smislu da otvori maliciozni link koji mu je poslan, što može rezultirati instalacijom

zlonamjernog programa te u konačnici otkrivanje nekih osjetljivih podataka napadaču. Kada se govori o *phishing*-u pristupnog tokena, do njega je moguće doći ili kod TPP-a ili kod banke na autorizacijskom poslužitelju te na poslužitelju s resursima. Na primjer, moguće je da neki zaposlenik u tvrtki TPP-a nasjedne na *phishing* te će na taj način omogućiti napadaču da dođe do pristupnog tokena. Pošto je pristupni token povezan s TPP-om preko mTLS-a taj će se token moći direktno iskoristiti samo u toku gdje je i izdan, no postoji način kako napadač može izvesti ubacivanje pristupnog tokena te samim time osigurati si pristup resursima nekog korisnika u banci [8]. U ovom će se napadu napadač predstaviti kod legitimnog TPP-a kao legitimni korisnik koji želi pristupiti svojim resursima u banci. Napadač će isto tako biti u posjedu svojeg autorizacijskog poslužitelja. TPP će morati podržavati korištenje više od jednog autorizacijskog poslužitelja od kojih će, u ovom slučaju, jedan od njih biti autorizacijski poslužitelj samog napadača.

Napad će krenuti OAuth2 tokom kakav je prikazan na slici 3.1. Napadač će pokrenuti tok tako da će od TPP-a zatražiti pristup svojim resursima u banci i napravit će to tako da će kod odabira banke, odnosno autorizacijskog poslužitelja, izabrati autorizacijski poslužitelj pod njegovom kontrolom. Pošto je autorizacijski poslužitelj pod njegovom kontrolom koraci autentifikacije napadača mogu se preskočiti te je napadač u mogućnosti vratiti autorizacijski odgovor direktno nazad TPP-u (korak 3 na slici 3.1). Nakon toga TPP će nastaviti uobičajenim OAuth2 tokom, iskoristiti će autorizacijski kod koji mu je vraćen od strane napadača te će ga razmijeniti za pristupni token na krajnjoj točki za token autorizacijskog poslužitelja u posjedu napadača (korak 4 na slici 3.1). Napadač, odnosno njegov autorizacijski poslužitelj, će sada odgovoriti nazad TPP-u sa zahtjevom koji će sadržavati pristupni token (korak 5 na slici 3.1) prethodno ukraden iz nekog legitimnog OAuth2 toka gdje je legitimni korisnik htio pristupiti svojim resursima u banci preko legitimnog TPP-a. TPP tada korištenjem pristupnog tokena, pristupa poslužitelju s resursima korisnika kod banke od koje je legitimni korisnik prethodno zatražio resurse u toku u kojem je ukraden sam pristupni token (koraci 6 i 7 na slici 3.1).

Način na koji se moguće zaštiti od ovog napada predstavljen je u [8]. Preporuča se da kada TPP šalje zahtjev prema poslužitelju s resursima banke, u svrhu dohvaćanja resursa, u njega uključi i identitet autorizacijskog poslužitelja koji je izdao pristupni token. Na taj način bi poslužitelj s resursima banke mogao provjeriti radi li se o legitimnom autorizacijskom poslužitelju, odnosno o autorizacijskom poslužitelju same banke. To je ostvareno tako da se u zahtjeve doda parametar *id_token* koji će onda



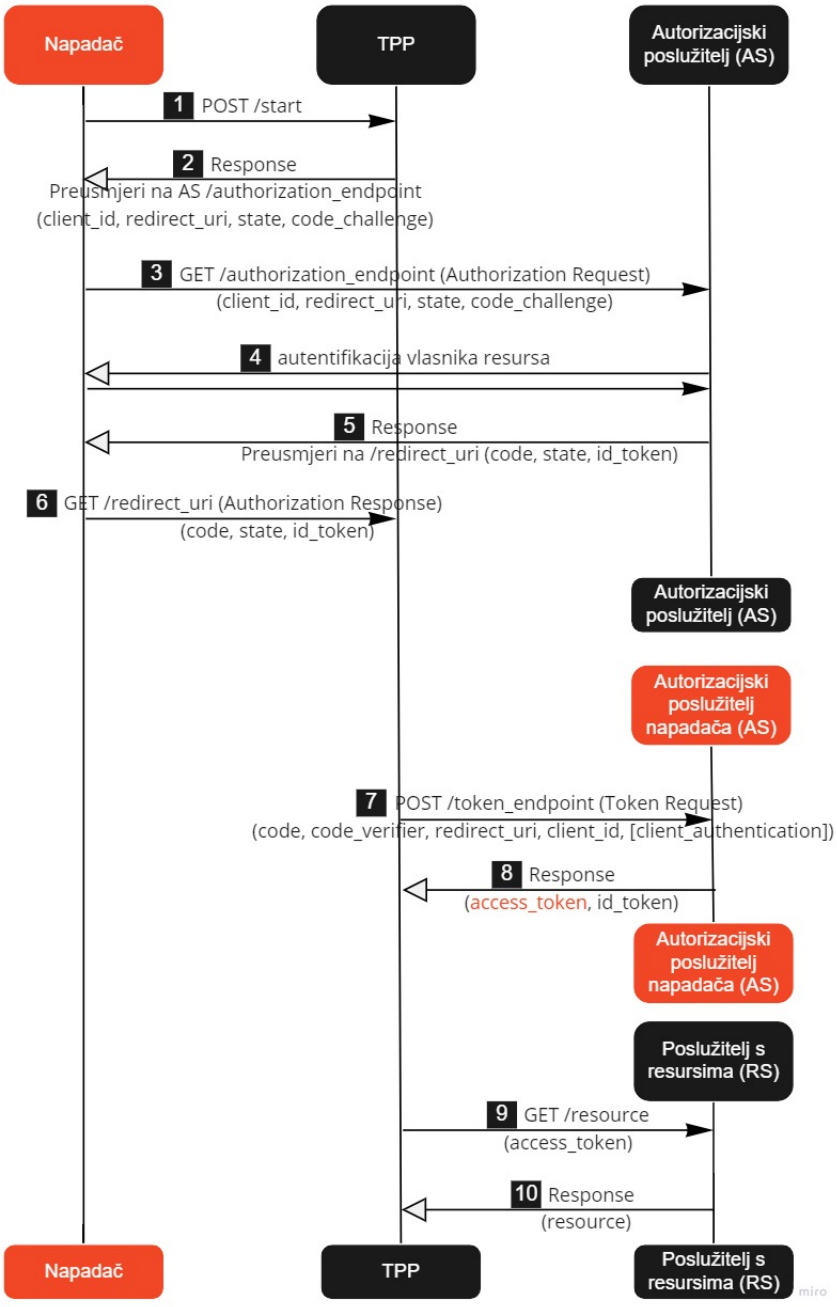
Slika 3.1: Napad ubacivanjem pristupnog tokena

povezivati autorizacijski poslužitelj s pristupnim tokenom koji izdaje (korak 5 na slici 3.1). Kako je pristupni token izdan od legitimnog autorizacijskog poslužitelja, odnosno povezan je na legitimni autorizacijski poslužitelj banke, novi parametar *id_token* neće biti povezan s autorizacijskim poslužiteljom banke već s autorizacijskim poslužiteljom napadača.

3.1.7. *ID Token Replay Attack*

Ovaj je napad još jedna verzija napada ubacivanjem pristupnog tokena kao i napad u poglavlju 3.1.6. Osim što je pristupni token povezan s TPP-om putem mTLS-a te ga samo on može direktno korisiti, token bi trebao biti siguran čak i ako je URI krajnje točke za dohvaćanje tokena kontroliran od strane napadača [8]. Zahtjev za tokenom može iscuriti (korak 7 na slici 2.1) na način da je zaposlenik ili programer tvrtke TPP-a manipuliran, odnosno prevaren, da vjeruje kako se URL krajnje točke za dohvaćanje tokena promijenio u URL koji je pod kontrolom napadača. Kada bi se to dogodilo, napadač bi došao u posjed parametara *code*, odnosno autorizacijskog koda. Naime, napadač neće biti u mogućnosti iskoristiti autorizacijski kod na legitimnoj krajnjoj točki za dohvaćanje tokena jer je autorizacijski kod povezan na TLS kanal te će ga samo TPP koji komunicira preko tog kanala s bankom moći iskoristiti u svrhu dohvaćanja pristupnog tokena kod autorizacijskog poslužitelja banke. Postoji scenario napada koji zaobilazi ovu i zaštitu od napada ubacivanjem pristupnog tokena opisanog u poglavlju 3.1.6.

Ovaj napad isto tako podrazumijeva da je pristupni token od nekog legitimnog TPP-a u posjedu napadača. Napadač će u napadu predstavljati normalnog legitimnog korisnika. Pokrenut će OAuth2 tok, prikazan na slici 3.2, tako da će TPP-u slati zahtjev za dohvaćanje resursa iz banke (korak 1 na slici 3.2). Proći će normalno autentifikaciju na autorizacijskom poslužitelju banke. Prvo odstupanje od normalnog toka događa se kada TPP šalje zahtjev s autorizacijskim kodom u svrhu dobivanja pristupnog tokena (korak 7 na slici 3.2). Sada će TPP, umjesto na krajnju točku za dohvaćanje tokena autorizacijskog poslužitelja banke, poslati zahtjev za dohvaćanje pristupnog tokena na krajnju točku za dohvaćanje tokena koja je pod kontrolom napadača. Kao što je rečeno na početku objašnjenja ovog napada, to je moguće kada je zaposlenik TPP-a zbog prevare uvjeren da se URL krajnje točke promijenio. Napadač će primiti ovaj zahtjev i vratiti pristupni token koji je prethodno ukraden nekom legitimnom TPP-u. Isto tako u zahtjevu koji će poslati TPP-u nalaziti će se i parametar *id_token* (korak 8 na slici



Slika 3.2: *ID Token Replay Attack*

3.2) koji će u ovom slučaju biti valjan, odnosno poslan od legitimnog autorizacijskog poslužitelja banke, jer ga je napadač od banke primio u zahtjevu koji mu je autorizacijski poslužitelj banke poslao nakon autentifikacije (korak 5 na slici 3.2). TPP tada šalje zahtjev prema poslužitelju s resursima u svrhu njihova dohvaćanja te napadač ima pristup resursima korisnika iz čijeg je toka pristupni token i ukraden. Na ovaj je način napadač izbjegao zaštitu navedenu u poglavlju 3.1.6.

Zaštita od ovakvog scenarija napada ubacivanjem pristupnog tokena proizlazi iz toga da u parametar *id_token*, koji se vraća u koraku 8 (slika 3.2), autorizacijski poslužitelj banke uključi i sažetak pristupnog tokena koji se šalje TPP-u [8]. Napadač neće moći kreirati *id_token* s valjanim potpisom autorizacijskog poslužitelja banke pa posljedično neće moći pristupiti resursima korisnika na poslužitelju s resursima pošto tamo parametar *id_token* neće proći provjeru bez valjanog potpisa.

3.1.8. Curenje pristupnog tokena na poslužitelju s resursima

Napadač bi mogao napraviti svoj poslužitelj s resursima u svrhu dolaska do pristupnog tokena te posljedično resursima nekog korisnika u banci. Za ovaj napad moraju biti zadovoljene određene pretpostavke i preduvjeti [19]. Ovo bi podrazumijevalo da TPP nakon što je izmijenio autorizacijski kod za pristupni token, taj token ide razmijeniti za pristup resursima na poslužitelju s resursima koji je pod kontrolom napadača. Napadač tada dolazi u posjed pristupnog tokena te ga može iskoristiti za pristup resursima nekog korisnika tako da pokrene novi tok s TPP-om. Jedna od pretpostavki koja mora biti zadovoljena kako bi se ovaj napad izvršio je to da TPP nije vezan samo za jedan poslužitelj s resursima već ima mogućnost pristupa više njih. Isto tako, preduvjet ovog napada bi bio da je zaposlenik ili programer TPP-a prevaren u smislu da se zahtjev pošalje na poslužitelj s resursima koji je pod kontrolom napadača a ne na legitimni poslužitelj banke. To je moguće ostvariti *phishing*-om kao i u napadu objašnjenom u poglavlju 3.1.6.

Zbog zaštite mTLS-om napadač neće biti u mogućnosti ukradeni pristupni token direktno iskoristiti na poslužitelju s resursima banke čiji ga je autorizacijski poslužitelj izdao. Pošto mTLS-a povezuje pristupni token na TPP-a kojemu je taj token izdan, TPP će moći pristupiti poslužitelju s resursima isključivo u tom toku. Dodatna zaštita od napada ubacivanjem pristupnog tokena nalazi se u scenarijima opisanima u poglavljima 3.1.6. i 3.1.7. Kako TPP nebi pristupao poslužitelju s resursima koji je u vlasništvu napadača moguće je i da autorizacijski poslužitelj banke definira kojim

poslužiteljima s resursima TPP može pristupiti, odnosno može mu predati listu poslužitelja koji su sigurni.

3.1.9. *307 Redirect* napad

HTTP *307 Temporary Redirect* status kod preusmjeravanja govori da se resurs koji je bio zatražen, u ovom slučaju to će biti vjerodajnice korisnika kod autentifikacije na autorizacijskom poslužitelju banke, privremeno nalazi na drugom URI-ju [20]. Taj URI je definiran u *Location* zaglavlju zahtjeva. Specifično je još da će metoda koja se koristila kod za dohvaćanje resursa te tijelo tog zahtjeva biti jednaki i u zahtjevu za preusmjeravanje. Nigdje u OAuth2 standardu nije specificirana koja točno metoda se mora koristiti kod preusmjeravanja [7]. U ovom napadu pretpostavka je da se *307 Temporary Redirect* status kod koristi kod preusmjeravanja te da je TPP u vlasništvu samog napadača.

Kada korisnik započne OAuth2 tok u svrhu pristupa svojim resursima u banci, prvo se mora prijaviti na stranici od banke s važećim vjerodajnicama npr. korisničko ime i lozinka (korak 4 na slika 2.1). Nakon unosa podataka za prijavu u web formu, autorizacijski će poslužitelj banke putem HTTP *POST* metode te podatke poslati nazad na autorizacijski poslužitelj kako bi se podaci provjerili [19]. Ako su podaci u redu, ako korisnik postoji, korisnik će se automatski preusmjeriti na stranicu TPP-a (koraci 5 i 6 na slika 2.1). Kao što je pretpostavljeno autorizacijski poslužitelj banke preusmjeravanje čini koristeći *307 Temporary Redirect* status kod, odnosno koristi isto HTTP *POST*. Korištenjem tog koda preusmjeravanja u HTTP *POST* zahtjevu, koji će autorizacijski poslužitelj banke poslati TPP-u, nalaziti će se vjerodajnice od korisnika. Pošto je u ovom slučaju TPP maliciozan, odnosno kontroliran od strane napadača, napadač će sada doći u posjed vjerodajnica od legitimnog korisnika.

Preporuča se korištenje HTTP *303 See Other* status koda [7]. *303 See Other* status kod govori da se preusmjeravanje vrši na neki drugi resurs, u ovom slučaju nazad na TPP-a [9]. Kod ovog status koda preusmjeravanje nije povezano na resurs koji se dohvaćao nego na neku drugu stranicu ili resurs. To znači da vjerodajnice korisnika neće biti poslane TPP-u kada mu autorizacijski poslužitelj banke odgovara poslije uspješne autentifikacije korisnika.

3.1.10. Napad degradiranjem PKCE-a

PKCE se koristi kao glavna zaštita od napada ubacivanjem autorizacijskog koda (poglavlje 3.1.4.) te od CSRF napada (poglavlje 3.1.5.). Naime, neki autorizacijski poslužitelji će podržavati PKCE no neće ga staviti da bude obavezan. Ako je to slučaj, pretpostavka je da postoji zastavica unutar zahtjeva za autorizaciju kada TPP preusmjerava korisnika da se autentificira kod banke [19]. Zastavica bi govorila koristi li se PKCE ili ne tako da bi joj se vrijednost sama mijenjala ovisno je li prisutan PKCE *code_challenge* ili ne. Ta bi zastavica mogla biti promijenjena od napadača što bi uzrokovalo napad degradiranjem PKCE-a.

Neki će korisnik htjeti dohvatiti svoje resurse u banci te će započeti normalan tok s TPP-om (slika 2.1). TPP će preusmjeriti korisnika na krajnju točku autorizacije, kako bi se korisnik autentificirao kod autorizacijskog poslužitelja banke, te će u taj zahtjev ubaciti *code_challenge*. Napadač sada radi isto kao i korisnik, započinje novi tok s TPP-om u svrhu pristupanja svojim resursima. TPP će isto tako u zahtjev prema autorizacijskom poslužitelju banke ubaciti *code_challenge* za tok s napadačem. Prije nego je napadač preusmjeren na autorizacijski poslužitelj on će korištenjem nekih programskih alata, taj zahtjev presresti na svojem uređaju te će iz zahtjeva maknuti *code_challenge*. Nadalje, napadač će se autentificirati kod banke. Pošto se u zahtjevu ne nalazi *code_challenge* banka će pretpostaviti da se PKCE ne koristi te će vratiti autorizacijski kod koji neće biti povezan s *code_challenge*-om. Napadač će sada manipulacijom, korisnika preusmjeriti na URL koji se nalazi unutar odgovora autorizacije nakon što se napadač autentificirao kod autorizacijskog poslužitelja, odnosno preusmjeriti će ga nazad na TPP-a no sa autorizacijskim kodom napadača a ne vlastitim kodom korisnika. Ovdje pretpostavljena još jedna stvar, a to je da TPP neće koristiti, odnosno neće pregledavati parametar *state* jer će se za zaštitu od CSRF napada oslanjati na PKCE [19]. U praksi je pokazano da mnogi TPP-i ne provjeravaju vrijednost parametra *state* na pravi način [19]. Dalje će TPP ići razmijeniti autorizacijski kod za pristupni token. Pošto se u ovom zahtjevu neće nalaziti autorizacijski kod koji je vezan uz neki *code_challenge*, autorizacijski poslužitelj neće ni pregledavati prisutnost *code_verifier*-a te će izdati pristupni token. Na ovaj će način korisnik pristupiti računu napadača. Ovo je slično kao i CSRF napad naveden u poglavlju 3.1.5. Svaki podatak ili informaciju koju korisnik ostavi na računu napadača, kao broj kartice i slično, napadač će moći vidjeti kada se sljedeći put prijavi u svoj račun.

Temeljni način zaštite bi bio da se PKCE stavi kao obavezan za korištenje u svim to-

kovima. Korištenje i pravilno pregledavanje parametra *state* isto tako dovodi do zaštite od ovog napada. U koraku 7 OAuth2 toka prikazanog na slici 2.1, autorizacijski će poslužitelj u zahtjevu za token poslanom od TPP-a, između ostalog dobiti i parametar *code_verifier* koji bi služio za potvrdu *code_challenge*-a vezanog uz autorizacijski kod. U ovom je napadu bilo pretpostavljeno kako autorizacijski poslužitelj banke neće ni gledati prisutnost *code_verifier*-a ako vidi da autorizacijski kod nije povezan s nikakvim *code_challenge*-om. Ako se dogodi da je autorizacijski kod izdan bez prisutnosti *code_challenge*-a, a TPP u svojem zahtjevu za token prema autorizacijskom poslužitelju uključi *code_verifier* taj bi zahtjev morao biti odbijen.

3.2. Napadi vezani uz API

Ovo će poglavlje sadržavati opise napada i prijetnji koje je moguće provesti zbog nedostataka ili pogrešne implementacije određenih zaštitnih mehanizama na samim API-ima banaka.

3.2.1. Slomljena autentifikacija korisnika

Napadači koji izvršavaju napade vezane uz slomljenu autentifikaciju korisnika koriste već poznate podatke o korisnicima potrebne za prijavu na račune korisnika u banci ili manipuliraju tim podacima u svrhu dobivanja pristupa računima korisnika. Jedan od takvih napada je *Credential stuffing*. *Credential stuffing* napad odvija se automatizacijom prijavljivanja, odnosno unošenja korisničkog imena i lozinke, na određeni web servis, npr. prijava na stranici od banke. Napadač će doći u posjed ukradenih korisničkih imena i lozinki tako što su oni iscurili na nekim drugim servisima [22]. Mnogo ljudi koristi jednake lozinke za prijavu na mnogo različitih računa [22]. Napadač može isprobavati šifre na mnogo različitih računa u banci istovremeno.

U OAuth2 toku napadač bi se predstavljao korisnika koji želi pristupiti računu u banci. Počeo bi, kao i uvijek, slanjem zahtjeva TPP-u u svrhu pristupanja resursima u banci. TPP bi ga preusmjerio na krajnju točku autorizacije (korak 2 na slici 2.1) gdje bi se napadač trebao autentificirati kod autorizacijskog poslužitelja. Konkretno ovaj napad bio bi vezan za autentifikaciju korisničkim imenom i lozinkom kod banke. Ako napadač pogodi korisničko ime i lozinku, odnosno u svom posjedu ima valjano korisničko ime i lozinku, autorizacijski će poslužitelj vratiti autorizacijski kod koji će TPP moći

razmijeniti za pristupni token. Time napadač ima pristup svim resursima i podacima nekog korisnika kojemu su korisničko ime i lozinka prethodno ukradeni. Napadač bi imao pristup podacima o kreditnim karticama, mogao bi obavljati plaćanja, prebacivati resurse na svoj ili neke druge račune. Isto tako, napadač bi valjana korisnička imena i lozinke mogao prodati drugima.

Još jedan scenario ovog napada mogao bi se izvršiti ako napadač zna korisničko ime ili e-mail korisnika, ovisno što se koristi kod prijave na banku. Mogao bi pokrenuti uslugu promijene lozinke računa [21]. Banka bi mogla zahtijevati upisivanje nekog brojčanog PIN-a što omogućava napadaču da napravi skriptu koja bi bila u mogućnosti isprobavati sve kombinacije npr. 6 brojeva u svrhu pogađanja pravog PIN-a.

Preporučena rješenja za zaštitu od ovog napada nalaze se u [22, 13]. Primarna zaštita od ovakov tipa napada bila bi korištenje *Multi-factor* autentifikacije. *Multi-factor* autentifikacija zahtjeva od korisnika da koristi dva ili više načina autentifikacije. Npr. uz prijavu na banku korisničkim imenom i lozinkom, banka zahtjeva da se još upiše PIN ili token za konačnu autentifikaciju. Isto tako, zaštita se pronalazi i u ograničenju broja pokušaja kod upisivanja pogrešnog korisničkog imena i lozinke te, u određenim slučajevima, PIN-a. Ograničenje broja pokušaja onemogućiti će napadaču da isproba sve parove korisničkih imena i lozinki, koje ima u posjedu, te sve kombinacije brojeva kod pogađanja PIN-ova. Još jedna od zaštita za ovakve napade bilo bi korištenje *Completely Automated Public Turing test to tell Computers and Humans Apart* (CAPTCHA). CAPTCHA zahtjeva od korisnika da ručno riješe neki test kojim bi dokazali da su ljudi. Na taj način ne mogu koristiti automatizirane skripte.

3.2.2. Manjak validacije unosa korisnika i napadi ubacivanjem

Kada se korisnik prijavljuje na račun, odnosno autentificira na autorizacijskom poslužitelju banke, morati će u web formu unijeti svoje vjerodajnice npr. korisničko ime i lozinku. Svaki put kada postoji forma za unos podataka, pogrešna validacija i loša kontrola unesenih podataka može dovesti do različitih napada ubacivanjem od kojih je najpoznatiji napad ubacivanjem SQL izraza [24]. Podaci uneseni kroz formu moraju se validirati prije nego se pošalje zahtjev prema bazi [13]. Za napad ubacivanjem nije nužno da postoji web forma. Svaki poslani zahtjev prema API-ju se isto mora validirati. Isto vrijedi i ako je moguće učitati datoteke na web stranici [13]. Napadi ubacivanjem mogu dovesti do gubitka podataka ili krađe podataka, odnosno napadač bi mogao ostvariti pristup nekim podacima koji su osjetljivi npr. informacije o karticama

u banci ali i do nekih osobnih podataka kao adrese korisnika i slično [21].

Kod napada ubacivanjem SQL izraza, napadač ubacivanjem SQL izraza dobiva pristup bazi tako da potakne izvršavanje SQL upita koji će rezultirati pristupanjem resursima iz baze podataka. Ranjivost na ovaj napad bi napadaču moglo osigurati pristup cijeloj bazi podataka. Primjer napada bi bio kada korisnik pristupi web stranici banke te se želi prijavit na svoj račun. To će se obavljati putem web forme gdje će upisati korisničko ime i lozinku. Kada upiše svoje podatke, poslati će se zahtjev na API te će izvršiti SQL upit na bazu. SQL upit bi npr. izgledao ovako:

```
SELECT *
FROM users
WHERE username = ' + some_username + '
    AND password = ' + some_password + '
```

Varijable *some_username* i *some_password* bi se zamijenile vrijednostima koje unese korisnik. To će dohvatiti sve podatke vezane za račun korisnika. U web formi, na mjestu gdje se unosi lozinka, uz lozinku bi se mogao dodati i SQL izraz. Na primjer ako unese *password1' OR 1=1*. SQL upit, s unesenim vrijednostima korisnika, bi se sada izvršio u bazi i izgledao bi ovako:

```
SELECT *
FROM users
WHERE username = 'username1'
    AND password = 'password1' OR 1=1'
```

Ovaj SQL izraz dohvaća sve vrijednosti vezane uz račun korisnika, npr. osobne podatke ili kreditne kartice, kojemu je korisničko ime jednako *username1* te lozinka jednaka *password1* ili, ako to nije zadovoljeno, vraća račune svih korisnika jer će uvjet *1=1* uvijek biti zadovoljen [15].

Za zaštitu od napada ubacivanjem SQL izraza mora se provesti validacija podataka koji se unose. Svaki unos bi se trebao provjeriti, odnosno filtrirati, da li odgovara očekivanim vrijednostima koje se moraju definirati [21]. To bi trebalo implementirati za sve krajnje točke API-ja. Isto tako, mogao bi se ograničiti broj računa koji se mogu vratiti odjednom što bi bila dobra zaštita za navedeni scenario napada. Za unesene vrijednosti mogla bi se provesti provjera da li sadržavaju SQL izraze. Korištenje određenih knjižica štiti od napada ubacivanjem SQL izraza [21].

3.2.3. Slomljena autorizacija na razini objekta

Slomljena autorizacija na razini objekta (*Broken Object-Level Authorization*, BOLA) naziva se i Nesigurno direktno referenciranje objekata (*Insecure Direct Object Reference*, IDOR). Objekt može biti bilo što unutar sustava, može biti npr. račun korisnika, plaćanje koje je korisnik obavio, kreditna kartica korisnika itd. Objekti često imaju svoje identifikatore kako bi se mogli referencirati i koristiti u drugim dijelovima aplikacije, odnosno API-ja. Ranjivost na BOLA je prisutna ako API dopusti pristup informacijama nekome, kome pristup tim podacima inače ne bi bio dozvoljen. Najčešće se ovakav napad provodi promjenom identifikatora negdje gdje korisnik može unijeti podatke, npr. URL ili slanjem zahtjeva [13]. Promjenom vrijednosti identifikatora na vrijednost identifikatora nekog korisnika, napadač bi dobio pristup informacijama koji nisu njegovi ako API ne provjerava je li osoba koja je zatražila te podatke ujedno i njihov vlasnik. Na primjer, kada bi se napadač prijavio na svoj račun u banci i pristupio jednom od svojih bankovnih računa, bio bi preusmjeren na sljedeću poveznicu: *https://bank/balance?acc=123*. Identifikator napadača u banci u ovom slučaju je 123. Sada bi napadač jednostavno promijenio identifikator u neku drugu vrijednost te tako pristupio računu nekog drugog korisnika. Osim već spomenutog tipa ovog napada, kada se mijenja identifikator korisnika, postoji i tip zasnovan na identifikatoru samog objekta [5]. Na primjer, neka je jedan od objekata u banci provedeno plaćanje. U sklopu API-ja postoji krajnja točka za dohvaćanje takvog objekta i neka izgleda ovako: */bank/api/payment?payment_id=123*. Sada bi napadač mogao na isti način, kao i kod tipa napada vezanog za identifikator korisnika, promijeniti identifikator provedenog plaćanja te doći u posjed informacija vezanih za drugog korisnika.

Jedan ovakav napad, koji nije vezan za banku ali služi kao dobar primjer ovog napada, dogodio se 2010. godine tvrtki AT&T [23]. Curenje podataka dogodilo se korisnicima koji su bili vlasnici *iPad 3G* uređaja koji je tada bio isključivo dostupan uz AT&T kao mobilnog operatera. Preko 114,000 računa bilo je ugroženo. Napad je bio povezan s identifikatorom koji je vezao svakog od pretplatnika za AT&T mrežu, odnosno njihovu SIM karticu. Taj je identifikator poznat kao *Integrated Circuit Card Identifier* (ICC-ID). Napadači su napravili napad preko skripte koja je bila dostupna na stranici AT&T-a. Ako se u HTTP zahtjev uključio ICC-ID ta bi skripta vratila adresu elektroničke pošte korisnika vezanog uz taj ICC-ID.

Zaštita od BOLA napada, odnosno ranjivosti, proizlazi iz implementacije autorizacijskih mehanizama koji će povjeravati je li osoba koja pokušava pristupiti podacima

ujedno i osoba prijavljena na račun koji sadrži te podatke. Isto tako, trebalo bi provjeravati svaku akciju koja se vrši na objekte koji su privatni, npr. čitanje, brisanje, izmjena itd.

3.2.4. Pretjerano izlaganje podataka

Pretjerano izlaganje podataka bi se dogodilo kada bi API banke vraćao više informacija nego je potrebno i/ili zahtjevano od strane korisnika. To je pogotovo opasno kada se vraćaju podaci koji su osjetljivi i privatni te koje bi napadač mogao zloupotrijebiti. Najčešće se API-ji oslanjaju na TPP da te podatke filtrira i korisniku ne vraća ono što nije potrebno za uslugu [21]. No osim podataka koji su u prijenosu između banke i TPP-a pretjerano se izložiti mogu i podaci koji su spremljeni kao npr. spremljene lozinke, kreditne kartice ili općenito vjerodajnice korisnika. Kako bi se zaštitili, takvi podaci zahtjevaju kriptiranje [13].

Korisnik bi mogao zatražiti od banke da mu vrati broj njegove kartice. Pokrenuo bi OAuth2 tok s TPP-om u svrhu dohvaćanja te informacije. Tok bi išao kao i svaki, korisnik bi se autentificirao na autorizacijskom poslužitelju banke kako bi dobio autorizacijski kod. Preusmjeravanje se vrši na TPP-a koji će onda taj kod zamijeniti za pristupni token s kojim će na poslužitelju s resursima pristupiti podacima korisnika. Kada će TPP na poslužitelju s resursima dohvatiti informaciju koju je korisnik zatražio, API banke će vratiti više informacija nego je trebao. Npr. TPP pošalje zahtjev *GET /api/v1/cards?id=123* prema poslužitelju s resursima banke te mu banka vraća JSON objekt:

```
{
    "CVV": "677",
    "creditCard": "1234567901234",
    "id": 123,
    "user": "API",
    "validUntil": "1997"
}
```

Korisnik će na svojem pregledniku vidjeti samo broj kartice te njen datum isticanja jer će ostatak biti filtriran od strane TPP-a. Napadač bi mogao doći to tih podataka ako njuškanjem prometa koji se događa između TPP-a i poslužitelja s resursima banke [13]. No pošto bi komunikacija između TPP-a i banke trebala biti zaštićena TLS-om

to neće biti moguće. Drugi način je da napadač do tih podataka dođe kod TPP-a. Npr. moguće je da *phishing*-om prevari jednog od zaposlenika ili programera tvrtke TPP-a, kao što je objašnjeno u poglavlju 3.1.6.

Prethodni napad vezan je za podatke koji se prenose između TPP-a i banke te krajnje između TPP-a i korisnika. Neka su podaci o kreditnim karticama spremljeni u bazi podataka. Baza obavlja automatsko kriptiranje podataka. No dohvaćanjem tih podataka iz baze vrši se automatsko dekriptiranje podataka. Napadač bi mogao izvesti napad ubacivanjem SQL izraza koji je opisan u poglavlju 3.2.2. i na taj način doći do informacija o kreditnim karticama.

Kao što je već spomenuto jedna od zaštita od ove vrste napada je korištenje TLS kod komunikacije između TPP-a i banke zbog toga što tada napadač neće moći njuškanjem prometa, između te dvije strane, doći do podataka. Nadalje, filtriranje podataka koji dolaze od banke ne bi trebao raditi TPP, pogotovo ako se radi o osjetljivim podacima. Kada API šalje odgovore s podacima, umjesto slanja svih podataka nekog objekta, odgovor bi trebao sadržavati samo minimalan broj podataka moguć za obavljanje usluge koju je korisnik zatražio [1]. Isto tako spremljeni podaci bi se trebali klasificirati, jer su neki osjetljiviji od drugih. Na taj način osjetljivije podatke bi trebalo dodatno zaštititi i dodatno provjeravati kada se dohvaćaju te tako smanjiti rizik od njihove zlouporabe [13].

3.2.5. Manjak limitiranja pokušaja i resursa

Neke se krajnje točke API-ja mogu pozivati mnogo puta po sekundi. Ako nije implementirano limitiranje pokušaja i resursima, može doći do zasićenja sustava. Zbog velikog broja zahtjeva koji se šalju API, odnosno sustav na kojemu se API nalazi, trošiti će sve više i više dostupnih resursa. Što je više različitih krajnjih točaka API-ja to je više resursa potrebno. Resursi su radna memorija, procesor (*Central Processing Unit*, CPU) i/ili sustav za pohranu podataka. Iskorištavanje svih resursa može dovesti do napada uskraćivanja usluge (*Denial of Service*, DoS) što će onemogućiti daljnje funckioniranje API-ja [13]. Ovakvu ranjivost API-ja mogli bi iskorištavati napadači kako bi provodili i druge napade, npr. *Credential stuffing* napada navedenog u poglavlju 3.2.1.

Jedan od primjera ovog napada bio bi ako API ima krajnju točku s koje bi se mogli dohvatiti određeni podaci. Podrazumijevani broj podataka koji bi se vraćao je postavljen na 100. Ako bi napadač išao slati zahtjev prema toj krajnjoj točki te bi u zahtjevu

promijenio pretpostavljeni parametar, nazovimo ga *limit*, na npr. 100,000 dogodilo bi se zasićenje sustava te bi sustav ili prestao raditi ili ne bi radio kako treba [13].

Ispravna implementacija limitiranja pokušaja i resursa nužna je za zaštitu od ovakvih napada. Ograničenje broja zahtjeva koje korisnik/napadač mogu poslati na krajnje točke, validiranje parametara kao što je *limit* u navedenom scenariju te ograničavanja količine podataka koje krajnje točke mogu vratiti odjednom neki su od detalja implementacije koji će spriječiti ovaj napad [21].

3.2.6. Slomljena autorizacija na razini funkcija

BOLA ranjivost javlja se kada napadači mogu pristupati resursima, odnosno objektima API-ja, kojima nebi smjeli imati pristup jer su to resursi drugih korisnika. To se radi na način da se identifikatori korisnika ili objekata zamijene s identifikatorom nekog drugog korisnika ili objekta koji mu pripada unutar zahtjeva koji se šalje ili na URL-u kao što je objašnjeno u poglavlju 3.2.3. Slomljena autorizacija na razini funkcija (*Broken Function-Level Authorization*, BFLA) je sličan problem, no ovdje se radi o tome da API ne ograničava pristup i korištenje funkcija na korisnike koji su autorizirani na korištenje tih funkcija. Prvenstveno se misli na to kada obični korisnik/napadač prisupi i koristi funkcije slanjem zahtjeva na krajnje točke API-ja kojima mogu upravljati samo administratori [21]. Danas API-ji sadrže mnogo različitih uloga te je iz tog razloga teže implemetirati pristupe za sve njih. Kada bi se ova ranjivost iskoristila, napadač bi mogao doći u posjed informacija korisnika koje su osjetljive te ih zlouporabiti.

Prilikom dohvaćanja svojih podataka iz banke korisnik bi izazvao da API pošalje zahtjev *GET /api/user/my_financial_info*. Napadač bi mogao poslati taj zahtjev na način *GET /api/admin/users/all_info*. Taj bi se zahtjev, s dobro implementiranom autorizacijom na razini funkcija, mogao izvršiti samo ako ga je poslao administrator. Na ovaj način napadač bi mogao doći u posjed osjetljivih informacija o svim korisnicima iz banke, npr. adrese stanovanja, adrese elektroničke pošte, brojeva kartica i slično. Isto tako ako postoji ova ranjivost kod API-ja napadač bi bio u mogućnosti promijeniti tip zahtjeva koji se šalje i tako poslati *DELETE* zahtjev umjesto *GET* zahtjeva te izbrisati podatke određenih korisnika. Inače bi zahtjev tipa *DELETE* bilo moguće poslati samo ako je netko administrator.

Kao zaštita od ovog napada morali bi postojati određeni mehanizmi koji bi podrazumijevali pristup nije dozvoljen [21]. Tek bi dodatnim provjerama uloga autorizirali slanje određenih zahtjeva, odnosno korištenje određenih funkcija. API bi trebao reagirati na

svaki zahtjev koji ima neuobičajene parametre ili ako se radi o zahtjevima *DELETE* i *POST* koje mogu slati samo administratori.

3.2.7. Masovno dodjeljivanje

Masovno dodjeljivanje je automatsko povezivanje parametara iz HTTP zahtjeva u varijable ili objekte u kodu. Parametri u zahtjevima su povezani na vrijednosti koje korisnik unese na web stranici. Na primjer, korisnik unese korisničko ime i lozinku koji se onda vežu za varijable objekta korisnik kao što su *user.username* i *user.password*. Putem parametara u zahtjevu te se vrijednosti šalju prema bazi kako bi se izvršila autentifikacija korisnika. Isto vrijedi i ako korisnik želi promijeniti neke podatke vezane uz njegov račun, poslat će se zahtjev prema bazi koji će ažurirati vrijednosti. Pošto se povezivanje parametara izvodi automatski, napadač bi mogao to iskoristiti tako da promijeni neke parametre u zahtjevima koji su vezani uz neke osjetljivije objekte ili varijable koje ne bi trebale biti izložene korisnicima, npr. *user.is_admin* [21]. Tako se mogu iskoristiti ranjivosti masovnim dodjeljivanjem.

Primjer iskorištavanja ove ranjivosti bio bi ako korisnik želi promijeniti neke podatke na svom računu u banci, npr. želi ažurirati svoje korisničko ime. Korisnik bi otvorio svoj račun te bi se ažuriranjem podataka automatski poslao zahtjev koji bi izgledao kao *PUT /api/v1/users/me*, s parametrom:

```
{"user_name":"some_username"}
```

Parametar iz tog objekta API bi automatski vezao za objekt, odnosno varijablu u objektu, i ažurirao se u banci. Naime, ako bi se poslao zahtjev *GET /api/users/me* banka bi vratila JSON objekt koji bi izgledao kao:

```
{"user_name":"some_username","credit_balance":10}
```

Napadač može otkriti to, npr. na svojem računu, te poslati novi zahtjev koji bi izgledao isto kao i prvi *PUT* zahtjev no sada bi u JSON objekt uključio i novi parametar *credit_balance*:

```
{"user_name":"attacker","credit_balance":10000}
```

Pošto je pretpostavka da je krajnja točka API-ja na koju se šalje ovaj zahtjev ranjiva na masovno dodjeljivanje, napadač će sada na svoje račun dodati sredstva bez da ih je stvarno uplatio.

Zaštita od ovoga može se dobiti tako da se ne koriste funkcije za automatsko povezivanje onoga što unese korisnik [13]. Neki se podaci moraju moći promijeniti od strane korisnika pa bi trebalo definirati koje su to točno varijable ili objekti. Isto tako trebalo bi i provjeravati sve zahtjeve poslane na API te koje parametre smiju sadržavati a koje ne.

3.2.8. Neispravno upravljanje resursima

Kada se izbaci nova verzija API-ja, stare verzije često znaju biti zaboravljene i ne ažuriraju se [21]. Pošto se prestanu ažurirati, takvi API-ji neće biti zaštićeni kao i novije verzije. Iz tih razloga napadi na takve API-je često mogu proći neopaženo [2]. Napadač bi mogao to iskoristiti i pristupiti nekim osjetljivim podacima putem starijih verzija API-ja koje su i dalje povezane s bazom podataka te resursima. To se naziva neispravno upravljanje resursima. Nedostatak, ili neispravnost, dokumentacije može dovesti do zapostavljanja starijih verzija API-ja jer se neće znati čemu točno služe ako npr. osoba koja je pisala dokumentaciju napusti tvrtku [2].

Ako je npr. banka izbacila novu verziju API-ja s adresom *api.bank.com/v2*, a stara verzija na adresi *api.bank.com/v1* i dalje funkcionira te je povezana s bazom podataka, napadač bi mogao zamijetiti da ako postoji verzija 2 postoji mogućnost da je verzija 1 još u pogonu. Zamjenom *v2* sa *v1* u adresi napadač će pristupiti staroj verziji API-ja. Pošto je ta stara verzija nezaštićena, napadač će slanjem zahtjeva na staru verziju moći pristupiti osjetljivim podacima korisnika.

Ovakav napad bi se mogao spriječiti ako se i starije verzije API-ja zaštićuju sigurnosnom stijenom (*firewall*) kojom se zaštićuje i nova verzija. Kreiranjem dostojne dokumentacije trebalo bi se pratiti čemu koji API služi, njihove verzije, namjenu, sigurnosni status i kontrole pristupa samim API-jima. Nakon što je novi API u punom pogonu, kada su provedeni testovi, stari API-ji koji se više ne koriste bi se trebali ukloniti.

3.2.9. Nepravilno bilježenje i praćenje

Iako ovo nije ranjivost koja bi uzrokovala napad sam po sebi, nepravilno ili nedovoljno praćenje i bilježenje događaja na API-ju može dovesti do ne poduzimanja određenih mjera kako bi se spriječili potencijalni napadi na API. Ako se bilježenje i praćenje ne implementira moguće je da napadač izvrši neki napad na API bez da se za njega uopće zna [13]. Na taj način se ranjivost API-ja nebi mogla ni otkriti a time niti ispraviti.

Napadači će također imati više vremena za provođenje napada, mogli bi isprobavati mnogo puta različite prijetnje bez da budu opaženi. Prijetnje ili napadi se moraju otkriti što ranije kako nebi uzrokovali još veću štetu.

Na primjer da korisnik želi pristupiti svojim resursima u banci, pokrenuo bi normalan OAuth2 tok (slika 2.1) komunikacijom s TPP-om. Nastavlja se kao i inače, korisnik se autentificira na autorizacijskom poslužitelju banke, dobije se autorizacijski kod koji će TPP razmijeniti za pristupni token i pristupiti resursima. Neka se sada dogodio napad ubacivanjem pristupnog tokena opisan u poglavlju 3.1.6. te da API nema implementiranu zaštitu za taj napad. Napadač bi pristupio resursima korisnika. Zbog nedostatka praćenja i bilježenja događa banka neće se ni znati da je do napada došlo dok se ne dogode neke posljedice direktno vezane uz korisnika, npr. krađa sredstava s računa i slično. Ako praćenje i bilježenje postoji no nije implementirano na pravi način, korisnik će biti obaviješten da je došlo do mogućeg proboja u njegov račun no banka neće moći vidjeti kojim je točno resursima napadač došao u posjed te koje su točno posljedice.

Osim što je potrebno implementirati praćenje i bilježenje događaja, potrebno je implementirati na pravi način. Bilješke moraju sadržavati dovoljno korisnih informacija kako bi se prijetnje mogle što bolje sanirati [13]. Bilješke ili dnevnik s bilješkama bi trebao biti siguran i bilo bi dobro da sadržava sve pokušaje autentifikacije, zabrane pristupa i greške kod validacije unosa korisnika [21].

4. Implementacija napada

U ovom će poglavlju biti prikazano kako se napadi, objašnjeni u poglavlju 3., mogu implementirati pomoću programskog alata razvijenog u sklopu diplomskog rada [14]. Programski je alat razvijen u svrhu automatiziranja slanja zahtjeva prema API-ima uz dodatnu podršku i naglasak na API-je vezane uz PSD2 direktivu. Alat će biti korišten za slanje svih zahtjeva između TPP-a i banke te će imitirati slanje zahtjeva koji se šalju između korisnika i banke, npr. kada se autentificira na stranici banke, kako bi se mogla pokazati simulacija. Treba napomenuti da je alat izrađen prema dokumentaciji za Erste API [10].

Slijedi objašnjenje nekih od razreda i metoda koje će se koristiti kod implementacije. Razred *Requester* služi za slanje zahtjeva prema API-ju. Razred se inicializira predajom parametra *request_method* koji definira metodu HTTP zahtjeva koji će se koristiti (*GET, POST, PUT*), parametra *base_url* koji definira na koji će se API zahtjevi slati te parametar *endpoint* koji definira krajnju točku API-ja na koju se zahtjev šalje. Ovaj razred sadrži određene metode. Metoda *setQueryParameter* dodaje parametar u zahtjev koji je potreban za dohvaćanje različitih resursa s određenih krajnjih točaka. Metoda *setBody* postavlja vrijednosti tijela zahtjeva. Zaglavlje zahtjeva može se postaviti metodom *setRequestHeader*. Zahtjevi se mogu poslati metodom *sendRequest*. Kada se implementiraju zahtjevi koji se šalju od TPP-a postoje razredi koji su izvedeni iz razreda *Requester* te sadržavaju pretpostavljene parametre i varijable potrebne za pristup API-jima. To su razredi *ConsentRequester* i *PSD2Requester*. Razred *ConsentRequester* se koristi kako bi se napravio *consent* zahtjev koji TPP šalje kako bi se pokrenuo OAuth2 tok s korisnikom. Pošto je ovaj alat razvijen u testnom okruženju, za navedeni ERSTE API, neće postojati fizički korisnik koji će se autentificirati kod banke nego je pretpostavljeno da se autentifikacija korisnika uspješno izvrši. Nadalje, razred *PSD2Requester* služi za kreiranje *status* zahtjeva koji će samo provjeravati je li autentikacija prošla, no kao što je rečeno u ovom slučaju autentifikacija uvijek prolazi. Razredom *PSD2Requester* kreiraju se i *token* zahtjev, koji je *POST* zahtjev kojim će

TPP zatražiti pristupni token kako bi pristupio resursima korisnika, te *resource* zahtjev kao *GET* zahtjev kojemu se definira krajnja točka s koje će dobaviti resurse. Kod implementacije slanja zahtjeva od strane TPP-a koriste se razredi *AccountInformationServiceTPP* i *PaymentInitiationServiceTPP* ovisno u usluzi koja se koristi. U te razrede će se kao parametri predati svi zahtjevi koji će se onda redom izvršavati te će razred automatski uzimati informacije iz zahtjeva i stavljati ih u sljedeće zahtjeve, ako je to potrebno.

4.1. Implementacija napada vezanih uz OAuth2

Ovo će poglavlje sadržavati implementaciju napada putem različitih razreda sadržanih u alatu. Putem programskog alata biti će simulirano slanje zahtjeva prema ili API-ju banke ili API-ju pod kontrolom napadača, od strane TPP-a. Pretpostavka da se *consent* i *status* zahtjevi uvijek kreiraju i šalju prvi na jednak URL API-ja kao i ostali zahtjevi pa oni neće biti dodatno navedeni kod napada gdje to nije potrebno. Objašnjenja svih napada nalaze se u poglavlju 3.1.

Kod *Mix-up* napada pretpostavlja se da TPP razmijenjuje autorizacijski kod na autorizacijskom poslužitelju napadača. U programskom alatu korištenjem razreda *PSD2Requester* kreira se *token* zahtjev s URL-om API-ja napadača. Zahtjevi se šalju preko razreda *AccountInformationServiceTPP*. Ako ne postoje određene mjere zaštite napadač će doći u posjed autorizacijskog koda na svojem autorizacijskom poslužitelju.

Napad ubacivanjem autorizacijskog koda pretpostavlja da napadač koristi ukradeni autorizacijski kod korisnika te ga koristi kod TPP-a za npr. obavljanje plaćanja preko računa korisnika. Pomoću razreda *Requester* definira se *POST* zahtjev, URL API-ja koji se koristi te željena krajnja točka npr. */payment*. U tijelo (*body*) zahtjeva mora biti definirano sljedeće: valuta, iznos, IBAN onoga tko plaća te IBAN onoga kome se plaća. Pomoću metode *setCertificates* postavljaju se certifikati u zahtjevu, a pomoću *setRequestHeader* potrebna zaglavlja za pristup API-ju. Nadalje se definira *resource* zahtjev pomoću razreda *Requester*. Svi se zahtjevi redom pošalju pomoću razreda *PaymentInitiationServiceTPP*. Zahtjevi se redom šalju kada korisnik/napadač želi provesti plaćanje. Ako nisu poduzete potrebne mjere zaštite napadač će moći provesti plaćanje s računa korisnika na vlastiti račun.

Kod *Login Cross Site Request Forgery* napada pretpostavlja se da je napadač ubacio svoj vlastiti autorizacijski kod u zahtjev korisnika prema TPP-u kako bi korisnik pris-

tupio računu napadača. Korištenjem razreda *PSD2Requester* kreira se *token* zahtjev s URL-om API-ja i *resource* zahtjev s URL-om API-ja te krajnjom točkom npr. */account*. Zahtjevi se šalju pomoću razreda *AccountInformationServiceTPP*. Ako zaštite od ovog napada ne postoje korisnik bi mogao dodati informacije ili učiniti promijene na računu kojima će napadač poslije moći pristupiti. Kod implementacije u programskom alatu mogao bi se dodati zahtjev pomoću razreda *Requester* s definiranom *PUT* metodom zahtjeva, URL-om API-ja te krajnjom točkom koja se želi promijeniti, npr. *account/credit_cards*. Putem metode *setQueryParameteres* mogu se definirati nove vrijednosti koje se ubacuju. Zahtjev se šalje kada se zatraži promjena. U alatu to se obavlja metodom *sendRequest*.

Napad ubacivanjem pristupnog tokena pretpostavlja da napadač u posjedu ima pristupni token ukraden od nekog korisnika te autorizacijski poslužitelj pod vlastitom kontrolom. TPP će sve zahtjeve slati prema autorizacijskom poslužitelju napadača. U programskom alatu redom se kreiraju *token* zahtjev s URL-om API-ja napadača te *resource* zahtjev s URL-om API-ja i krajnjom točkom npr. */accounts*. Zahtjevi se kreiraju uz pomoć razreda *PSD2Requester* te šalju pomoću razreda *AccountInformationServiceTPP*. Ako zaštita od ovog napada nije implementirana, napadač će vratiti prethodno ukradeni pristupni token te preko TPP-a pristupiti resursima korisnika.

Kod *ID Token Replay Attack* autentifikacija korisnika se izvodi na autorizacijskom poslužitelju banke, no dohvaćanje tokena se izvodi na autorizacijskom poslužitelju pod kontrolom napadača. Pomoću razreda *ConsentRequester* definirao bi se *consent* zahtjev s URL-om API-ja banke, a pomoću razreda *PSD2Requester* definirao bi se *status* zahtjev isto tako s URL-om API-ja banke. Korištenjem razreda *PSD2Requester* kreiraju se *token* zahtjev s URL-om API-ja napadača te *resource* zahtjev s URL-om API-ja napadača te krajnjom točkom npr. */accounts*. Zahtjevi se šalju pomoću razreda *AccountInformationServiceTPP*. Ako ne postoje adekvatne zaštite, napadač će vratiti prethodno ukradeni pristupni token te preko TPP-a pristupiti resursima korisnika.

4.2. Implementacija napada vezanih uz API

Ovi će napadi biti implemetirani korištenjem razreda *Requester*. Putem programskog alata biti će simulirano slanje zahtjeva prema API-ju banke ili od strane napadača ili same banke. Ove je napada moguće implementirati da se šalju i od strane TPP-a no to bi podrazumijevalo maliciozan TPP ili TPP pod kontrolom napadača. Objašnjenja

svih napada nalazi se u poglavlju 3.2.

Kod implementacije *Credential stuffing*, odnosno slomljene autentifikacije korisnika, u razredu *Requester* može se definirati korištenje npr. *POST* zahtjeva, parametar *base_url* će biti URL API-ja koji se koristi te krajnja točka autorizacije npr. */basic_auth*. Metodom *setRequestHeader* dodat će zaglavlje naziva *Authorization* s vrijednošću za određeno korisničko ime i lozinku. Zahtjev za autentifikaciju šalje se kada korisnik/napadač unese korisničko ime i lozinku. Metoda *sendRequest* šalje definirani zahtjev. Ako ne postoji zaštita od *Credential stuffing* napada, napad se simulira daljnjim kreiranjem i slanjem zahtjeva samo s drugim korisničkim imenima i lozinkama.

Manjak validacije unosa korisnika i napadi ubacivanjem, konkretno napad ubacivanjem SQL izraza, može se implementirati na sljedeći način. Korištenjem razreda *Requester* definirati će se parametri kao i u prethodnom napadu. Koristit će se *POST* zahtjev, s URL-om API-ja te krajnjom točkom za autentifikaciju. Isto tako, dodati će se zaglavlje *Authorization* korištenjem metode *setRequestHeader* koji će u ovom slučaju sadržavati korisničko ime te lozinku koja će sadržavati SQL izraz, npr. *some_password' OR 1=1*. Zahtjev se šalje metodom *sendRequest*. Ako određena zaštita od napada ubacivanjem SQL izraza nije implementirana, napadač se na ovaj način može prijaviti bez valjanog korisničkog imena i lozinke.

Slomljena autorizacija na razini objekta može se simulirati definiranjem *GET* zahtjeva, u razredu *Requester*, uz URL API-ja te krajnju točku npr. */accounts*. Zahtjev se šalje kada korisnik/napadač želi pogledati detalje svog računa. Putem metode *setQueryParameter* kao parametar će se postaviti *account_id* s vrijednošću identifikatora za račun koji je u vlasništvu korisnika/napadača. Zahtjevi se izvršavaju korištenjem metode *sendRequest*. Naime, ako ne postoji zaštita za slomljenu autorizaciju na razini objekta, napadač može pristupiti računu drugog korisnika. U alatu, to bi bila promjena vrijednosti parametra *account_id*.

Manjak limitiranja pokušaja i resursa može se implemetirani putem razreda *Requester* s definiranim *GET* zahtjevom, URL-om API-ja te krajnjom točkom koja će dohvaćati provedena plaćanja npr. */payments*. Metodom *setQueryParameter* postaviti će se parametri *limit* koji ograničava broj resursa koji se vraćaju te *user_id* kao identifikator korisnika čijim se resursima pristupa. Zahtjev se šalje metodom *sendRequest*. Ako zaštita od limitiranja broja resursa koji se vraćaju, u ovom slučaju svih provedenih plaćanja, nije implementirana moguće je da dođe do zasićenja sustava. Pomoću alata moguće je kreirati isti zahtjev samo promijeniti parametar *limit* na puno veću vrijed-

nost od pretpostavljene, npr. sa 100 na 100,000.

Kod implementacije slomljene autorizacije na razini funkcija putem razreda *Requester* definira se *GET* zahtjev, URL API-ja koji se koristi te krajnja točka npr. */user/my_financial_info*. Zahtjev se šalje kada korisnik/napadač želi pristupiti tim informacijama. Za slanje zahtjeva koristi se metoda *sendRequest*. Kada zaštite od slomljene autorizacije na razini funkcija nebi bile dobro implementirane, napadač bi mogao pristupiti informacijama svih korisnika čemu bi inače mogao pristupiti samo administrator. U alatu je to moguće simulirati promjenom krajnje točke na npr. *admin/users/all_info*.

Kod masovnog dodjeljivanja u razredu *Requester* definirati će se *PUT* kao metoda zahtjeva, URL API-ja te krajnja točka npr. */users/me*. Zahtjev će se poslati kada korisnik/napadač želi promijeniti npr. korisničko ime. U zahtjev se metodom *setQueryParameter* dodaje parametar *user_name* s novim korisničkim imenom. Sljedeći zahtjev definira se jednako kao i prvi, samo je metoda zahtjeva u ovom slučaju postavljena na *GET*. U odgovoru na zahtjev osim korisničkog imena bio bi prisutan i parametar *credit_balance* s određenom vrijednošću. Ako je API korisni masivno dodjeljivanje bez određenih mjera zaštite napadač može poslati novi zahtjev uključujući parametar *credit_balance* te ga postaviti na željenu vrijednost i time dodati sredstva na svoj račun. Zahtjev bi u alatu izgledao isto kao prvi *PUT* zahtjev, samo bi se metodom *setQueryParametar* dodao parametar *credit_balance* s određenom vrijednošću. Svi se zahtjevi šalju metodom *sendRequest*.

Neispravno upravljanje resursima pretpostavlja postojanje stare verzije API-ja koja je nezaštićena, ali i dalje povezana na bazu podataka. U programskom alatu korištenjem razreda *Requester* može se definirati *GET* kao metoda zahtjeva, URL stare verzije API-ja, npr. *api.bank.com/v1*, te krajnja točka */accounts*. Zahtjev bi se slao metodom *sendRequest*.

5. Zaključak

Napadi na API banke mogu uzrokovati velike posljedice. Napadači mogu doći u posjed resursa korisnika i privatnih informacija vezanih uz korisnika. Na taj način korisnik može nepovratno izgubiti sredstva ili anonimnost. Postoje mnogi načini kako napadač može izvesti napade na API-je banaka, pogotovo pojavom PSD2 direktive te se mora paziti na mnogo različitih faktora kada se govori o zaštiti od tih napada. Iako je jedan od fokusa direktive na zaštiti i snažnoj autentifikaciji korisnika, u ovom je radu prikazano kako se uz lošu ili nedovoljnu implementaciju zaštita OAuth2 toka te samih API-ja ranjivosti lako mogu iskoristiti. Već postoje mnogi poznati napadi i mnogi poznati scenariji napada te razrađeni načini zaštite od njih, no daljnji se napredak uvijek mora ostvarivati. Moguća je pojava novih vrsta napada, napada koji nisu razmatrani pri osiguranju te zato kod razvijanja API-ja za banke te mehanizama zaštite moraju se proširiti vidici te sigurnost staviti na početak liste prioriteta.

6. Literatura

[1] Dan Barahona. Drilling down into excessive data exposure: How to protect your apis sensitive data, April 2022. URL https://www.apisec.ai/blog/excessive-data-exposure.

[2] Dan Barahona. How improper assets management can leave your apis vulnerable to attacks, 2022. URL https://www.apisec.ai/blog/improper-assets-management.

[3] Brian Campbell, John Bradley, Nat Sakimura, i Torsten Lodderstedt. OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens. RFC 8705, Veljača 2020. URL https://www.rfc-editor.org/info/rfc8705.

[4] Dragos-Marian Chivulescu. Balanced, as all things should be: Psd2 and cybersecurity risks. Magistarski rad, University of Twente, 2021.

[5] Antonia Din. What is broken object level authorization (bola) and how can affect you?, Ožujak 2021. URL https://heimdalsecurity.com/blog/what-is-broken-object-level-authorization-bola/.

[6] Open Banking Europe. Open banking europe: eidas qualified certificates under psd2 frequently asked questions. Siječanj 2021. URL https://www.openbankingeurope.eu/media/1940/obe-eidas-qualified-certificates-faq.pdf.

[7] Daniel Fett, Ralf Küsters, i Guido Schmitz. A comprehensive formal security analysis of oauth 2.0. Siječanj 2016.

[8] Daniel Fett, Pedram Hosseyni, i Ralf Küsters. An extensive formal security analysis of the openid financial-grade api. U *2019 IEEE Symposium on Security and Privacy (SP)*, stranice 453–471, 2019. doi: 10.1109/SP.2019.00067.

[9] Roy T. Fielding i Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231, Lipanj 2014. URL https://www.rfc-editor.org/info/rfc7231.

[10] ERSTE Group. URL https://developers.erstegroup.com/docs/tutorials/bank.ebc.

[11] The Berlin Group. Nextgenpsd2 xs2a framework implementation guidelines. Rujan 2021. URL https://www.berlin-group.org/_files/ugd/c2914b_e69952d3a2e247cb8ce1bc767b2946fa.pdf.

[12] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, Listopad 2012. URL https://www.rfc-editor.org/info/rfc6749.

[13] Wallarm Inc. Owasp api security top 10, 2021. URL https://www.wallarm.com/whats-wallarm-learning-center-api-security-owasp.

[14] Tarik Karamehmedović. Izrada klijentskog dijela psd2 sučelja. Magistarski rad, University of Zagreb, Faculty of Electrical Engineering and Computing, Lipanj 2022.

[15] kingthorin. Sql injection. URL https://owasp.org/www-community/attacks/SQL_Injection.

[16] KirstenS. Cross site request forgery (csrf). URL https://owasp.org/www-community/attacks/csrf.

[17] KnowBe4. What is phishing? URL https://www.phishing.org/what-is-phishing.

[18] Torsten Lodderstedt, Mark McGloin, i Phil Hunt. OAuth 2.0 Threat Model and Security Considerations. RFC 6819, Siječanj 2013. URL https://www.rfc-editor.org/info/rfc6819.

[19] Torsten Lodderstedt, John Bradley, Andrey Labunets, i Daniel Fett. OAuth 2.0 Security Best Current Practice. Internet-Draft draft-ietf-oauth-security-topics-19, Internet Engineering Task Force, Prosinac 2021. URL https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics-19. Work in Progress.

[20] mozilla.org. 307 temporary redirect. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/307.

[21] OWASP. Owasp top ten, 2021. URL https://owasp.org/www-project-top-ten/.

[22] Diego Poza. What is credential stuffing? how to prevent credential stuffing attacks, Svibanj 2021. URL https://auth0.com/blog/what-is-credential-stuffing/.

[23] Ryan Tate. Apple's worst security breach: 114,000 ipad owners exposed, 2021. URL https://www.gawker.com/5559346/apples-worst-security-breach-114000-ipad-owners-exposed.

[24] Vaadata. Api security, vulnerabilities and common attacks, Svibanj 2022. URL https://www.vaadata.com/blog/how-to-strengthen-the-security-of-your-apis-to-counter-the-most-common-attacks.

[25] Pieter TJ Wolters i Bart PF Jacobs. The security of access to accounts under the psd2. *Computer law & security review*, 35(1):29–41, 2019.

Napadi na PSD2 API

Sažetak

PSD2 direktiva Europske Unije izdana je u svrhu modernizacije i unapređenja bankovne industrije i usluga koje ona pruža. Banke dopuštaju TPP-ima dopuštaju pristup svojim API-ima čime obavljanje usluga korisnicima postaje pristupačnije. Na taj način otvaraju se novi napadački vektori. Veliki se fokus stavlja na sigurnost korisnika i njegovih resursa. U ovom radu prikazani su mogući napadi na API-je banaka. Osim napada koji iskorištavaju nedostatke implementacija na samim API-ima, fokus je bio na OAuth2 protokolu koji se koristi kao jedan od glavnih načina zaštite i autentifikacije korisnika. Ako se sigurnosti ne preda posebna važnost te ako zaštitni mehanizmi nisu ispravno imeplementirni ili nedostaju, korisnici banaka te same banke mogu pretrpjeti loše posljedice.

Ključne riječi: PSD2, banka, API, napadi, OAuth2, TPP, korisnik, sigurnost

Attacks na PSD2 API

Abstract

The PSD2 Directive of the European Union was issued for the purpose of modernizing and improving the banking industry and the services it provides. Banks allow TPPs to access their APIs making services to customers more accessible. In that way, new attack vectors are present. Great emphasis is placed on the security of the user and his resources. This paper presents possible attacks on bank APIs. In addition to attacks that take advantage of implementation shortcomings on the APIs themselves, the focus has been on the OAuth2 protocol, which is used as one of the main ways to protect and authenticate users. If security is not given special importance and if safeguards are not properly implemented or lacking, bank users and banks themselves can suffer bad consequences.

Keywords: PSD2, bank, API, attacks, OAuth2, TPP, user, security