Sigurnost aplikacijskog programskog sučelja

Salaj, Nina

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Nina Salaj

SIGURNOST APLIKACIJSKOG PROGRAMSKOG SUČELJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Andrej Dujella

Zagreb, studeni 2024.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Mojoj obitelji koja mi je najveća podrška, te svima koji su sa mnom dijelili svoje znanje, a posebno mentoru čije su smjernice i podrška uvelike pridonijeli nastanku ovog rada.

Sadržaj

Uvod

Aplikacijska programska sučelja (API) predstavljaju ključnu ulogu u komunikaciji različitih softverskih komponenti. Koriste se u raznim industrijama, uključujući financijski sektor, e-trgovinu i zdravstvene usluge, pri čemu je sigurnost posebno važna prilikom obrade osjetljivih transakcija i osobnih podataka. S porastom broja sigurnosnih napada na API-je, postalo je očito da neadekvatna zaštita može uzrokovati ozbiljne financijske gubitke i pravne posljedice, što naglašava važnost implementacije učinkovitih sigurnosnih mjera. Cilj ovog diplomskog rada je istražiti ključne mehanizme zaštite te analizirati sigurnosne izazove i najbolje prakse.

Prvo poglavlje istražuje ključne aspekte sigurnosti API-ja. Osnovni sigurnosni mehanizmi, poput autentikacije i autorizacije, neophodni su za osiguravanje sigurnog pristupa resursima. Međutim, kako API-ji postaju sve češće meta napada, neophodno je i razumijevanje najčešćih napada na API-je kako bi se usvojile najbolje prakse za zaštitu. Literatura korištena za ovo poglavlje uključuje API Security in Action [12].

U drugom poglavlju obrađene su kriptografske tehnike koje čine temelj zaštite podataka u komunikaciji. Detaljno su opisani simetrični i asimetrični kriptosustavi te kako oni omogućuju povjerljivost, integritet i neporecivost u prijenosu podataka. Sigurnosni protokol TLS, koji počiva na kombinaciji simetrične i asimetrične kriptografije, ilustrira kako se ove tehnike koriste za siguran prijenos podataka. Na ovim temeljima izgrađeni su napredniji sigurnosni mehanizmi poput autentikacije, autorizacije i tehnika dokazivanja posjedovanja ključeva koje dodatno jačaju sigurnost API-ja. Korištena literatura temelji se na materijalima iz kolegija Kriptografija i sigurnost mreža [5].

Treće poglavlje, pod nazivom *Protokoli za upravljanje identitetom*, fokusira se na mehanizme za autentikaciju i autorizaciju korisnika bez potrebe za izravnim dijeljenjem vjerodajnica. Moderni protokoli, poput OAuth 2.0, omogućuju sigurno delegiranje pristupa korisničkim podacima i resursima, čime se smanjuju rizici od krađe identiteta ili drugih sigurnosnih propusta. Osnovna literatura za ovo poglavlje uključuje specifikacije poput RFC 6749 [7], RFC 7517 [8], RFC 7519 [9].

Posljednje poglavlje istražuje tehnike dokazivanja posjedovanja kriptografskih ključeva koje igraju ključnu ulogu u zaštiti API-ja jer osiguravaju da se pristup resursima može odobriti samo onim entitetima koji posjeduju privatni ključ povezan s javnim ključem

korištenim prilikom autentikacije. Dokazivanje posjedovanja dodatno osnažuje sigurnost jer sprječava napade poput krađe pristupnih tokena – čak i ako napadač dođe u posjed tokena, neće moći koristiti resurse bez valjanog privatnog ključa. To pruža dodatni sloj zaštite u okruženjima gdje je sigurnost od presudne važnosti. Literatura korištena za ovo poglavlje uključuje stručne smjernice i protokole, poput RFC 8705 [1], RFC 9449 [6].

Diplomski rad napravljen je u sklopu aktivnosti Projekta PK.1.1.02.0004 - Znanstveni centar izvrsnosti za kvantne i kompleksne sustave te reprezentacije Liejevih algebri.

Poglavlje 1

Ključni aspekti sigurnosti API-ja

Za postizanje sigurnosti API-ja važno je razumjeti način na koji API funkcionira. Jednako je ključno prepoznati koji su sigurnosni mehanizmi potrebni za njegovu zaštitu i potencijalne napade kojima može biti izložen.

1.1 Osnovni pojmovi

Aplikacijsko programsko sučelje (eng. application programming interface) API, odnosi se na skup pravila i protokola koji omogućuju različitim softverskim komponentama međusobnu komunikaciju, dijeljenje podataka i funkcionalnosti. U kontekstu API-ja, ”aplikacija” označava bilo koji softver s određenom funkcijom, dok se ”sučelje” može smatrati ugovorom o usluzi između aplikacija. Taj ugovor definira način na koji te aplikacije međusobno komuniciraju koristeći zahtjeve (eng. request) i odgovore (eng. response). Kada jedan program koristi funkcionalnosti drugog, prvi se naziva klijentom, a drugi pružateljem usluge. Ovisno o namjeni, postoje unutarnji API-ji, koji služe unutar organizacije za povezivanje različitih komponenti i sustava, i vanjski API-ji, koji omogućuju komunikaciju s vanjskim sustavima. Dobro dizajnirana aplikacija trebala bi imati unutarnje API-je koji pomažu u organizaciji koda i čine njegove komponente ponovo upotrebljivima.

API omogućava skrivanje složenosti implementacije, organizira kod i omogućuje povezivanje različitih sustava i usluga na standardiziran način. Standardizirani API protokoli ključni su za interoperabilnost sustava, omogućujući im razmjenu podataka i komunikaciju, neovisno o njihovoj unutarnjoj arhitekturi. Na taj način, različiti softverski sustavi mogu nesmetano surađivati, dok svaki od njih zadržava vlastitu kompleksnost iza sučelja.

Dobar dizajn API-ja omogućuje jednostavno održavanje i proširivost softverskih komponenti. Komponente trebaju biti oblikovane tako da funkcioniraju kao ”crne kutije”, skrivajući unutarnju složenost od korisnika API-ja. Smanjenjem izloženosti unutarnjih detalja,

API-ji omogućuju da komponente budu ponovno upotrebljive, skalabilne i lakše održive. Time se postiže jednostavnija integracija različitih sustava.

Jedan od primjera primjene API-ja je web preglednik, koji omogućava korisnicima pregledavanje internetskih stranica. U tom slučaju, preglednik koristi API-je za interakciju s operativnim sustavom, registrira događaje poput pomicanja miša i pruža korisnicima mogućnosti putem sučelja. API-ji se koriste i na niskoj razini unutar sustava, kao npr. kod upravljačkih programa, koji koriste API za komunikaciju između hardverskih i softverskih komponenata.

Protokoli su ključni za komunikaciju između API-ja. Protokol definira pravila i formate za razmjenu podataka između dvaju sustava ili aplikacija. Svaki API koristi određeni protokol kako bi osigurao pravilan prijenos podataka, uz pridržavanje standardiziranih pravila. Razni oblici komunikacije između računala ili programa obično se ne uspijevaju realizirati jednim velikim protokolom. Umjesto toga, stvaraju se porodice protokola koji međusobno surađuju i organizirani su u slojeve. Donji sloj neposredno radi s hardverom i podacima u sirovom obliku, srednji slojevi pozivaju usluge nižih slojeva, te tako postaju neovisni o hardverskim detaljima, a gornji sloj poziva usluge srednjih slojeva i bavi se porukama specifičnim za određenu aplikaciju.

API-ji su ključni i u modernim softverskim arhitekturama, kao što su mikroservisi. Ova arhitektura podrazumijeva razbijanje složenih aplikacija na manje, neovisne komponente koje međusobno komuniciraju putem API-ja. Svaka komponenta u mikroservisnoj arhitekturi može se zasebno razvijati, implementirati i skalirati, što omogućuje veću fleksibilnost i otpornost sustava. Međutim, upravljanje velikim brojem mikroservisa zahtijeva *API gateway*, odnosno centraliziranu točku kroz koju prolaze svi zahtjevi.

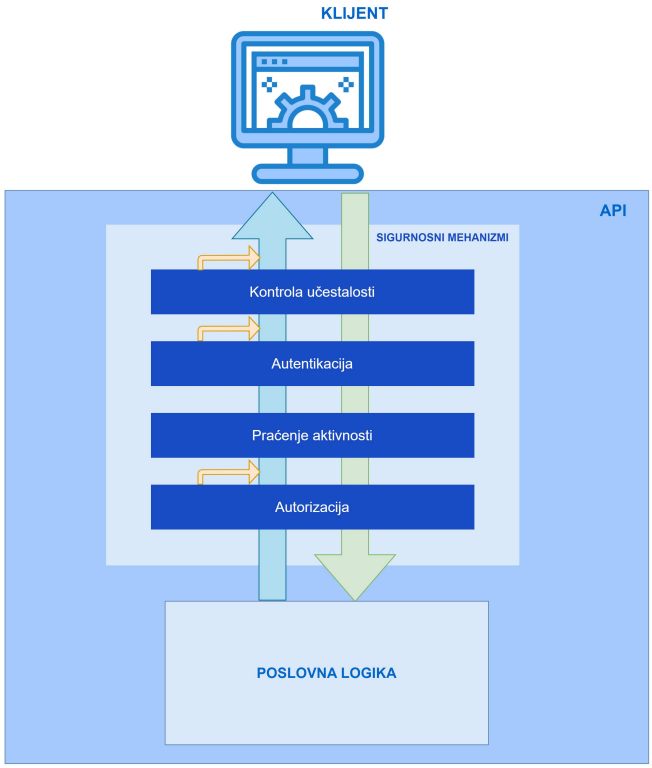
Važno je napomenuti da ne postoji jedinstvena definicija sigurnosti koja se može primijeniti na sve API-je. Sigurnosne mjere ovise o specifičnostima svakog sustava, vrstama podataka koje API obrađuje, kao i o specifičnim prijetnjama koje mogu ugroziti taj sustav. Definirajući sigurnost API-ja, ključno je uzeti u obzir nekoliko aspekata:

- Imovina koju treba zaštititi, uključujući podatke i fizičke resurse.
- Sigurnosni ciljevi koji definiraju što znači ”sigurno” za konkretni API.
- Modeliranje prijetnji kako bi se identificirale i adresirale potencijalne prijetnje specifične za API.

1.2 Sigurnosni mehanizmi

API obrađuje zahtjeve klijenata u ime korisnika. Klijenti mogu biti web preglednici, mobilne aplikacije ili drugi API-ji. API obrađuje zahtjeve prema svojoj unutarnjoj logici i u nekom trenutku vraća odgovor klijentu. Kada obrađuje zahtjev, siguran API primjenjuje

nekoliko standardnih koraka. Zahtjevi i odgovori šifriraju se kako bi se spriječilo otkrivanje informacija. Primjenjuje se ograničenje brzine, te se identificiraju i autenticiraju korisnici i klijenti, a pokušaj pristupa bilježi se u revizijski zapisnik. Na kraju se provjerava treba li korisniku omogućiti izvršenje tog zahtjeva. Ishod zahtjeva također se bilježi u revizijskome zapisniku.



Slika 1.1: Sigurnosni mehanizmi

Autentikacija

Autentikacija je proces kojim se provjerava identitet korisnika, sustava ili uređaja, odnosno provjerava se da entitet jest onaj za kojeg se predstavlja. Identitet se odnosi na skup atributa koji jednoznačno određuju entitet, dok se tvrdnje (eng. claims) odnose na informacije koje entitet pruža kako bi dokazao svoj identitet. Za dokazivanje valjanosti tih tvrdnji koriste se autentikatori, kao na primjer lozinka i sigurnosni token.

U kontekstu API sigurnosti, potrebno je provesti autentikaciju klijenta i poslužitelja kako bi se osiguralo da komunikacija dolazi od legitimnih strana. Utvrđivanje identiteta

štiti resurse od neovlaštenog korištenja, stoga je provjeravanje identiteta korisnika i sustava koji komuniciraju s API-jem prvi korak u njegovoj zaštiti.

Postoje različiti pristupi autentikaciji, uključujući izravnu i posredničku autentikaciju, koji se razlikuju prema načinu na koji se autentikacijske tvrdnje pružaju i provjeravaju. U slučaju izravne autentikacije, entitet koji se želi autenticirati izravno predaje svoje autentikacijske tvrdnje usluzi kojoj želi pristupiti. S druge strane, posrednička autentikacija uključuje treću stranu, poznatu kao pružatelj identiteta (eng. identity provider).

Metode autentikacije mogu se svrstati u tri glavne kategorije poznate kao faktori autentikacije:

- Nešto što entitet zna - tajne informacije poput lozinki ili PIN kodova.
- Nešto što entitet ima - uključuje fizičke objekte poput sigurnosnih ključeva, pametnih kartica ili mobilnih uređaja.
- Nešto što entitet jest - biometrijske karakteristike, poput otiska prsta ili uzorka šarenice oka. Iako biometrijski faktori nude visok stupanj sigurnosti, oni mogu imati visoke stope pogrešaka što ih ponekad čini manje pouzdanim.

Višefaktorska autentikacija (eng. multi-factor authentication), MFA, zahtijeva da korisnici dostave više od jednog tipa dokaza kako bi se autenticirali na sustavu. MFA poboljšava sigurnost kombiniranjem različitih faktora autentikacije. Korištenje više instanci istog faktora (npr. lozinka i PIN) ne smatra se pravom MFA i nudi minimalnu dodatnu sigurnost. Učinkovita MFA zahtijeva neovisne faktore koji nisu podložni istom napadu. Glavna prednost MFA je u zaštiti od uobičajenih ranjivosti povezanih s lošim, ponovno korištenim ili ukradenim lozinkama. MFA značajno poboljšava sigurnost sustava protiv napada poput napada grubom silom, napada punjenjem vjerodajnica i napada raspršivanjem lozinki.

Autorizacija

Autorizacija je proces kojim se utvrđuje koje radnje autenticirani korisnik, sustav ili entitet smije izvoditi unutar definiranih granica sustava. Za razliku od autentikacije, koja potvrđuje identitet korisnika, autorizacija je usmjerena na kontrolu pristupa resursima, što omogućuje zaštitu povjerljivosti, integriteta i dostupnosti tih resursa. Pravilno upravljanje autorizacijom osigurava da korisnici mogu pristupati samo onim resursima i operacijama za koje su ovlašteni, čime se smanjuje mogućnost zlouporabe ili neovlaštenog pristupa.

Postoje dva glavna pristupa kontroli pristupa koji se koriste za API-je:

- Kontrola pristupa temeljena na identitetu: Ovaj se model oslanja na identifikaciju korisnika, a pristup se odobrava ili odbija na temelju korisničkog identiteta i povezanih pravila. U ovom modelu, korisničke dozvole određuju koja sredstva i radnje su mu dostupni. Ovaj pristup usporediv je s načinom na koji operativni sustavi

dodjeljuju prava pristupa korisnicima temeljeno na njihovom identitetu i grupnim ulogama, omogućujući im pristup samo onim datotekama i resursima za koje imaju prava.

- Kontrola pristupa temeljena na sposobnostima: Korisnik dobiva posebne tokene ili ključeve (sposobnosti) koje definiraju njegove dozvole, neovisno o njegovom identitetu.

Primjena modela ovisi o specifičnim zahtjevima sigurnosne politike sustava. Model temeljen na identitetu nudi prednosti poput jednostavnosti implementacije u sustavima s centraliziranim korisničkim bazama, dok model temeljen na sposobnostima omogućuje precizniju i fleksibilniju kontrolu pristupa, osobito u distribuiranim sustavima. U nekim slučajevima, kombinacija ovih pristupa može omogućiti optimizaciju sigurnosnih postavki prema potrebama organizacije.

Jedan od ključnih sigurnosnih principa u upravljanju pristupom je princip najmanjih privilegija, prema kojem korisnik dobiva samo onoliko prava koliko je potrebno za izvršenje zadatka. Ovaj pristup smanjuje mogućnost zloupotrebe sustava i osigurava da korisnik ne može pristupiti podacima ili operacijama koje mu nisu neophodne za obavljanje njegovih aktivnosti.

Enkripcija

Enkripcija, odnosno šifriranje, proces je koji se koristi za osiguravanje tajnosti, integriteta i autentičnosti podataka. Njezin je primarni cilj zaštita informacija tijekom prijenosa ili pohrane. Podaci u prijenosu odnose se na informacije koje se šalju preko mreža, dok su podaci u mirovanju oni koji su pohranjeni na uređajima ili poslužiteljima. Enkripcija omogućava da podaci ostanu nečitljivi bez odgovarajućeg ključa za dekripciju ako budu presretnuti ili im se neovlašteno pristupi. Integritet podataka osigurava se korištenjem funkcija sažetka, koje omogućuju detekciju bilo kakvih neovlaštenih izmjena podataka.

Međutim, enkripcija također ima svoje izazove. Potrebno je pravilno upravljati ključevima za enkripciju i dekripciju jer kompromitiranje ključa može dovesti do ranjivosti podataka. Također, postoji potreba za balansiranjem između snage enkripcije i performansi sustava, jer složeni algoritmi mogu utjecati na brzinu rada aplikacija.

Kontrola učestalosti

Kontrola učestalosti (eng. rate-limiting) ključna je tehnika za upravljanje prometom prema API-ju i održavanje njegove dostupnosti. Cilj ograničavanja brzine je kontrolirati broj zahtjeva koje klijent ili aplikacija može poslati API-ju u određenom vremenskom razdoblju, što sprječava preopterećenje i omogućuje ravnomjernu raspodjelu resursa.

Glavne vrste napada koje se mogu ublažiti primjenom ograničavanja brzine su:

- Distribuirani napadi uskraćivanja usluge (eng. distributed denial-of-service attacks), odnosno DDoS napadi, uključuju slanje ogromne količine zahtjeva kako bi sustav bio preopterećen i postao nedostupan. Ograničavanje brzine može pomoći u prevenciji ovih napada blokiranjem ili usporavanjem zahtjeva s pojedinih IP adresa ili klijenata koji prelaze određeni prag.

- Napadi grubom silom (eng. brute force attacks) uključuju ponavljanje pogađanja prijavnih podataka ili drugih osjetljivih informacija pomoću automatiziranih alata. Ograničavanje brzine može ograničiti broj pokušaja prijave unutar određenog vremenskog okvira, čime se otežava napadačima pronalaženje valjanih vjerodajnica.

Ograničavanje brzine može se postići primjenom različitih tehnika i pristupa, ovisno o specifičnostima API-ja i infrastrukture. Ključne metode uključuju:

- Postavljanje vremenskih ograničenja: Ova metoda uključuje definiranje maksimalnog broja zahtjeva koji klijent može poslati u određenom vremenskom razdoblju. Kada se dostigne limit, novi će zahtjevi biti odbijeni dok ne prođe novo vremensko razdoblje.

- Kvote i tokeni: Korisnicima se dodjeljuju kvote ili tokeni, a novi zahtjevi se blokiraju ili usporavaju nakon što se potroše.

- Dinamički algoritmi: Prilagođavaju ograničenja prema trenutnom opterećenju sustava.

- Usporavanje (eng. throttling): Usporava zahtjeve bez blokiranja korisnika. To se može postići stavljanjem zahtjeva u red za kasniju obradu ili odgovaranjem sa statusnim kodom koji obavještava klijenta da uspori.

Ove se metode implementiraju na razini ravnotežnika opterećenja[1] ili reverse proxy poslužitelja[2] kako bi se zahtjevi filtrirali prije nego što dođu do API poslužitelja, čime se smanjuje opterećenje i štite resursi.

[1]Ravnotežnik opterećenja (eng. load balancer) je komponenta u mrežnoj arhitekturi koja raspoređuje dolazne zahtjeve ili promet među više poslužitelja ili resursa.

[2]Reverse proxy je poslužitelj koji se nalazi ispred web poslužitelja i prosljeđuje zahtjeve klijenata prema tim web poslužiteljima.

Praćenje aktivnosti koristeći revizijske zapise

Revizijski zapis (eng. audit log) predstavlja proces dokumentiranja aktivnosti unutar softverskih sustava. Revizijski zapisi bilježe nastanak događaja, vrijeme nastanka, odgovornog korisnika ili servis, te pogođeni entitet. Omogućavaju praćenje i nadzor aktivnosti unutar sustava, te identificiranje i istraživanje potencijalnih sigurnosnih incidenata. Ako dođe do sigurnosnog proboja ili sumnjivih aktivnosti, analize revizijskih zapisa mogu pomoći u razumijevanju što se dogodilo, tko je bio uključen, te kako je incident mogao nastati. Revizijski zapisi također osiguravaju i usklađenost s industrijskim regulacijama koje zahtijevaju detaljno praćenje i dokumentiranje aktivnosti za potrebe revizije.

Pravilno upravljanje revizijskim zapisima uključuje ne samo njihovo detaljno bilježenje, već i osiguranje da su zaštićeni od manipulacije i da se s njima postupa u skladu s važećim zakonodavstvom i standardima zaštite podataka. Integritet revizijskog zapisa presudan je za njegovu vrijednost. Ako napadač uspije modificirati revizijske zapise, može prikriti svoje tragove i otežati istraživanje incidenta. Kako bi se spriječila manipulacija, pristup revizijskim zapisima trebao bi biti ograničen na mali broj osoba unutar organizacije. Također, primjena tehnologija za zaštitu od manipulacije i šifriranje podataka može dodatno povećati sigurnost. Revizijski zapisi često sadrže osjetljive informacije, poput brojeva bankovnih računa, koje bi trebale biti zaštićene od neovlaštenog pristupa.

1.3 Uobičajene ranjivosti u softveru

Uobičajene ranjivosti u softveru odnose se na slabosti u kodu koje napadači mogu iskoristiti kako bi ugrozili sigurnost sustava. Napad na sustav podrazumijeva namjerno djelovanje kojim entitet pokušava izbjeći sigurnosne usluge i prekršiti sigurnosnu politiku sustava. U ovome potpoglavlju spomenute su neke od najčešćih ranjivosti u softveru.

OWASP popis ranjivosti

OWASP[3] popis ranjivosti često je korišten vodič za razumijevanje najozbiljnijih prijetnji koje se pojavljuju u softverskim aplikacijama. Prijetnje koje OWASP identificira izravna su posljedica propusta u dizajnu i implementaciji sigurnosnih mehanizama.

Neispravna autentikacija korisnika omogućava napadačima da zaobiđu zaštitu, preuzmu korisničke sesije, kompromitiraju autentikacijske tokene ili iskoriste greške u implementaciji kako bi preuzeli identitete drugih korisnika privremeno ili trajno. Napadi na autentikacijske mehanizme često uključuju korištenje lažnih ili ukradenih tokena kako bi

[3]Open Web Application Security Project (OWASP) je pouzdana neprofitna organizacija koja objavljuje analize softverske sigurnosti. Ova je grupa poznata po godišnjim pregledima glavnih ranjivosti web aplikacija. Od 2019. godine također objavljuju popis ranjivosti API sigurnosti.

se pristupilo API-ju, a dodatni rizik predstavlja kompromitacija samog sustava autentikacije ili nenamjerno izlaganje API ključeva. U napade na autentikaciju spadaju i *Cross-Site Request Forgery* (CSRF) napadi koji omogućavaju napadaču da prevari korisnika kako bi nehotice izvršio neovlaštene radnje na serveru na kojem je već autenticiran. Napadač iskorištava činjenicu da preglednik automatski šalje autentikacijske podatke, poput kolačića sesije, kada korisnik pristupi zlonamjernoj web stranici. Tako napadač može, primjerice, promijeniti korisnikove postavke ili izvršiti transakcije, a server vjeruje da je zahtjev došao od legitimnog korisnika. Zaštita od ovih napada obično uključuje korištenje CSRF tokena i sigurnosnih postavki kolačića.

Uobičajene ranjivosti vezane uz autorizaciju uključuju neispravnu provjeru autorizacije na razini objekata, na razini svojstava objekta i na razini funkcija. Neispravna provjera autorizacije na razini objekata omogućuje korisnicima manipuliranje identifikatorima objekata i neovlašten pristup podacima drugih korisnika. Slični problemi javljaju se i na razini svojstava objekata i funkcija, gdje napadači mogu dobiti neovlašten pristup pojedinačnim svojstvima ili funkcijama bez odgovarajućih prava pristupa.

Na OWASP-ovom popisu ranjivosti također se nalazi prekomjerno trošenje resursa. Obrada API zahtjeva zahtijeva resurse poput mrežne propusnosti, procesorske snage, memorije i pohrane, a napadi poput već spomenutog DDoS-a mogu iscrpiti te resurse ili prouzročiti povećanje operativnih troškova.

Još jedna uobičajena ranjivost je krivotvorenje zahtjeva na strani poslužitelja (eng. server-side request forgery) SSRF gdje API dohvaća udaljeni resurs bez provjere URI-ja kojeg je korisnik dostavio. Time napadač može prisiliti aplikaciju da pošalje zlonamjeran zahtjev na nepredviđeno odredište, čak i ako je sustav zaštićen vatrozidom ili VPN-om.

API-ji često imaju složene konfiguracije koje, ako nisu pravilno postavljene, mogu dovesti do sigurnosnih propusta. Redovite revizije i sigurnosna testiranja ključni su za sprječavanje takvih rizika.

Zbog velikog broja pristupnih točaka, potrebno je održavati ažurnu dokumentaciju i evidenciju verzija kako bi se izbjegle zastarjele verzije s ranjivostima. Također, posebna pažnja mora biti posvećena sigurnosnim standardima i redovitim provjerama usluga trećih strana koje API koristi.

Ranjivosti uzrokovane neispravnom provjerom unosa

Neispravna provjera unosa predstavlja značajnu ranjivost koja nastaje kada aplikacija nepravilno provjeri, sanitizira ili obradi podatke koje prima od korisnika. Takvi propusti omogućuju napadačima unošenje zlonamjernog koda ili neispravnih podataka što može ugroziti sustav.

Napadi umetanjem (eng. injection attacks) uključuju umetanje zlonamjernog koda u aplikacije koje dinamički obrađuju korisničke unose. Jedan od najpoznatijih oblika

ovog napada je SQL umetanje, gdje napadač može podmetnuti maliciozne SQL naredbe, što može dovesti do krađe podataka ili potpunog preuzimanja baze podataka. Primjer učinkovite prevencije SQL umetanja uključuje korištenje pripremljenih upita (eng. prepared statements) i ORM (Object-Relational Mapping) alata koji pomažu u sanitizaciji unosa i onemogućuju umetanje zlonamjernog SQL koda.

Do ranjivosti otvorenog preusmjeravanja (eng. open redirect vulnerability) dolazi kada aplikacija prihvati URL koji korisnik unosi bez provjere njegove valjanosti. Napadači mogu iskoristiti ovu ranjivost kako bi preusmjerili korisnike na zlonamjerne web stranice koje mogu prikupiti osjetljive informacije ili pokrenuti druge napade. Kako bi se spriječili napadi otvorenog preusmjeravanja, potrebno je ograničiti korisnički unos i validirati sve URL-ove za preusmjeravanje.

Cross-Site Scripting (XSS) napad je umetanjem zlonamjernih skripti na web stranice koje drugi korisnici posjećuju. Skripte mogu krasti osjetljive informacije, poput sesijskih kolačića, ili preusmjeravati korisnike na zlonamjerne web stranice. Za sprječavanje XSS napada potrebno je pažljivo validirati i kodirati korisnički unos.

Socijalni inženjering

Socijalni inženjering iskorištava ljudske slabosti kako bi napadači manipulirali korisnicima u stjecanju osjetljivih informacija ili izvršavanju neželjenih radnji.

Jedan od najpoznatijih oblika socijalnog inženjeringa je phishing, u kojem napadači lažno predstavljaju legitimne entitete putem e-pošte, lažnih web stranica ili drugih komunikacijskih kanala. U ovim napadima, korisnici su često prevareni da kliknu na zlonamjerni link ili podijele osjetljive podatke, poput lozinki ili brojeva kreditnih kartica. Ključ phishing napada leži u njihovoj ovisnosti o ljudskoj pogrešci. Napadači se oslanjaju na to da će korisnici nepažljivo reagirati, stoga je važno da sigurnosni sustavi ne ovise isključivo o korisničkom ponašanju. Uvođenje sigurnosnih mehanizama poput višefaktorske autentikacije, enkripcije i proaktivnih alata za otkrivanje phishing napada može značajno umanjiti potencijalnu štetu i spriječiti kompromitaciju sustava, čak i kada korisnici naprave pogrešku.

Ransomware napadi postali su sve češći oblik zlonamjernog softvera koji šifrira korisničke podatke i sustave, a zatim zahtijeva otkupninu kako bi se pristup vratio. Ovi napadi često započinju phishingom ili iskorištavanjem ranjivosti u aplikacijama. Organizacije se mogu zaštititi implementacijom jakih enkripcijskih mehanizama i redovitim sigurnosnim kopijama podataka kako bi ublažile posljedice ovakvih napada.

Poglavlje 2

Kriptografija i sigurnosni protokoli

Povijesni pregled kriptografije otkriva kako su se tehnike šifriranja razvijale kroz stoljeća. Klasična kriptografija započela je s osnovnim metodama poput supstitucijskih i transpozicijskih šifri. Supstitucijske šifre zamjenjuju svaki znak poruke drugim znakom prema određenom pravilu, poput Cezarove šifre. Složenije varijante, poput Vigenèreove šifre koja koristi ključnu riječ za varijabilne pomake, te Playfairove šifre koja šifrira parove slova, dodatno su zakomplicirale analizu. Hillova šifra uvodi linearne algebarske transformacije, čime se postiže veća sigurnost.

Za razliku od njih, transpozicijske šifre ne mijenjaju znakove, već samo njihovu poziciju, što otežava čitanje bez ispravnog ključa.

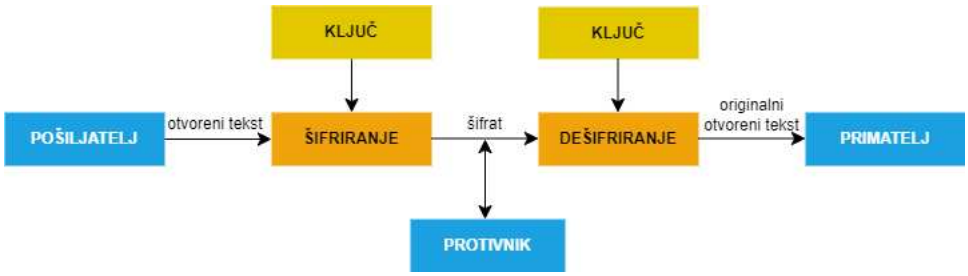
S pojavom računala, razvijeni su moderni simetrični blokovni sustavi poput DES-a, koji je postao standard šifriranja 1970-ih, no zamijenjen je sigurnijim AES-om zbog ranjivosti. Ključni trenutak dogodio se 1976. godine kada su Whitfield Diffie i Martin Hellman objavili rad "New Directions in Cryptography", u kojem su predstavili koncept javnog ključa, čime je omogućeno sigurno dijeljenje ključeva putem nesigurnih komunikacijskih kanala. Ovaj pristup bio je revolucionaran jer je riješio problem razmjene kriptografskih ključeva, temeljni izazov dotadašnjih sustava. RSA kriptosustav dodatno je usavršio kriptografiju javnog ključa, a danas se razvijaju post-kvantni algoritmi kao odgovor na buduće prijetnje kvantnih računala.

U ovome su poglavlju razmotreni osnovni kriptografski koncepti i algoritmi koji su temelj sigurnosnih mjera u aplikacijama kako bi se pružila osnova za razumijevanje kako se kriptografski mehanizmi implementiraju za zaštitu podataka i sustava.

2.1 Osnovni pojmovi

Kriptografija je znanstvena disciplina koja se bavi proučavanjem metoda za slanje poruka u takvom obliku da ih samo onaj kome su namijenjene može pročitati.

U uobičajenom kriptografskom scenariju postoje tri sudionika: pošiljatelj, primatelj i prisluškivač. Pošiljatelj želi poslati legitimnom primatelju poruku, koju nazivamo otvoreni tekst (eng. plaintext). Pretpostavljamo da je komunikacijski kanal između pošiljatelja i primatelja javan, odnosno njegova sigurnost nije dovoljna za potrebe njegovih korisnika. Budući da prisluškivač može doći do podataka u nesigurnom komunikacijskom kanalu, pošiljatelj mora primijeniti funkciju šifriranja kako bi dobio šifrirani tekst koji se zove šifrat (eng. ciphertext). Ključ je tajni podatak koji služi kao ulaz kriptografskog algoritma, te omogućuje proces šifriranja i dešifriranja. Bez ključa, šifrat ostaje nečitljiv stoga se može poslati nesigurnim komunikacijskim kanalom. Legitimni primatelj, koji posjeduje ključ, može primijeniti kriptografski algoritam za dešifriranje i time otkriti otvoreni tekst, odnosno originalnu poruku. Budući da prisluškivač ne poznaje ključ, nije u mogućnosti otkriti sadržaj poruke.



Slika 2.1: Komuniciranje preko nesigurnog komunikacijskog kanala

Spomenute funkcije šifriranja i dešifriranja zovu se šifra ili kriptografski algoritmi. Ovi algoritmi matematičke su funkcije koje se biraju iz određene familije funkcija na temelju ključa, čime se omogućava pretvorba otvorenog teksta u šifrat i obratno. Kriptosustav sastoji se od kriptografskog algoritma, te svih mogućih otvorenih tekstova, šifrata i ključeva. Dakle, imamo sljedeću formalnu definiciju:

Definicija 2.1.1. *Kriptosustav je uređena petorka* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ *za koju vrijedi:*

- $\mathcal{P}$ *je konačan skup svih mogućih osnovnih elementa otvorenog teksta;*
- $\mathcal{C}$ *je konačan skup svih mogućih osnovnih elemenata šifrata;*
- $\mathcal{K}$ *je prostor ključeva, tj. konačan skup svih mogućih ključeva;*
- *Za svaki* $K \in \mathcal{K}$ *postoji funkcija šifriranja* $e_K \in \mathcal{E}$ *i odgovarajuća funkcija dešifriranja* $d_K \in \mathcal{D}$*. Pritom su* $e_K : \mathcal{P} \to \mathcal{C}$ *i* $d_K : \mathcal{C} \to \mathcal{P}$ *funkcije sa svojstvom da je* $d_K(e_K(x)) = x$ *za svaki otvoreni tekst* $x \in \mathcal{P}$*.*

Najpoznatiji cilj kriptosustava je očuvanje privatnosti. Privatni sustavi sprječavaju neovlaštene strane u ekstrakciji informacija iz poruka koje se prenose preko javnih kanala, čime se osigurava da poruku može pročitati samo namijenjeni primatelj.

Osim privatnosti, drugi značajan cilj u kriptografiji je autentikacija kako bi se osiguralo da poruka doista dolazi od izvornog pošiljatelja, te da nije promijenjena tijekom prijenosa.

Cilj pri dizajniranju kriptosustava nije samo osigurati privatnost i autentikaciju, već i postići efikasnost. Idealno, operacije šifriranja i dešifriranja trebaju biti relativno jeftine u smislu resursa i vremena, dok svaka uspješna kriptoanalitička operacija mora biti toliko složena da postaje ekonomski neisplativa. Kriptoanaliza je znanstvena disciplina koja se bavi proučavanjem postupaka za čitanje skrivenih poruka bez poznavanja ključa. Kriptoanaliza zajedno s kriptografijom čini kriptologiju, znanstveno područje koje se bavi sigurnošću komunikacija. Sigurnost većine kriptografskih sustava temelji se na računalnoj složenosti koju kriptoanalitičar mora svladati kako bi otkrio otvoreni tekst bez poznavanja ključa.

Prvi korak u procjeni adekvatnosti kriptografskih sustava je klasificiranje prijetnji kojima su izloženi. Postoje različite vrste napada na kriptografske sustave, uključujući:

- Napad samo na šifrat: Kriptoanalitičar posjeduje samo šifrat od nekoliko poruka šifriranih pomoću istog algoritma. Njegov je zadatak otkriti otvoreni tekst od što više poruka ili u najboljem slučaju otkriti ključ kojim su poruke šifrirane.

- Napad na poznati otvoreni tekst: Kriptoanalitičar posjeduje šifrat neke poruke, ali i njemu odgovarajući otvoreni tekst. Njegov zadatak je otkriti ključ ili neki algoritam za dešifriranje poruka šifriranih s tim ključem.

- Napad na odabrani otvoreni tekst: U ovom napadu, kriptoanalitičar može poslati neograničen broj poruka po vlastitom izboru i pregledati rezultate šifriranja.

- Napad na odabrani šifrat: Kriptoanalitičar je dobio pristup alatu za dešifriranje, pa može odabrati šifrat, te dobiti odgovarajući otvoreni tekst. Ovaj napad je tipičan kod kriptosustava s javnim ključem. Tu je zadatak kriptoanalitičara otkriti ključ za dešifriranje (tajni ključ).

U svim slučajevima pretpostavlja se da protivnik poznaje opći sustav u upotrebi. Prema Kerckhoffovom načelu, sigurnost kriptosustava ne smije ovisiti o tajnosti algoritma, već isključivo o tajnosti kriptografskih ključeva. Ovaj princip omogućuje transparentnost algoritama i njihovu reviziju od strane stručnjaka, čime se jača sigurnost. Zbog toga mnoge komercijalne aplikacije zahtijevaju da opći sustav bude javan i standardiziran, što osigurava povjerenje i široku primjenu kriptografskih rješenja.

Osnovna podjela kriptosustava obzirom na tajnost i javnost ključeva je na simetrične kriptosustave i kriptosustave s javnim ključem.

2.2 Simetrični kriptografski sustavi

Kod simetričnih ili konvencionalnih kriptosustava, ključ za dešifriranje se može izračunati poznavajući ključ za šifriranje i obratno. Najčešće su ti ključevi identični. Sigurnost ovih sustava leži u tajnosti tog ključa, koji se mora pažljivo čuvati kako bi se spriječilo neovlašteno dešifriranje poruka. Upravo zbog toga, simetrični kriptosustavi često se nazivaju kriptosustavima s tajnim ključem.

Simetrični algoritmi često su brži od asimetričnih jer se temelje na jednostavnijim matematičkim operacijama, što ih čini pogodnima za šifriranje velikih količina podataka u stvarnom vremenu.

Glavni problem simetričnih kriptosustava je sigurna razmjena ključeva. Ključ mora biti poznat i pošiljatelju i primatelju poruke, što znači da prije komunikacije mora biti uspostavljen siguran kanal putem kojeg će se ključ razmijeniti. Ako ključ bude kompromitiran, cijela komunikacija je ugrožena. Nedostatak je i skalabilnost, svaki par korisnika mora imati tajni ključ koji dijele. U mreži od n korisnika, potrebno je čak $(n(n-1))/2$ ključeva za međusobnu komunikaciju.

2.3 Asimetrični kriptografski sustavi

Asimetrični kriptosustavi, također poznati kao kriptosustavi s javnim ključem, koriste ključ koji se sastoji od dva dijela: javnog i privatnog ključa pri čemu se različita komponenta para koristi za svaku od dviju suprotnih kriptografskih operacija šifriranja i dešifriranja. Iako su matematički povezani, tajni se ključ ne može (barem ne u nekom razumnom vremenu) izračunati iz javnog ključa. Sigurnost asimetričnih sustava temelji se na složenim matematičkim problemima koji su vrlo teški za rješavanje, poput faktorizacije velikih brojeva ili diskretnog logaritma. Asimetrični kriptosustavi omogućuju korisnicima da sigurno komuniciraju bez potrebe za prethodnim dijeljenjem tajnog ključa.

Asimetrični algoritmi znatno su sporiji od simetričnih zbog složenih matematičkih operacija. Zbog toga se asimetrična kriptografija obično koristi za razmjenu ključeva, dok se stvarna enkripcija podataka često provodi simetričnim algoritmima. Za postizanje iste razine sigurnosti kao kod simetričnih sustava, asimetrični ključevi moraju biti znatno duži. Prednost u odnosu na simetrični kriptosustav je da svaki korisnik ima jedan par ključeva (javni i privatni), bez obzira na broj komunikacijskih partnera.

Asimetrična kriptografija može se koristiti za stvaranje algoritama za šifriranje, digitalni potpis i dogovor o ključu. Šifriranje javnim ključem omogućava tajnost poruke dok digitalni potpis pruža autentičnost poruke.

Algoritmi za šifriranje

Asimetrični algoritmi za šifriranje omogućuju korisnicima sigurno slanje informacija bez prethodnog dogovora o zajedničkom tajnom ključu. Koristi se javni ključ primatelja za šifriranje poruke, tako da samo primatelj, koristeći svoj privatni ključ, može dešifrirati sadržaj poruke. Ovaj princip omogućuje komunikaciju između stranaka koje se ne poznaju unaprijed, osiguravajući povjerljivost podataka.

Digitalni potpis

Digitalni potpis osigurava autentičnost i integritet poruke, omogućujući primatelju da provjeri identitet pošiljatelja i neizmijenjenost poruke. Proces digitalnog potpisivanja koristi privatni ključ pošiljatelja kako bi se stvorio potpis, dok se javni ključ koristi za njegovu provjeru. Time se osigurava da samo vlasnik privatnog ključa može kreirati digitalni potpis, a bilo tko s javnim ključem može provjeriti njegovu valjanost.

Da bi potpisao dokument, korisnik mora posjedovati privatni ključ, koji se koristi u kriptografskom procesu stvaranja digitalnog potpisa. Proces potpisivanja dokumenta započinje tako da se korištenjem hash funkcije generira hash vrijednost dokumenta koja predstavlja jedinstvenu saženu reprezentaciju sadržaja dokumenta. Kriptografske hash funkcije osiguravaju da ako se podatkovni objekt promijeni, digitalni potpis više neće odgovarati. Hash vrijednost dokumenta zatim se šifrira s privatnim ključem potpisnika. Rezultat ove enkripcije digitalni je potpis. Digitalni potpis je jedinstven za sadržaj dokumenta i privatni ključ koji je korišten.

Verifikacija digitalnog potpisa započinje tako da se izračuna hash vrijednost iz primljenog dokumenta koristeći istu hash funkciju koja je korištena prilikom potpisivanja. Digitalni se potpis dešifrira pomoću javnog ključa potpisnika. Dešifriranje vraća originalnu hash vrijednost koja je bila kriptografski enkriptirana prilikom potpisivanja. Na kraju, dešifrirani hash i hash izračunat od primljenog dokumenta se uspoređuju. Ako su obje hash vrijednosti identične, to znači da dokument nije promijenjen i da potpis dolazi od odgovarajućeg potpisnika. Ako se hash vrijednosti razlikuju, to ukazuje na to da je dokument modificiran ili da potpis nije važeći.

Ovaj mehanizam pruža sigurnost da poruka dolazi od stvarnog pošiljatelja, da poruka nije izmijenjena tijekom prijenosa (integritet) i da pošiljatelj ne može kasnije negirati da je poslao poruku (neporecivost).

Dogovor o ključu

Dogovor o ključu je proces u kojem dvije ili više stranaka uspostavljaju zajednički simetrični ključ pomoću kojeg će šifrirati i dešifrirati podatke. Asimetrična kriptografija omogućuje siguran dogovor o ključu čak i preko nesigurnih komunikacijskih kanala. Je-

dan od najpoznatijih algoritama za dogovor o ključu je Diffie-Hellmanov algoritam, koji se temelji na složenosti problema diskretnog logaritma.

Diffie-Hellmanov algoritam omogućuje dvjema stranama da, koristeći svoje privatne i javne ključeve, stvore zajednički tajni ključ, koji je poznat samo njima, a nemoguće ga je rekonstruirati samo na temelju javnih informacija. Ovaj se ključ zatim koristi za daljnju komunikaciju pomoću simetričnih algoritama.

Osnovni koraci protokola su sljedeći:

1. Odabir parametara: Strane se dogovaraju o korištenju velikog prostog broja q i primitivnog korijena α (koji je generator za grupu $\mathbb{Z}_q^*$).

2. Generiranje privatnih ključeva: Svaka strana bira svoj privatni broj. Recimo da korisnik A bira privatni broj X_A, a korisnik B bira X_B, gdje su oba broja manja od q.

3. Izračun javnih ključeva: Korisnici tada izračunavaju svoje javne ključeve koristeći formulu:
$$Y_A = \alpha^{X_A} \bmod q, \text{ za korisnika } A$$
i
$$Y_B = \alpha^{X_B} \bmod q, \text{ za korisnika } B.$$

4. Razmjena javnih ključeva: Korisnici A i B međusobno razmjenjuju svoje javne ključeve preko nesigurnog kanala.

5. Izračun zajedničkog tajnog ključa: Korisnik A koristi Y_B (javni ključ korisnika B) i svoj privatni ključ X_A za izračun zajedničkog ključa:
$$\begin{aligned} K &= Y_B^{X_A} \bmod q \\ &= \left(\alpha^{X_B}\right)^{X_A} \bmod q \\ &= \alpha^{X_A X_B} \bmod q. \end{aligned}$$
Na isti način, korisnik B koristi Y_A i svoj privatni ključ X_B da izračuna isti tajni ključ:
$$\begin{aligned} K &= Y_A^{X_B} \bmod q \\ &= \left(\alpha^{X_A}\right)^{X_B} \bmod q \\ &= \alpha^{X_A X_B} \bmod q. \end{aligned}$$

Zahvaljujući komutativnosti eksponenciranja, obje strane dobivaju isti zajednički ključ, iako su samo razmijenile javne ključeve.

U originalnoj verziji Diffie-Hellmanovog algoritma [4] koristi se multiplikativna grupa $\mathbb{Z}_q^*$ svih ne-nul ostataka modulo q, gdje je q dovoljno veliki prost broj. Međutim, moguće je

koristiti i druge grupe G koje imaju svojstvo da su operacije množenja i potenciranja u njoj jednostavne, dok je logaritmiranje vrlo teško. Također bi trebalo biti moguće generirati slučajne elemente grupe na gotovo uniforman način. Ipak, centralno pitanje jest koliko je težak tzv. problem diskretnog logaritma u grupi G.

Definicija 2.3.1. *Problem diskretnog logaritma: Neka je* $(G, *)$ *konačna grupa,* $g \in G$, $H = \{g^i : i \geq 0\}$ *podgrupa od G generirana s g, te* $h \in H$. *Treba naći najmanji nenegativni cijeli broj x takav da je* $h = g^x$, *gdje je* $g^x = \underbrace{g * g * ... * g}_{x \text{ puta}}$. *Taj broj x se zove diskretni logaritam i označava se s* $\log_g h$.

Sigurnost Diffie-Hellman razmjene temelji se na tome da je vrlo teško izračunati privatne ključeve korisnika na osnovu javnih ključeva zbog problema diskretnog logaritma.

2.4 PKI

Infrastruktura javnih ključeva (eng. public key infrastructure) PKI predstavlja skup tehnologija i protokola koji omogućuju sigurno uspostavljanje komunikacije među korisnicima putem kriptografije. Ova tehnologija čini osnovu za razmjenu osjetljivih informacija na internetu. Korištenjem digitalnih certifikata, PKI osigurava da entiteti mogu vjerovati kako je javni ključ koji koriste u komunikaciji povezan s pravim identitetom subjekta. Na taj način sprječava se lažno predstavljanje, a enkripcija i autentikacija postaju sigurni i pouzdani procesi.

Digitalni certifikati

Ključni element PKI-a digitalni su certifikati, koji povezuju javni ključ s identifikacijskim podacima njegovog vlasnika. Digitalni certifikati omogućuju sigurno dijeljenje javnog ključa jer su potpisani digitalnim potpisom autoriteta kojemu korisnici vjeruju. Na taj način osigurava se povjerenje u autentičnost ključa i identitet subjekta. Certifikat sadrži informacije kao što su identitet vlasnika, javni ključ, izdavatelj certifikata i razdoblje valjanosti. Postoji međunarodni standard X.509 [2], koji definira format i ključne elemente digitalnih certifikata.

Certifikacijska tijela (CA)

Certifikacijska tijela (eng. certificate authorities) CA ovjeriteljske su organizacije odgovorne za izdavanje digitalnih certifikata. CA djeluje kao treća strana kojoj korisnici vjeruju da će provjeriti identitet subjekta prije nego što izda certifikat. Kada CA izda certifikat, ona

digitalno potpisuje taj certifikat, pri čemu jamči da su podaci u njemu točni i povezani s pravim identitetom. Time se uspostavlja povjerenje između svih sudionika u komunikaciji.

Postupak izdavanja certifikata odvija se kroz nekoliko koraka:

- Zahtjev za certifikat (eng. certificate signing request) CSR: Korisnik, koji posjeduje privatni ključ, kreira zahtjev za certifikat koji uključuje njegov javni ključ i informacije o identitetu.
- Provjera identiteta: Ovjeriteljska organizacija provjerava identitet korisnika na temelju podataka iz zahtjeva, a ovisno o vrsti certifikata, može zahtijevati dodatne dokumente.
- Izdavanje certifikata: Nakon uspješne provjere, ovjeriteljska organizacija izdaje digitalni certifikat, potpisujući ga vlastitim privatnim ključem, te ga vraća korisniku.
- Korištenje certifikata: Korisnik sada može koristiti certifikat za razne sigurnosne svrhe, poput šifriranja komunikacije ili provjere identiteta.

CA može izdavati različite vrste certifikata, ovisno o razini provjere identiteta koju obavlja. U hijerarhijskim sustavima, više CA-ova može biti organizirano kako bi se poboljšala sigurnost i fleksibilnost, dok registracijska tijela (eng. registration authorities) RA provode provjeru identiteta i posreduju između korisnika i CA-a.

Upravljanje ključevima i certifikatima

Sustavi za upravljanje ključevima osiguravaju sigurno pohranjivanje privatnih ključeva i upravljanje digitalnim certifikatima. Oni omogućuju izdavanje, obnovu, povlačenje i reviziju certifikata. Kada se privatni ključ kompromitira ili istekne valjanost certifikata, certifikat se povlači, a informacije o povučenim certifikatima bilježe se u listu opozvanih certifikata (eng. Certificate Revocation List) CRL.

Sigurnost PKI sustava

Sigurnost PKI sustava ovisi o nekoliko faktora, uključujući sigurnu pohranu privatnih ključeva i integritet certifikacijskih tijela. Ako je bilo koji od tih elemenata kompromitiran, cijeli sustav povjerenja može biti ugrožen. Zbog toga su sigurnosni protokoli poput SSL/TLS izuzetno važni za zaštitu informacija u PKI sustavu, omogućujući povjerljivost, integritet i autentičnost podataka u elektroničkoj komunikaciji.

2.5 Sažetak poruke

Sažetak poruke, također poznat kao hash vrijednost, ključan je alat u kriptografiji koji se koristi za osiguranje integriteta, odnosno nepromijenjenosti podataka. Hash funkcija, koja generira sažetak poruke, pretvara ulazne podatke proizvoljne duljine u niz fiksne duljine. Formalnu definiciju iznio je Rompay [14] i ona glasi:

Definicija 2.5.1. *Neka je* $n \geq 1$. ***Hash funkcija*** *je funkcija* $h : D \to R$*, gdje je domena* $D = \{0,1\}^*$ *i* $R = \{0,1\}^n$.

U kriptografiji, hash funkcije moraju zadovoljiti dodatne uvjete kako bi bile korisne za zaštitu podataka i autentikaciju:

- Jednosmjernost: Iz sažetka $h(x)$ je praktički nemoguće izračunati izvornu poruku x.
- Otpornost na kolizije: Teško je pronaći dva različita ulaza koja generiraju isti sažetak, odnosno ako je zadana h, gotovo nemoguće je pronaći par (x, y) takav da $h(x) = h(y)$.
- Deterministočnost: Hash funkcija mora uvijek generirati isti izlaz za isti ulaz. To znači da za bilo koju ulaznu vrijednost x, hash funkcija $h(x)$ uvijek vraća isti hash.
- Brza izvedba: Hash funkcija mora biti učinkovita i omogućiti brzo izračunavanje hash vrijednosti za bilo koji ulaz.

Hash funkcije imaju široku primjenu u kriptografiji, a neke od najčešćih primjena uključuju:

- Provjera integriteta podataka: Hash funkcije se često koriste za provjeru da podaci nisu promijenjeni tijekom prijenosa. Na primjer, kada se prenosi datoteka putem interneta, hash vrijednost (poznata kao checksum) generira se za izvornu datoteku. Primatelj može izračunati hash za primljenu datoteku i usporediti ga s poslanim hashom kako bi provjerio je li došlo do izmjena tijekom prijenosa.
- Digitalni potpisi: Hash funkcije se koriste kao dio procesa stvaranja i provjere digitalnih potpisa. Umjesto da se cijela poruka digitalno potpisuje, često se prvo izračuna hash poruke, a zatim se taj hash digitalno potpisuje, što značajno ubrzava proces.
- Pohranjivanje lozinki: Hash funkcije se koriste u autentikacijskim sustavima gdje se pohranjuju samo hash vrijednosti lozinki, a ne stvarne lozinke. Kada korisnik unese lozinku, hash te lozinke se izračunava i uspoređuje s pohranjenom hash vrijednošću. Na taj način, čak i ako napadač dobije pristup bazi podataka lozinki, teško može rekonstruirati izvorne lozinke.

- Hash funkcije igraju ključnu ulogu omogućavajući jednoj strani da dokaže da ima određene informacije, a da ih pri tome ne otkriva.

Iako su kriptografske hash funkcije iznimno korisne, postoje i određeni sigurnosni izazovi. Napadi na kolizije mogu ozbiljno ugroziti sigurnost hash funkcija. To su napadi u kojima se traže dva različita ulaza koja proizvode isti hash. Napadi na MD5 i SHA-1 su uspješno izvedeni, što je rezultiralo napuštanjem ovih algoritama u kriptografskim aplikacijama. Također, napadi poznati kao napadi rođendana koriste se za pronalaženje kolizija iskorištavanjem činjenice da je puno lakše pronaći dvije različite poruke s istim hashom nego predvidjeti hash određene poruke.

Podjela hash funkcija

Hash funkcije mogu se podijeliti u dvije glavne kategorije ovisno o tome koriste li tajni ključ u procesu stvaranja sažetka. Hash funkcije zasnovane na ključu, poznate kao Message Authentication Codes (MACs) omogućuju autentikaciju poruka i očuvanje njihovog integriteta. Hash funkcije mogu se podijeliti i ovisno o algoritmu sažimanja.

Familija SHA funkcija sažetaka

SHA-256 je jedan od najpopularnijih algoritama za generiranje kriptografskih sažetaka i dio je obitelji SHA-2 (Secure Hash Algorithm 2), koju je razvila Nacionalna sigurnosna agencija (NSA). Ovaj algoritam nudi značajno poboljšanu sigurnost u odnosu na svog prethodnika, SHA-1, čija je ranjivost prema kolizijskim napadima dovela do njegovog postupnog povlačenja iz upotrebe. SHA-256 generira fiksnu hash vrijednost duljine 256 bita, bez obzira na veličinu ulaznog podatka.

Jedan od ključnih razloga za široku primjenu SHA-256 jest jednosmjernost, što znači da je gotovo nemoguće izračunati izvorni podatak samo iz hash vrijednosti. Osim toga, čak i najmanja promjena ulaznih podataka rezultira potpuno različitom izlaznom hash vrijednošću, što značajno otežava napade poput pokušaja lažiranja podataka. Zbog svoje sigurnosti, SHA-256 se često koristi u sigurnosnim protokolima poput TLS-a (Transport Layer Security) te u kriptovalutama, primjerice za rudarenje i validaciju transakcija u Bitcoin mreži.

Nadalje, veće izlazne hash vrijednosti, poput onih koje generira SHA-256, zahtijevaju znatno više resursa za izvođenje napada, što ga čini pouzdanim izborom za sigurnosne svrhe gdje je potrebna visoka razina zaštite.

Proces dobivanja hash izlazne vrijednosti sastoji se od sljedećih koraka:

1. Dodavanje bitova: Originalna poruka proširuje se dodatnim bitovima tako da njezina duljina bude 64 bita kraća od najbližeg višekratnika 512. Ako je duljina manja od te vrijednosti, preostali se bitovi popunjavaju odgovarajućim obrascem.

2. Dodavanje duljine poruke: Originalna duljina poruke prikazuje se u 64 bita i dodaje prethodnom rezultatu.

3. Inicijalizacija: Definiraju se početne vrijednosti spremnika, potrebne za izračunavanje sažetka, kao i kreiranje 64 ključa.

4. Rastavljanje poruke: Poruka se dijeli na blokove od 512 bitova, a svaki blok prolazi kroz niz od 64 operacije. Izlaz svake operacije postaje ulaz za sljedeću.

5. Generiranje izlaznog bloka: Kroz ponavljanja, izlazi jednog bloka postaju ulazi za sljedeći sve dok se ne dobije konačni blok od 512 bitova, od kojih prvih 256 predstavlja izlaznu hash vrijednost.

Familija SHA funkcija uključuje i SHA-3, dizajniran kao alternativa u slučaju da SHA-2 postane ranjiv. Dok SHA-2 koristi konstrukciju baziranu na iterativnim operacijama, SHA-3 se oslanja na Keccakov algoritam. Keccak koristi tzv. ”spužvasti model”, što znači da se ulazni podaci se prvo ”upijaju” u unutarnje stanje koje je mnogo veće od same hash vrijednosti, transformiraju nizom operacija, a zatim se ”cijeđenjem” tog unutarnjeg stanja dobiva hash. SHA-3 dolazi u četiri varijante sažetka (SHA3-224, SHA3-256, SHA3-384, SHA3-512), koje proizvode sažetke različitih duljina, te dvije verzije proširivih izlaznih funkcija (SHAKE128 i SHAKE256).

Funkcija	**Veličina izlaza**	**Otpornost na kolizije**	**Jednosmjernost**
SHA-1	160	<80	160
SHA-224	224	112	224
SHA-512/224	224	112	224
SHA-256	256	128	256
SHA-512/256	256	128	256
SHA-384	384	192	384
SHA-512	512	256	512
SHA3-224	224	112	224
SHA3-256	256	128	256
SHA3-384	384	192	384
SHA3-512	512	256	512
SHAKE128	d	min(d/2, 128)	>=min(d, 128)
SHAKE256	d	min(d/2, 256)	>=min(d, 256)

Tablica 2.1: Sigurnosni kapaciteti SHA-1, SHA-2 i SHA-3 funkcija [13]

Tablica 2.1 pruža usporedne informacije o sigurnosnim kapacitetima SHA funkcija. Veličina izlaza označava broj bitova koji se generiraju kao sažetak, a veća veličina obično implicira bolju sigurnost, povećavajući broj mogućih sažetaka i otežavajući napadačima pronalaženje kolizije. Veličine otpornost na kolizije i jednosmjernost iskazane su brojem bitova sigurnosti, odnosno kao broj bitova koji predstavlja veličinu prostora pretraživanja koji napadač mora proći da bi uspješno razbio sigurnost funkcije, koristeći najbolji poznati napad.

Stariji algoritmi poput SHA-1 pokazuju slabiju otpornost na kolizije u usporedbi s novijim varijantama poput SHA-3. Jednosmjernost ukazuje na to koliko je teško pronaći izvorni tekst za određeni sažetak, pri čemu visoki brojevi ukazuju na veću sigurnost. Ova tablica ističe evoluciju SHA funkcija i važnost odabira sigurnih algoritama u kontekstu zaštite podataka.

2.6 TLS protokol

TLS (Transport Layer Security) protokol osigurava sigurnu komunikaciju putem Interneta. Protokol omogućava klijent/server aplikacijama da komuniciraju na način koji je dizajniran da spriječi prisluškivanje, neovlaštene izmjene ili lažiranje poruka.

Ciljevi TLS protokola su prvenstveno uspostavljanje kriptografski sigurne veze između dvije strane. Protokol je također dizajniran s namjerom da bude interoperabilan, omogućujući razvijanje aplikacija koje mogu uspješno razmjenjivati kriptografske parametre bez potrebe za poznavanjem međusobnog koda. TLS je zamišljen kao razvojni okvir koji omogućava integraciju novih metoda javnog ključa bez potrebe za razvojem novog protokola. Na kraju, TLS uzima u obzir relativnu efikasnost, budući da kriptografske operacije mogu biti vrlo zahtjevne po pitanju procesorskih resursa, te stoga uključuje mehanizme za smanjenje broja novih konekcija koje se moraju uspostavljati od početka.

Tok TLS protokola

Prema RFC 5246 [3], TLS protokol sastoji se od dva sloja: TLS Record Protocol i TLS Handshake Protocol. Na najnižem nivou, izgrađen na vrhu pouzdanog transportnog protokola (npr. TCP), nalazi se TLS Record Protocol.

TLS Record Protocol osigurava privatnost i pouzdanost veze koristeći simetričnu kriptografiju za šifriranje podataka. Transport poruka uključuje provjeru integriteta poruka korištenjem MAC-a (Message Authentication Code) s ključem, pri čemu se za MAC izračune koriste sigurni hash algoritmi (npr. SHA-1). Iako TLS Record Protocol može funkcionirati bez MAC-a, to se obično koristi samo kada drugi protokol koristi Record Protocol za pregovaranje sigurnosnih parametara.

TLS Record Protocol se koristi za inkapsulaciju različitih protokola višeg nivoa. Jedan od tih protokola, TLS Handshake Protocol, omogućava serveru i klijentu međusobnu autentikaciju i dogovaranje algoritma šifriranja i kriptografskih ključeva prije nego što aplikacijski protokol prenese ili primi svoj prvi bajt podataka. TLS Handshake Protocol osigurava tri osnovna svojstva sigurnosti veze: autentikaciju identiteta koristeći asimetričnu kriptografiju, sigurnu razmjenu dogovorenih tajni koja je nedostupna prisluškivačima, te pouzdanost pregovora tako da napadač ne može izmijeniti komunikaciju bez otkrivanja.

TLS u sigurnosti API-ja

TLS je postao standard u osiguravanju komunikacija na internetu sredinom 90-ih godina, kada je zamijenio stariji SSL protokol. Danas je TLS neophodan za osiguranje povjerljivosti i integriteta podataka, naročito u kontekstu sigurnosti API-ja, gdje osigurava da osjetljivi podaci koji se prenose između aplikacija ostanu zaštićeni od potencijalnih prijetnji.

Jedan od najčešćih napada na nesigurnu komunikaciju je napad *Man-in-the-Middle* (MitM), oblik aktivnog napada prisluškivanja u kojem napadač presreće i selektivno mijenja komunikacijske podatke kako bi se prikazao kao jedan ili više entiteta uključenih u komunikacijsku vezu. TLS protokol omogućava klijentu i serveru da provjere jesu li njihovi vršnjaci izračunali iste sigurnosne parametre i je li handshake proces prošao bez ometanja od strane napadača. Iako TLS minimalizira rizik od napada smanjivanjem sigurnosne razine na najmanje sigurne metode, viši slojevi moraju biti svjesni svojih sigurnosnih zahtjeva i nikada ne smiju prenositi informacije putem manje sigurnog kanala nego što to zahtijevaju. TLS osigurava da svaka odabrana kriptografska metoda nudi obećanu razinu sigurnosti, što je ključna komponenta u zaštiti API komunikacija od potencijalnih sigurnosnih prijetnji.

TLS autentikacija klijenta

TLS autentikacija klijenta, također poznata kao obostrana TLS (eng. two-way TLS) autentikacija, proširuje uobičajeni TLS protokol tako da oba sudionika u komunikaciji, i klijent i poslužitelj, tijekom TLS handshake procesa razmjenjuju i provjeravaju TLS certifikate. Standardni TLS protokol obično autenticira samo poslužitelja, dok klijent ostaje neautenticiran osim ako se ne primijene dodatne metode autentikacije kao što su lozinke ili tokeni. Međutim, s TLS autentikacijom klijenta, i klijent može biti autenticiran putem certifikata.

Ovaj proces funkcionira tako da, uz standardnu verifikaciju poslužitelja putem certifikata koji je potpisan od strane verifikacijski valjanog certifikacijskog tijela, poslužitelj također prima certifikat od klijenta. Poslužitelj tada validira taj klijentski certifikat pomoću CA treće strane ili vlastitog internog CA sustava. Kako bi se ovo postiglo, poslužitelj mora

korisniku izdati certifikat specifično generiran za njega, s jedinstvenim vrijednostima dodijeljenim subjektu unutar certifikata kako bi se mogao identificirati točno određen korisnik.

Korisnik zatim instalira taj certifikat u svoj preglednik i koristi ga za autentikaciju na web stranici. Ova metoda autentikacije korisna je u okruženjima gdje je potrebna visoka razina sigurnosti, kao što su korporativne mreže ili aplikacije sa strogo kontroliranim pristupom. Kombinacijom TLS autentikacije klijenta i lozinke, moguće je značajno poboljšati sigurnost, eliminirajući rizik od krađe lozinki, dok se istovremeno osigurava da pristup resursima imaju samo ovlašteni korisnici.

Međutim, TLS autentikacija klijenta ima određena ograničenja i nije idealna za široko dostupne javne web stranice s velikim brojem korisnika. Na primjer, za društvene mreže, uvođenje TLS autentikacije klijenta moglo bi stvoriti nepotrebne složenosti i probleme s korisničkim iskustvom. Također, ako se klijent nalazi iza korporativnog proxyja koji provodi SSL/TLS dešifriranje, TLS autentikacija klijenta može biti prekinuta, osim ako proxy ne dozvoli prolaz takve komunikacije.

TLS autentikacija klijenta predstavlja dodatni sloj sigurnosti u zaštiti API komunikacija, posebno u kontekstima gdje je ključna zaštita osjetljivih podataka i identiteta korisnika.

Poglavlje 3

Protokoli za upravljanje identitetom

Identitet u kontekstu digitalnih sustava odnosi se na skup informacija koje jedinstveno identificiraju korisnika, aplikaciju ili uređaj unutar određenog mrežnog okruženja. Te informacije mogu uključivati korisničko ime, lozinku, certifikate, biometrijske podatke, ali i različite atribute poput prava pristupa ili preferencija. Registracijom svih relevantnih svojstava entiteta dobiva se njegov identitet. Upravljanje identitetom podrazumijeva procese i tehnologije koji omogućuju sigurnu autentikaciju i autorizaciju korisnika, osiguravajući da samo ovlašteni subjekti mogu pristupiti osjetljivim podacima i resursima.

S razvojem tehnologija i promjenom načina na koji se koriste digitalni sustavi, upravljanje identitetom postaje sve kompleksniji izazov. Današnji korisnici pristupaju mrežnim resursima s raznih uređaja, često s različitih lokacija i mreža što povećava potrebu za sigurnim prijenosom i pohranjivanjem identiteta. Također, potrebno je minimizirati potrebu za ponovnom autentikacijom zbog jednostavnog korisničkog iskustva.

Kroz povijest, rješenja za upravljanje identitetom evoluirala su kako bi odgovorila na ove rastuće izazove. Svako rješenje uvedeno u određenom periodu donijelo je inovacije za rješavanje problema upravljanja identitetom.

U ranim fazama internetskih aplikacija tijekom 1990-ih, najjednostavniji i najčešće korišteni način autentikacije bio je putem lozinki. Nakon što bi se korisnik uspješno prijavio, server bi generirao sesijski kolačić koji bi se pohranjivao u pregledniku i koristio za daljnje zahtjeve korisnika. Međutim, ovaj pristup imao je ozbiljne sigurnosne nedostatke, kao što su krađa kolačića ili presretanje sesija, što je omogućavalo napadačima da preuzmu korisničke sesije bez potrebe za dodatnom autentikacijom.

Porastom složenosti mrežnih okruženja, posebno u korporativnim sustavima, pojavio se zahtjev za sigurnim prijenosom vjerodajnica između različitih servisa. SOAP (Simple Object Access Protocol) je bio široko korišten protokol za komunikaciju između web servisa. Za osiguranje poruka koje su se razmijenjivale putem SOAP-a koristio se WS-Security, standard organizacije OASIS. Omogućavao je napredne sigurnosne značajke po-

put enkripcije i digitalnih potpisa, ali je bio izazovan za implementaciju zbog kompleksnosti rukovanja kriptografskim ključevima. Istovremeno, SAML (Security Assertion Markup Language) je uveden kao način za sigurno razmjenjivanje autentikacijskih i autorizacijskih podataka između različitih entiteta, omogućujući Single Sign-On (SSO). Ovim je pristupom korisnicima omogućeno da pristupe više aplikacija s jednom prijavom, čime se smanjila potreba za ponovnim unosom lozinki i povećala sigurnost, jer su korisničke vjerodajnice ostale pohranjene samo na pružatelju identiteta (eng. identity provider).

S porastom popularnosti API-ja i mobilnih aplikacija 2007. godine, postalo je potrebno omogućiti trećim stranama siguran pristup resursima korisnika bez dijeljenja njihovih lozinki. OAuth 1.0 predstavljen je kao prvi protokol koji omogućuje sigurno delegiranje prava pristupa. Korisnik je mogao autorizirati aplikaciju da pristupi određenim resursima u njegovo ime, bez potrebe za dijeljenjem svojih vjerodajnica. Iako je ovo bio značajan korak naprijed, OAuth 1.0 imao je složen postupak potpisa zahtjeva, što je otežavalo implementaciju i vodilo prema daljnjim poboljšanjima.

Pojavom OAuth 2.0 i OpenID Connecta između 2010. i 2020. godine, industrijski standard za autorizaciju dodatno se razvijao. OAuth 2.0 predstavljen je kao pojednostavljena i fleksibilnija verzija OAuth-a, eliminirajući potrebu za složenim potpisivanjem zahtjeva. Time je omogućeno aplikacijama da sigurno pristupaju korisničkim podacima putem pristupnih tokena, dok je OpenID Connect proširio OAuth 2.0 kako bi uključio autentikaciju. Ovim su standardima korisnicima omogućene jednostavnije i sigurnije metode autentikacije putem treće strane, što je značajno pojednostavilo proces upravljanja identitetom u modernim sustavima. Bearer tokeni, uvedeni s OAuth 2.0, omogućili su jednostavnu autorizaciju: posjedovanje tokena značilo je pravo na pristup resursima. No, problem s ovim tokenima je bio u tome što su podložni krađi i zloupotrebi. S druge strane, WS-Federation je standardiziran kako bi omogućio interoperabilnost između različitih sigurnosnih domena, koristeći SAML kao osnovni mehanizam za prijenos identiteta između servisa.

Danas su OAuth 2.0 i OpenID Connect među najraširenijim standardima za autorizaciju i autentikaciju u digitalnim sustavima, pružajući temelj za sigurnu razmjenu informacija u širokom spektru aplikacija i usluga. U ovom poglavlju, fokusirat ćemo se na ove tehnologije i detaljno razmotriti ključnu ulogu tokena u upravljanju identitetom.

3.1 Tokeni

Tokeni su ključni mehanizam u modernom upravljanju identitetom, omogućujući prijenos informacija između različitih entiteta. Njihovo razumijevanje ključno je za pravilnu implementaciju i upravljanje sigurnosnim aspektima OAuth 2.0 i OpenID Connecta.

Tokeni su digitalne informacije koje se koriste za autentikaciju i autorizaciju. Token predstavlja niz znakova koji služi kao dokaz da je korisnik ili sustav ovlašten za pristup određenim resursima ili funkcionalnostima. Osim što predstavljaju ovlaštenje za pristup

resursima, tokeni često sadrže tvrdnje o identitetu korisnika, kao što su njegova uloga ili jedinstveni identifikator. Često su kodirani kako bi se osigurao integritet i sigurnost podataka koje sadrže. Niz znakova je izrazito jednostavno prenositi putem komunikacijskih kanala, pogotovo dok je niz zaštićen. Komunikacijskim kanalom prenosi se samo osiguran token umjesto složene strukture podataka.

Tokeni se mogu klasificirati u nekoliko glavnih kategorija, ovisno o njihovoj svrsi i načinu primjene:

- Pristupni tokeni koriste se za pristup resursima u API-ju nakon što je korisnik ili sustav autenticiran, te predstavljaju dozvolu koju je korisnik dao klijentu za pristup zaštićenim resursima na poslužitelju. Obično imaju ograničen vijek trajanja i specifične dozvole, čime osiguravaju kontrolu nad pristupom resursima. Pristupni se tokeni mogu koristiti u različitim formatima za komunikaciju s poslužiteljima resursa, uključujući slanje tokena u HTTP Authorization zaglavlju, kao parametar u URL-u zahtjeva ili u tijelu zahtjeva, kako je definirano u RFC 6750 [10]. Postoji nekoliko tipova pristupnih tokena, među kojima je najčešće korišten **Bearer token**. Bearer tokeni omogućuju svakome tko posjeduje token da pristupi zaštićenim resursima, stoga je važno da se oni pravilno zaštite tijekom prijenosa kako bi se spriječila krađa ili presretanje. Vrijednost Bearer tokena značajna je samo za autorizacijski poslužitelj, dok klijent ne treba interpretirati sadržaj tokena.

- Refresh tokeni (tokeni za osvježavanje) koriste se za dobivanje novih pristupnih tokena kada trenutni pristupni token istekne. Oni omogućuju dugotrajan pristup bez potrebe za ponovnom autentikacijom korisnika što je korisno u dugotrajnim sesijama gdje je potrebno osvježiti pristupni token kako bi se održala sigurnost bez ometanja korisnika. Izdavanje refresh tokena je opcionalno i ovisi o politici autorizacijskog poslužitelja.

- ID tokeni koriste se u scenarijima autentikacije kako bi se prenijele informacije o identitetu korisnika. ID tokeni često su povezani s protokolima kao što je OpenID Connect.

Tokeni se mogu koristiti u različitim formatima, od kojih je najpopularniji JSON Web Token (JWT). JWT-ovi široko su prihvaćeni u industriji zbog svoje sposobnosti da enkodiraju podatke u JSON formatu, što ih čini pogodnim za različite aplikacije i okruženja.

JSON Web Token (JWT)

JSON Web Token (JWT) format je poruke standardiziran kroz RFC 7519 [9] pogodan za prijenos mrežom. Predstavljen je kao string koji sadrži skup tvrdnji u JSON[1] formatu. Tvrdnja predstavlja informaciju o subjektu i prikazuje se kao par ključa i vrijednosti, gdje je ključ string, a vrijednost može biti bilo koji JSON tip podatka.

Postoje dvije vrste JWT-ova: potpisani, koji se nazivaju JWS (JSON Web Signature) i enkriptirani koji se nazivaju JWE (JSON Web Encryption). Ove vrste definiraju način na koji su tvrdnje unutar tokena zaštićene. Potpisani JWT-ovi koriste kriptografski potpis za osiguranje integriteta i autentičnosti podataka. Sadržaj JWS-a može biti javno čitljiv, ali se može provjeriti da podaci nisu izmijenjeni. Enkriptirani JWT-ovi, s druge strane, koriste enkripciju za zaštitu privatnosti podataka, odnosno može ga dešifrirati samo entitet s odgovarajućim ključem.

Struktura JWT-a

JWT se sastoji od tri osnovna dijela: zaglavlja, tijela i potpisa. Ovisno o tome radi li se o JWS-u ili JWE-u, ovi dijelovi mogu biti potpisani, enkriptirani ili oboje, a zatim spojeni u jedan string odvojen točkama ('.').

Zaglavlje (eng. header) sadrži osnovne informacije o tokenu, kao što su tip tokena i kriptografski algoritam korišten za potpisivanje ili enkripciju. Zaglavlje se kodira kao JSON objekt, a u njemu se nalaze metapodaci koji određuju kako će se podaci u tokenu obraditi. Ovo zaglavlje je također poznato kao JOSE (JSON Object Signing and Encryption) Header.

Primjer zaglavlja:

```
1 {
2   "typ": "JWT",
3   "alg": "HS256"
4 }
```

Tijelo (eng. payload) sadrži informacije, odnosno skup tvrdnji, koje token treba prenijeti od jednog entiteta do drugog. Organizirano je u JSON objekt koji se naziva JWT Claims Set. Tvrdnje mogu sadržavati različite informacije, kao što su identitet korisnika, vrijeme isteka tokena, prava pristupa i slično. Nazivi tvrdnji standardizirane su prema IANA registru, a svi nazivi su kratki jer je jedan od glavnih ciljeva JWT-a kompaktnost. Primjer standardnih tvrdnji su:

- "iss" (issuer) - entitet koji je izdao token

[1]JSON (JavaScript Object Notation) je format za razmjenu podataka, zasnovan na JavaScript notaciji objekata. Sastoji se od parova ključeva i vrijednosti (objekti) te poredanih lista vrijednosti (nizovi).

- ”exp” (expiration) - vrijeme isteka tokena
- ”sub” (subject) - identitet korisnika

Ključna stvar koju treba napomenuti je da unutar JWT Claims Seta imena claimova moraju biti jedinstvena; ako postoji duplikat, JWT parseri moraju odbaciti JWT ili uzeti samo zadnji duplikat.

Primjer JWT Claims Set-a:

```
{
  "iss": "joe",
  "exp": 1300819380,
  "http://example.com/is_root": true
}
```

Potpis osigurava integritet tokena, odnosno osigurava da niti zaglavlje niti tijelo nisu izmijenjeni nakon što je token izdan. Potpis se generira tako da se napravi hash vrijednost ulaznih podataka koja se zatim potpiše. Način generiranja potpisa ovisi o tome radi li se o JWS-u ili JWE-u:

- U JWS-u, potpis se dobiva tako da se kodirano zaglavlje i tijelo spoje točkom, a zatim se taj niz potpiše pomoću algoritma specificiranog u zaglavlju. Iako potpis osigurava integritet podataka, ne sprječava druge da pregledavaju sadržaj tokena. Stoga, osjetljive informacije poput korisničkih lozinki nikada ne bi trebale biti prenesene u JWT-u; sve u zaglavlju i payloadu može i treba biti sigurno za javnu upotrebu.
- U JWE-u, tvrdnje se prvo enkriptiraju pomoću algoritma navedenog u zaglavlju, a zatim se cijeli token, uključujući zaglavlje i enkriptirano tijelo, potpisuje.

Konačni JWT sastoji se od Base64url[2] kodiranih zaglavlja, tijela i potpisa, koji su spojeni točkama:

header.payload.signature

Na primjer, jednostavan JWT može izgledati ovako:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJqb2UiLCJleHAiOjEzMDA4MTkz
ODAsImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ.8Be0R9C3Aj9D4nte4Q3YKl
cSXFRgX7zlghlSTYXtM8
```

Token ima zaglavlje i tijelo kakvo je ranije navedeno. Potpis je generiran korištenjem

[2]Base64 je metoda enkodiranja koja binarne podatke pretvara u niz znakova koji sadrži samo alfanumeričke znakove zajedno s ’+’ i ’/’. Base64url je njegova varijanta prilagođena URL-ovima.

HMAC-SHA256 algoritma i sljedećeg tajnog ključa:
`NTNv7j0TuYARvmNMmWXo6fKvM4o6nv/aUi9ryX38ZH+L1bkrnD1ObOQ8JAUmHCBq7Iy7otZcy`
`AagBLHVKvvYaIpmMuxmARQ97jUVG16Jkpkp1wXOPsrF9zwew6TpczyHkHgX5EuLg2MeBuiT/q`
`JACs1J0apruOOJCg/gOtkjB4c=`

JSON Web Key (JWK)

JSON Web Key (JWK) je podatkovna struktura u JSON formatu koja omogućava standardiziran i siguran način reprezentacije kriptografskih ključeva, definirana u RFC 7517 [8]. Ključ je prikazan kao JSON objekt, čiji članovi predstavljaju svojstva ključa, uključujući i njegovu vrijednost.

JWK može predstavljati razne vrste kriptografskih ključeva, uključujući simetrične i asimetrične ključeve. Svaki ključ unutar JWK-a definiran je nizom parametara specifičnih za tip ključa, pri čemu je ključno svojstvo ”kty” (Key Type) koje identificira kriptografsku obitelj algoritama s kojom se ključ koristi, poput ”RSA”.

Parametri unutar JWK objekta moraju biti jedinstveni, a implementacije moraju odbaciti JWK-ove s dupliciranim nazivima članova. Ako naiđu na neprepoznate članove, implementacije ih moraju ignorirati. Parametri JWK-a također uključuju sljedeće:

- ”kty” (Key Type): Ovaj parametar identificira vrstu ključa, npr. ”RSA” ili ”EC”. Mora biti prisutan u svakom JWK objektu.
- ”use” (Public Key Use): Ovaj parametar definira svrhu javnog ključa, bilo za potpisivanje (”sig”) ili enkripciju (”enc”). Njegova upotreba je opcionalna, osim ako aplikacija ne zahtijeva njegovu prisutnost.
- ”key_ops” (Key Operations): Definira operacije za koje je ključ namijenjen, poput potpisivanja, verifikacije, enkripcije ili dekripcije. Vrijednosti su predstavljene kao niz operacija, a kombinacije poput ”sign” i ”verify” ili ”encrypt” i ”decrypt” su dopuštene, dok se ostale kombinacije ne preporučuju zbog sigurnosnih rizika.
- ”alg” (Algorithm): Ovaj parametar identificira algoritam koji se koristi s ključem. Vrijednosti trebaju biti registrirane u IANA registru ili sadržavati ime otporno na kolizije.
- ”kid” (Key ID): Ovaj parametar se koristi za identifikaciju određenog ključa unutar seta ključeva, što je osobito korisno pri rotaciji ključeva.
- ”x5u” (X.509 URL), ”x5c” (X.509 Certificate Chain), ”x5t” (X.509 Certificate Thumbprint): Ovi parametri povezuju JWK s X.509 certifikatom ili njegovim otiskom, omogućujući interoperabilnost s tradicionalnim PKI sustavima.

Validacija tokena

Validacija tokena proces je validacije tvrdnji i potpisa. Kako bi se token validirao, prvo je potrebno preuzeti i parsirati skup ključeva u JWK formatu, koji se koristi za verifikaciju potpisa. Zatim se token dekodira kako bi se pristupilo njegovim dijelovima, uključujući zaglavlje i sadržaj, koji sadrže potrebne informacije za validaciju. Validacija tvrdnji uključuje provjeru tih informacija, poput trajanja tokena i dozvola korisnika. Potpis se provjerava pomoću odgovarajućeg ključa iz JWK skupa ključeva koji odgovara "kid" identifikatoru u zaglavlju tokena. Redoslijed izvršavanja koraka može varirati, ovisno o specifičnim potrebama i implementaciji sustava.

Introspekcija tokena

Tokeni se dijele na referentne i samostalne tokene. Referentni token je proizvoljan niz znakova koji se koristi za pretraživanje podataka na autorizacijskom poslužitelju, dok je samostalni token, poput JWT-a, samodostatan i može biti validiran neovisno od strane autorizacijskog poslužitelja. Iako JWT omogućuje da se informacije nalaze unutar samog tokena, ponekad je korisno koristiti mehanizam introspekcije za provjeru stanja tokena. Introspekcija tokena omogućuje zaštićenom resursu da upita autorizacijski poslužitelj o validnosti i stanju tokena. Protokol introspekcije uključuje slanje HTTP zahtjeva na introspekcijsku pristupnu točku autorizacijskog poslužitelja. Odgovor na ovaj zahtjev daje detaljne informacije o tokenu, uključujući njegov status (aktivno/isteklo/opozvano) i dodatne tvrdnje poput opsega i korisničkih atributa.

Kombinacija JWT-a i introspekcije može pružiti snažan alat za upravljanje pristupom. JWT može sadržavati osnovne informacije o tokenu, dok introspekcija omogućava dodatnu provjeru i povlačenje detaljnijih informacija kada je to potrebno. Ovaj pristup omogućuje fleksibilno upravljanje tokenima i može biti posebno koristan u složenim sustavima koji koriste više autorizacijskih poslužitelja.

3.2 OAuth

OAuth (Open Authorization) standardni je protokol za autorizaciju dizajniran za prijenos ovlaštenja, odnosno omogućavanje aplikacijama trećih strana pristup zaštićenim resursima na poslužitelju u ime krajnjih korisnika, bez potrebe za dijeljenjem njihovih korisničkih akreditiva (eng. credentials). OAuth omogućuje korisnicima da dodijele ograničen pristup svojim resursima trećim stranama na način koji minimizira sigurnosne rizike povezane s dijeljenjem akreditiva, kao što su neautorizirani pristup i složenost povlačenja pristupa.

OAuth 1.0 (i OAuth 1.0a) bio je prvi standard za OAuth i koristio je digitalno potpisivanje svakog zahtjeva u svrhu delegiranja pristupa, dok njegov nasljednik OAuth 2.0

rješava ovaj problem koristeći pristupne tokene koji pojednostavljuju i sigurnije upravljaju pristupom resursima. Iako je svojevremeno bio koristan, OAuth 1.0 više se ne koristi zbog složenosti u implementaciji i sigurnosnih nedostataka koje je OAuth 2.0 riješio moderniziranim i fleksibilnijim pristupom autorizaciji.

Kako bi se olakšala implementacija i smanjili sigurnosni rizici, OAuth 2.0 zamijenio je digitalne potpise pristupnim tokenima (Bearer tokens), koji omogućuju autorizaciju putem sigurnih TLS kanala bez potrebe za potpisivanjem svakog pojedinačnog zahtjeva. Ova promjena omogućila je jednostavniju i bržu integraciju u moderne web aplikacije.

Prema RFC 6749 [7], ključni akteri u OAuth 2.0 protokolu su vlasnik resursa, poslužitelj resursa, klijent i autorizacijski poslužitelj. Vlasnik resursa (često krajnji korisnik) odobrava pristup trećoj strani, dok poslužitelj resursa pohranjuje podatke i odgovara na zahtjeve pomoću pristupnih tokena. Klijent je aplikacija koja traži pristup tim podacima, a autorizacijski poslužitelj izdaje pristupne tokene na temelju odobrenja vlasnika resursa.

OAuth 2.0 uvodi jasnu razliku između autorizacijskog poslužitelja, koji upravlja izdavanjem tokena, i poslužitelja resursa, koji pohranjuje zaštićene podatke. Ova podjela omogućava jasnije granice između faza autorizacije i pristupa resursima te poboljšava sigurnost i upravljanje pristupom.

Važno je napomenuti da su ovi akteri logički koncepti, a ne nužno fizički odvojeni entiteti. U stvarnosti, može se dogoditi da jedan fizički poslužitelj može djelovati kao autorizacijski poslužitelj i poslužitelj resursa. Također, jedan autorizacijski poslužitelj može izdavati pristupne tokene koje različiti poslužitelji resursa prihvaćaju.

Apstraktni tok OAuth 2.0, prikazan na slici 3.1, opisuje interakciju između ovih logičkih uloga i uključuje sljedeće korake:

1. Klijent zahtijeva autorizaciju od vlasnika resursa. Zahtjev za autorizaciju može biti upućen direktno vlasniku resursa, kao što je prikazano, ili posredstvom poslužitelja autorizacije.

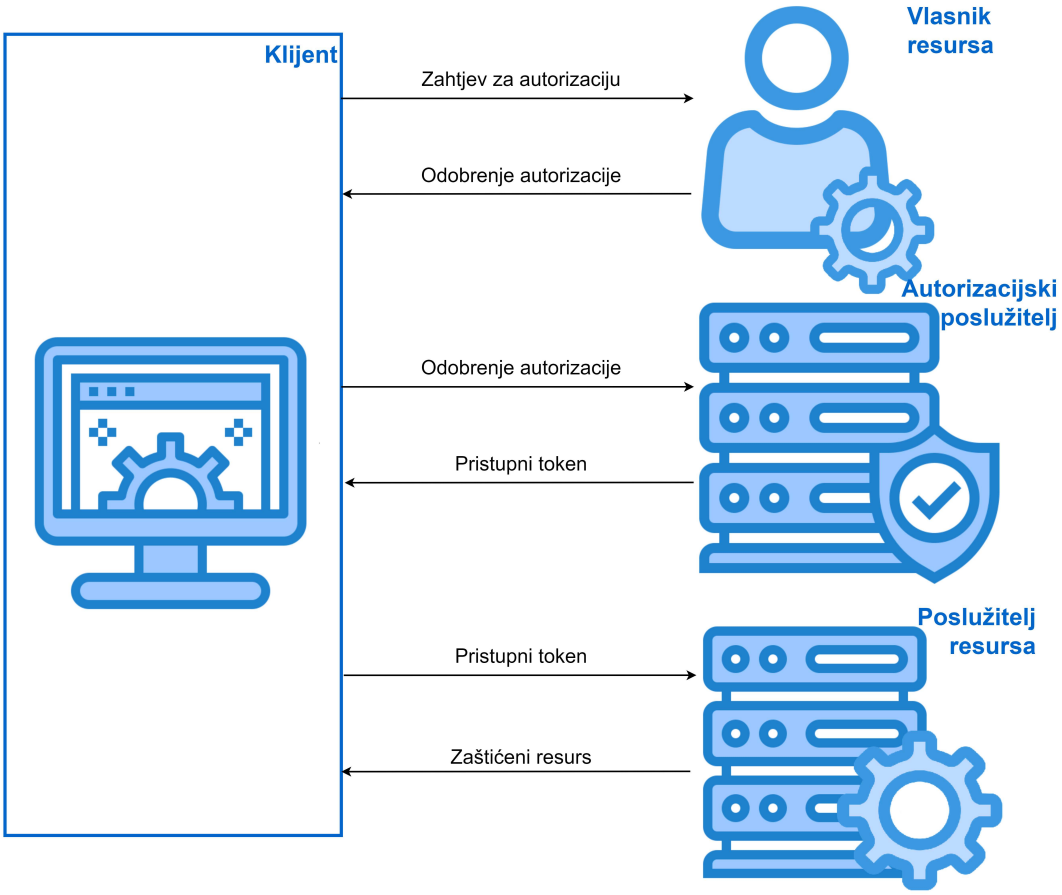
2. Klijent prima odobrenje autorizacije (eng. authorization grant), vjerodajnicu koja predstavlja odobrenje vlasnika resursa.

3. Klijent zahtijeva pristupni token od poslužitelja autorizacije kojem predstavlja odobrenje autorizacije. Autorizacijski poslužitelj autenticira klijenta, provjerava odobrenje autorizacije, te ako je valjano, izdaje pristupni token.

4. Klijent pristupa zaštićenom resursu koristeći dobiveni pristupni token.

OAuth 2.0 definira nekoliko vrsta odobrenja koji određuju kako klijent može dobiti pristup zaštićenim resursima. Četiri osnovna tipa su:

- Autorizacijski kod - korisnik se autenticira na autorizacijskom poslužitelju i dobiva kod koji se zatim koristi za dobivanje pristupnog tokena.



Slika 3.1: Apstraktni tok OAuth 2.0 protokola

- Implicitno odobrenje - token se odmah isporučuje korisniku nakon autorizacije.
- Lozinka vlasnika resursa - klijent izravno koristi korisničke akreditive za dobivanje pristupnog tokena.
- Klijentski akreditivi

Odabir odgovarajućeg tipa odobrenja ovisi o scenariju primjene. Ako aplikacija treba pristupiti API-ju kao korisnik, treba se koristiti autorizacijski kod. Ako aplikacija pristupa vlastitim resursima, klijentski akreditivi su prikladniji. Tipovi poput implicitnog i lozinki vlasnika resursa sada su zastarjeli zbog sigurnosnih rizika.

Poglavlje 4

Dokaz posjedovanja (Proof of Possession)

OAuth protokol često se koristi za autorizaciju, no standardni OAuth Bearer tokeni imaju određena sigurnosna ograničenja, posebno kada se radi o osjetljivim podacima. OAuth Bearer tokeni funkcioniraju poput gotovine – tko god ih posjeduje, može ih koristiti. U idealnom svijetu, gdje je svaki segment sustava potpuno siguran, ovaj pristup bi mogao biti dovoljan. Međutim, u stvarnosti, uvijek postoji rizik od presretanja ili zloupotrebe tokena. Zbog ovih ograničenja, industrija je počela tražiti sigurniji način rukovanja tokenima.

Glavna ideja bila je razviti tip tokena koji bi bio povezan s klijentom koji je inicijalno zatražio taj token. Tako, čak i ako bi netko presreo token, ne bi ga mogao koristiti bez dodatne autentikacije. API bi idealno trebao moći na neki način provjeriti da je pristupni token izdan istom klijentu koji šalje zahtjev API-ju. Postoji nekoliko predloženih rješenja kako bi se to moglo postići, a većina njih opisuje način kriptografske povezivosti pristupnog tokena s nekim tajnim podatkom koji je poznat samo klijentu. Zatim bi klijent koristio isti tajni podatak prilikom poziva API-ja, čime bi API bio u mogućnosti provjeriti da je isti tajni podatak korišten i na autorizacijskom serveru i na API-ju.

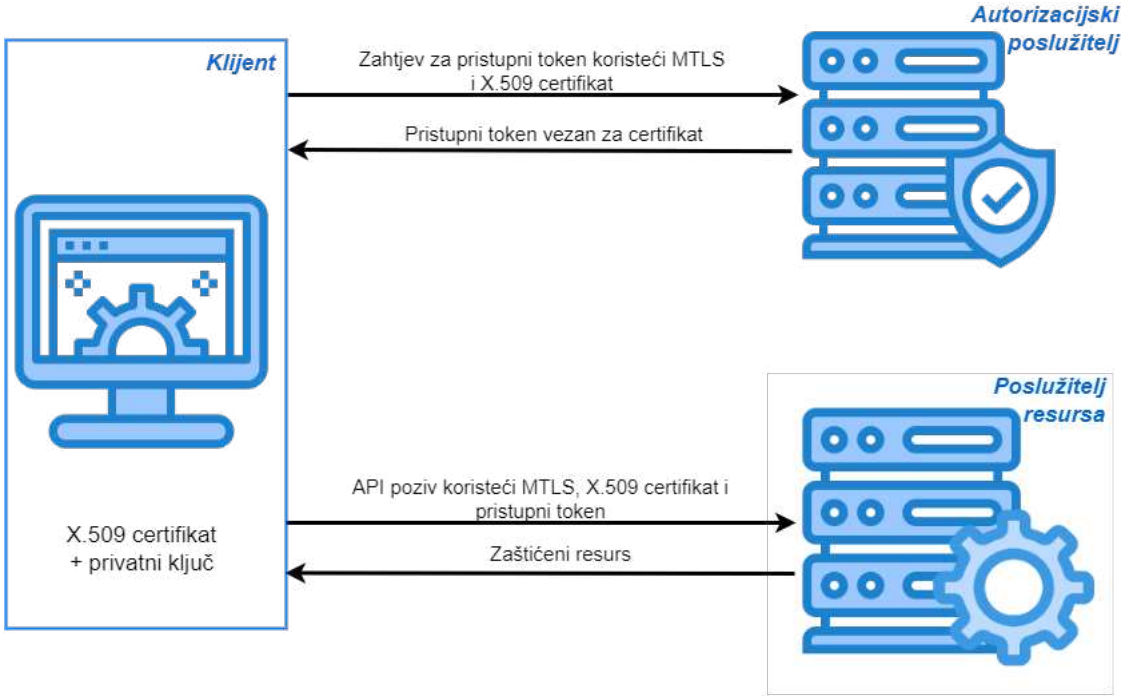
U ovom poglavlju bit će obrađena dva glavna mehanizma za dokaz posjedovanja (eng. Proof of Possession) PoP: uzajamni TLS (eng. mutual TLS) mTLS i demonstrativni dokaz posjedovanja (eng. Demonstrating Proof of Possession) DPoP. Ovi mehanizmi osiguravaju da samo strana koja posjeduje odgovarajući privatni ključ može koristiti izdani token, čime se značajno smanjuje rizik od zloupotrebe.

4.1 Uzajamni TLS

Uzajamni TLS, mTLS je mehanizam koji omogućuje obostranu provjeru autentičnosti između klijenta i poslužitelja, osiguravajući da obje strane mogu potvrditi identitet druge

strane kroz razmjenu certifikata. U kontekstu OAuth protokola, uzajamni TLS koristi se za dodatno osiguranje da se pristupni tokeni vežu uz TLS certifikat klijenta, čime se jamči da samo klijent koji posjeduje odgovarajući privatni ključ može koristiti token za pristup zaštićenim resursima.

mTLS proces u OAuth sustavima



Slika 4.1: Tok MTLS mehanizma

1. Generiranje certifikata: Kada klijentska aplikacija želi dobiti pristupni token od autorizacijskog poslužitelja, generira X.509 certifikat i privatni ključ ili koristi postojeći certifikat.

2. Uspostavljanje mTLS veze: Klijent koristi X.509 certifikat za otvaranje mTLS veze s poslužiteljem tokena. Tijekom TLS rukovanja, klijent šalje svoj certifikat autorizacijskom poslužitelju, a poslužitelj provjerava certifikat i autenticira klijenta.

3. Izdavanje pristupnog tokena: Nakon što poslužitelj tokena uspješno autenticira klijenta putem mTLS-a, izdaje pristupni token. Taj pristupni token sadrži hash vrijednost X.509 certifikata klijenta. Ugradnja hash vrijednosti certifikata unutar pristup-

nog tokena služi kao mehanizam kojim se pristupni token ”veže” uz taj specifični certifikat.

4. Upotreba pristupnog tokena: Klijent sada posjeduje pristupni token koji je sigurno povezan s njegovim X.509 certifikatom. Kada klijent želi pristupiti zaštićenim resursima na API-ju, ponovno koristi isti X.509 certifikat kako bi uspostavio mTLS vezu s API poslužiteljem.

Veza između pristupnih tokena i certifikata

Kada klijent koristi mTLS tijekom spajanja na pristupnu točku za izdavanje tokena, autorizacijski poslužitelj može vezati izdani pristupni token uz certifikat klijenta. Ova veza između tokena i certifikata može se postići na više načina, uključujući:

- Ugradnju hash vrijednosti certifikata unutar samog pristupnog tokena, u ”cnf” tvrdnji:

```
{
  header
}.
{
  "iss": "https://server.example.com",
  "exp": 1493726400,
  "cnf":{
    "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}.
signature
```

- Token introspekcijom, gdje autorizacijski poslužitelj na zahtjev resursnog poslužitelja može potvrditi koji certifikat je povezan s tokenom.
- Ovisno o dogovoru između autorizacijskog poslužitelja i resursnog poslužitelja, mogući su i drugi načini vezivanja certifikata uz pristupni token.

Vezivanje pristupnog tokena uz certifikat klijenta na ovaj način omogućuje da mTLS služi isključivo kao dokaz o posjedovanju ključa tijekom pristupa zaštićenim resursima, neovisno o inicijalnoj autentikaciji klijenta na autorizacijskom poslužitelju.

Korištenje mTLS-a na poslužiteljima resursa

Za pravilnu primjenu mTLS-a, poslužitelji resursa moraju unaprijed biti konfigurirani. Uobičajeno je da resursni poslužitelji koji koriste mTLS zahtijevaju mTLS za sve resurse, ili pak odvajaju resurse na različite hostname-ove i port-ove kako bi jasno razdvojili one koji zahtijevaju mTLS od onih koji to ne čine. Bitno je napomenuti da se sam pristupni token ne može koristiti kao ulaz za odluku hoće li se zahtijevati mTLS, budući da se token razmjenjuje kroz aplikacijski sloj, tek nakon što je TLS rukovanje dovršeno. Tijekom TLS rukovanja, klijent mora predočiti svoj certifikat, a pristupni token postaje dostupan tek nakon tog procesa.

Provjera pristupnih tokena i certifikata na poslužitelju resursa

Kada klijent podnosi zahtjev za pristup zaštićenim resursima, šalje pristupni token kao dio svog zahtjeva. Taj zahtjev mora biti poslan putem mTLS veze, koristeći isti certifikat koji je korišten prilikom spajanja na token pristupnu točku. Resursni poslužitelj pregledava pristupni token i izvlači hash vrijednost X.509 certifikata iz njega. Zatim provjerava TLS sloj veze, traži certifikat koji je korišten za uspostavu mTLS veze, i hashira taj certifikat. Ako su hash vrijednosti certifikata (iz pristupnog tokena i mTLS veze) jednake, potvrđuje da klijent posjeduje privatni ključ povezan s certifikatom i dopušta pristup resursima. Ako vrijednosti nisu jednake, pokušaj pristupa resursu mora biti odbijen s odgovarajućom greškom, prema RFC 6750 [10], uz HTTP statusni kod 401 i kod greške ”invalid_token”. Ovaj proces osigurava da čak i ako napadač ukrade pristupni token, ne bi mogao koristiti taj token za pristup resursima jer ne posjeduje privatni ključ potreban za uspostavu mTLS veze. Napadač bi mogao imati token, ali bez privatnog ključa, ne bi mogao otvoriti TLS kanal, čime bi pristup bio onemogućen.

Prednosti mTLS-a uključuju visoku sigurnost zahvaljujući korištenju postojećih tehnologija poput PKI i X.509 certifikata. Bankarski sektor, s već uspostavljenim PKI sustavima, bio je ključan u promicanju ove tehnologije. Međutim, mTLS ima i nedostatake. Implementacija je složena, zahtijeva postojanje PKI infrastrukture i nije pogodna za sve scenarije, osobito one koji uključuju mobilne uređaje ili široko distribuirane aplikacije. Također, mTLS se suočava s izazovima u prijenosu certifikata kroz mrežne slojeve, posebno kod korištenja reverse proxy servera, što dodatno otežava implementaciju u nekim okruženjima.

4.2 DPoP

Kako bi se prevladali nedostaci mTLS-a i ponudilo rješenje koje je primjenjivo u širem spektru okruženja, razvijen je novi mehanizam nazvan Demonstrating Proof-of-Possession,

DPoP. DPoP predstavlja moderni pristup dokazivanju posjedovanja na aplikacijskoj razini, koji koristi kriptografske metode za vezivanje tokena uz specifičnan HTTP zahtjev, čime se smanjuje mogućnost zloupotrebe presretnutih tokena. Za razliku od mTLS-a, koji se oslanja na postojeću mrežnu infrastrukturu i certifikate, DPoP omogućuje vezanje tokena uz specifični klijentov par ključeva generiran na aplikacijskoj razini. Ovaj pristup pruža veću fleksibilnost i ne zahtijeva posebne mrežne konfiguracije, što ga čini pogodnijim za širok raspon aplikacija, uključujući mobilne i web aplikacije.

DPoP dokazni token

Demonstriranje dokaza o posjedovanju uvodi posebnu vrstu JWT-a koji se koristi za dokazivanje posjedovanja privatnog ključa koji odgovara određenom javnom ključu. Prema RFC 9449 [6], DPoP dokaz se šalje kao posebno zaglavlje u HTTP zahtjevu i uključuje potpis koji pokriva nekoliko ključnih komponenti:

- Podaci HTTP zahtjeva
- Vrijeme kada je DPoP dokaz generiran (nalazi se u ”iat” tvrdnji) - dokazni token bi trebao imati vrlo kratko trajanje, otprilike minutu
- Jedinstveni identifikator koji osigurava da je svaki DPoP dokaz jedinstven (nalazi se u ”jti” tvrdnji)
- Opcionalni nonce, odnosno vrijednost koju poslužitelj može pružiti i koja pruža dodatnu sigurnost
- Hash pristupnog tokena: Ako je pristupni token prisutan u zahtjevu, njegov hash također se uključuje u DPoP dokaz

Važno je uočiti da se ne potpisuje cijeli HTTP zahtjev nego samo njegovi dijelovi: metodu i HTTP URL, koji je krajnja točka na koju se šalje dokazni token. Žrtvuje se dio sigurnosti potpisivanjem samo dijelova zahtjeva, ali postiže se bolja učinkovitost i smanjuje se složenost.

Primjer DPoP dokaznog tokena koji se šalje u zahtjevu za pristupni token:

```
{
  "typ":"dpop+jwt",
  "alg":"ES256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
```

```
  }
}.
{
  "jti":"-BwC3ESc6acc2lTc",
  "htm":"POST",
  "htu":"https://server.example.com/token",
  "iat":1562262616
}.signature
```

Parametar zaglavlja ”typ” postavljen je na *dpop+jwt*. Javni se ključ nalazi u zaglavlju tokena, kao vrijednost ”jwk” parametra. Potpis potvrđuje poznavanje privatnog ključa.

Vezanje DPoP dokaza za pristupni token na autorizacijskom poslužitelju

Autorizacijski poslužitelj izdaje pristupni token koji sadrži informaciju o javnom ključu. Taj pristupni token u tijelu sadrži ”cnf” tvrdnju čija je vrijednost Base64url šifrirani JWK SHA-256 thumbprint javnog ključa (RFC 7638 [11]).

JWK SHA-256 thumbprint javnog ključa predstavlja jedinstveni sažetak javnog ključa koji je definiran u JWK formatu. Izračunava se u nekoliko koraka. Prvo se konstruira JSON objekt koji sadrži samo obavezne članove JWK-a, poput tipa ključa (”kty”), modula (”n”) i eksponenta (”e”) za RSA javni ključ. Zatim se osigurava da je JSON objekt bez razmaka i linijskih prijeloma, te da su članovi poredani leksikografskim redoslijedom. Nakon toga, UTF-8 prikaz tog JSON objekta se kriptografski hashira korištenjem algoritma poput SHA-256, što rezultira JWK thumbprintom. Ovaj thumbprint koristi se kao jedinstveni identifikator javnog ključa bez potrebe za prikazom cijelog ključa, čime se pojednostavljuje njegova upotreba u različitim sigurnosnim protokolima.

Primjer pristupnog tokena vezanog za pristupni token:

```
{
  "sub":"someone@example.com",
  "iss":"https://server.example.com",
  "nbf":1562262611,
  "exp":1562266216,
  "cnf":
  {
    "jkt":"0ZcOCORZNYy-DWpqq30jZyJGHTN0d2HglBV3uiguA4I"
  }
}
```

Vezanje DPoP dokaza za pristupni token pri zahtjevu za resursima

Vezanje DPoP dokaza za pristupni token prilikom zahtjeva za resursima temelji se na principu dvostrukog vezanja (eng. two-way binding). Ova metoda osigurava da su dva ključna elementa zahtjeva, dokazni token i pristupni token, kriptografski povezani.

Pristupni token sadrži hash ključa dokaznog tokena u "cnf" tvrdnji, dok DPoP zaglavlje zahtjeva uključuje dokazni token s poljem "ath" (access token hash). Vrijednost tog polja je hash pristupnog tokena, što stvara kriptografsko vezanje između dokaznog tokena i specifičnog pristupnog tokena koji se koristi u zahtjevu. Ova veza osigurava da dokazni token ne može biti upotrijebljen s drugim pristupnim tokenom, jer je striktno vezan uz konkretan pristupni token putem funkcije sažetka. Na taj način dvostruko vezanje sprječava napade zamjene tokena, gdje bi dokazni token mogao biti zloupotrijebljen s drugim pristupnim tokenom.

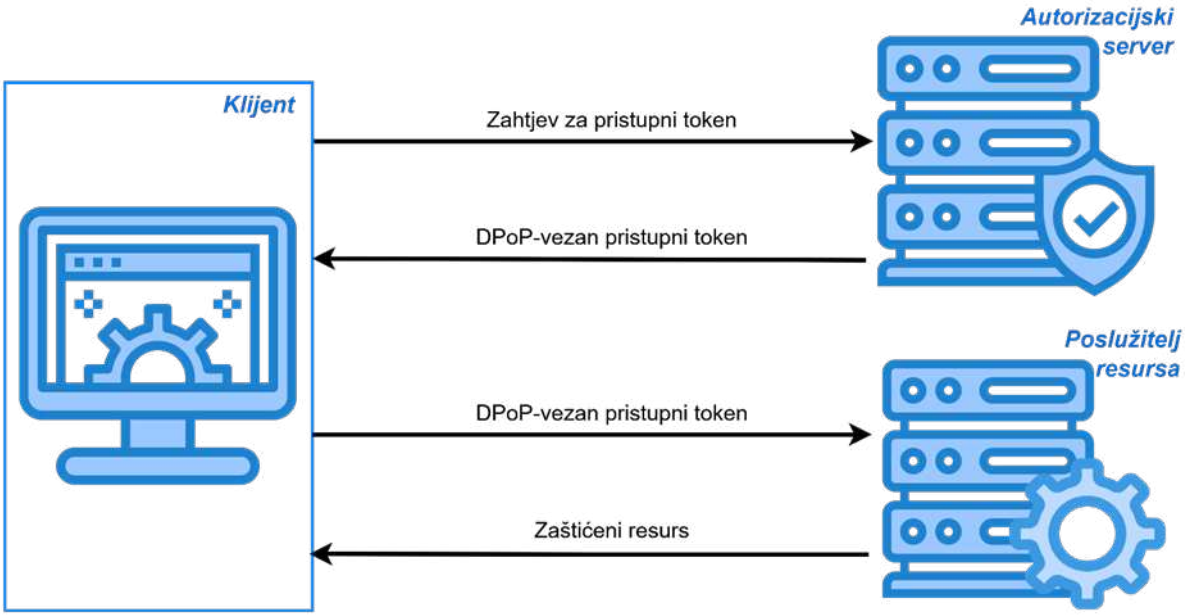
Resursni poslužitelj mora provjeriti autentičnost zahtjeva tako što izračuna hash dostavljenog pristupnog tokena i usporedi ga s vrijednošću u polju ath dokaznog tokena. Budući da je polje ath pokriveno digitalnim potpisom, ovo vezanje osigurava da je pristupni token zaista povezan s klijentom koji posjeduje privatni ključ korišten za potpisivanje DPoP dokaza. To znači da samo onaj tko posjeduje odgovarajući privatni ključ može generirati valjan dokaz za taj pristupni token.

Međutim, samo dvostruko vezanje ne osigurava zaštitu od napada ponovnog korištenja (eng. replay attack) DPoP dokaznog tokena – napada u kojem bi napadač ponovno iskoristio dokaz unutar iste sesije. Stoga, dodatni mehanizmi kao što su provjera vremenskog okvira dokaza te parametri poput "htm" i "htu" također moraju biti provjereni. *Replay cache* je mehanizam za čuvanje već korištenih dokaza unutar određenog vremenskog razdoblja i može spriječiti takve napade osiguravajući da se isti dokaz ne može ponovno koristiti.

```
{
  "typ":"dpop+jwt",
  "alg":"ES256",
  "jwk": {
    "kty":"EC",
    "x":"l8tFrhx-34tV3hRICRDY9zCkDlpBhF42UQUfWVAWBFs",
    "y":"9VE4jf_Ok_o64zbTTlcuNJajHmt6v9TDVrU0CdvGRDA",
    "crv":"P-256"
  }
}
.
{
  "jti":"e1j3V_bKic8-LAEB",
  "htm":"GET",
  "htu":"https://resource.example.org/protectedresource",
```

```
16   "iat":1562262618,
17   "ath":"fUHyO2r2Z3DZ53EsNrWBb0xWXoaNy59IiKCAqksmQEo"
18 }
```

Tok DPoP mehanizma



Slika 4.2: Tok DPoP mehanizma

1. U zahtjevu za pristupni token, klijent pošalje odobrenje za autorizaciju, te priloži DPoP dokazni token u zaglavlju tog HTTP zahtjeva. Dokazni token sadrži javni ključ i potpisan je privatnim ključem.

2. Autorizacijski poslužitelj validira dokazni token i, ako je valjan, izdaje pristupni token koji sadrži hash javnog ključa kojeg je klijent potvrdio u dokaznom tokenu. Ovaj hash postaje jedinstveni identifikator koji povezuje token uz specifični klijentov ključ.

3. Kako bi koristio pristupni token, klijent mora dokazati posjedovanje privatnog ključa tako da ponovno u zaglavlje zahtjeva priloži DPoP dokaz za taj zahtjev. Poslužitelj resursa treba primiti informaciju o javnom ključu koji je vezan za pristupni token

(preko pristupne točke za introspekciju ili direktno unutar tokena). Poslužitelj resursa verificira da se javni ključ iz pristupnog tokena i javni ključ iz DPoP dokaza podudaraju, te odgovara li hash pristupnog tokena u DPoP dokazu pristupnom tokenu iz zahtjeva.

4. Poslužitelj resursa odbija odgovoriti na zahtjev ako ne prođe provjera potpisa ili ako su podaci u DPoP dokazu pogrešni.

Validacija DPoP dokaza

Validacija DPoP dokaza uključuje niz provjera kako bi se osigurala valjanost i sigurnost. Prvo, poslužitelj mora potvrditi da je u HTTP zahtjevu prisutno točno jedno DPoP polje i da je to polje ispravno formatiran JWT. JWT mora sadržavati sve obavezne tvrdnje i imati parametar zaglavlja ”typ” postavljen na *dpop+jwt*, dok ”alg” parametar mora ukazivati na prihvaćeni asimetrični potpisni algoritam.

Nadalje, poslužitelj treba provjeriti da je potpis JWT-a verificiran javnim ključem sadržanim u ”jwk” parametru i da ”jwk” ne sadrži privatni ključ. Također, tvrdnje unutar JWT-a, uključujući ”htm” i ”htu”, moraju se podudarati s HTTP metodom i URI-jem zahtjeva, respektivno. Ako je poslužitelj pružio nonce vrijednost, ona mora odgovarati onoj koja je navedena u ”nonce” tvrdnji. Vrijeme kreiranja JWT-a, koje može biti određeno ”iat” tvrdnjom, mora biti unutar prihvatljivog vremenskog okvira.

Kada se DPoP dokaz koristi zajedno s pristupnim tokenom, vrijednost tvrdnje ”ath” mora odgovarati hash vrijednosti pristupnog tokena, te javni ključ vezan uz pristupni token mora se podudarati s javnim ključem iz DPoP dokaza.

Ove provjere omogućuju poslužitelju da potvrdi valjanost DPoP dokaza i osigura da se pristupni token koristi samo od strane onih koji posjeduju odgovarajući privatni ključ, čime se sprječava neovlaštena upotreba tokena.

Zaključak

Prednost DPoP-a je njegova neovisnost o mrežnoj infrastrukturi, što omogućuje njegovo korištenje u različitim okruženjima bez potrebe za implementacijom složenih mrežnih rješenja poput PKI-a. DPoP koristi široko dostupne kriptografske metode poput RSA ili eliptičkih krivulja, te JWT za stvaranje i validaciju dokaznih tokena, što dodatno olakšava njegovu implementaciju.

Međutim, postoje izazovi. Budući da se radi o novoj tehnologiji, ona zahtijeva prilagodbe na svim razinama aplikacije – klijentske aplikacije, poslužitelji tokena i API-ji moraju podržavati DPoP i implementirati dodatne mehanizme, poput zaštite od ponavljanja napada (replay cache). Također, budući da DPoP funkcionira na aplikacijskoj razini, to može zahtijevati značajne promjene u postojećem kodu i infrastrukturi.

Iako su mTLS i DPoP različiti pristupi, oba imaju svoje mjesto u poboljšanju sigurnosti API-ja. mTLS se pokazao učinkovitim u okruženjima s već postojećom PKI infrastrukturom, dok DPoP nudi fleksibilniju i moderniju alternativu koja se može koristiti u različitim aplikacijskim scenarijima. Ovisno o specifičnim potrebama i tehničkim mogućnostima, organizacije mogu odabrati najprikladniji mehanizam za osiguranje svojih API-ja i zaštitu osjetljivih podataka.

Bibliografija

[1] B. Campbell, J. Bradley, N. Sakimura i T. Lodderstedt, *OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens*, RFC 8705, https://www.rfc-editor.org/info/rfc8705, veljača 2020.

[2] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley i W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 5280, https://www.rfc-editor.org/info/rfc5280, svibanj 2008.

[3] T. Dierks i E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, https://www.rfc-editor.org/info/rfc5246, kolovoz 2008.

[4] W. Diffie i M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory **22** (1976), br. 6, 644–654.

[5] A. Dujella, *Kriptografija i sigurnost mreža*, https://web.math.pmf.unizg.hr/~duje/kriptsig.html, Accessed: 2024-10-13.

[6] D. Fett, B. Campbell, J. Bradley, T. Lodderstedt, M. Jones i D. Waite, *OAuth 2.0 Demonstrating Proof of Possession (DPoP)*, RFC 9449, https://www.rfc-editor.org/info/rfc9449, rujan 2023.

[7] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749, https://www.rfc-editor.org/info/rfc6749, listopad 2012.

[8] M. Jones, *JSON Web Key (JWK)*, RFC 7517, https://www.rfc-editor.org/info/rfc7517, svibanj 2015.

[9] M. Jones, J. Bradley i N. Sakimura, *JSON Web Token (JWT)*, RFC 7519, https://datatracker.ietf.org/doc/html/rfc7519, svibanj 2015.

[10] M. Jones i D. Hardt, *The OAuth 2.0 Authorization Framework: Bearer Token Usage*, RFC 6750, https://www.rfc-editor.org/info/rfc6750, listopad 2012.

[11] M. Jones i N. Sakimura, *JSON Web Key (JWK) Thumbprint*, RFC 7638, https://www.rfc-editor.org/info/rfc7638, rujan 2015.

[12] N. Madden, *API Security in Action*, Manning, 2020.

[13] National Institute of Standards and Technology (NIST), *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, FIPS PUB 202, kolovoz 2015, http://dx.doi.org/10.6028/NIST.FIPS.202.

[14] B. Van Rompay, *Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers*, Disertacija, Katholieke Universiteit Leuven, Electrical Engineering Department, Leuven, Belgium, 2004.

Sažetak

Ovaj rad istražuje ključne aspekte sigurnosti API-ja, naglašavajući njihovu ulogu u zaštiti osjetljivih podataka i sprječavanju neovlaštenog pristupa. Analiziraju se osnovni sigurnosni mehanizmi koji su neophodni za kontrolu pristupa resursima i zaštitu korisničkih informacija. Razumijevanje ovih mehanizama omogućava bolje upravljanje sigurnosnim rizicima u dinamičnom okruženju.

Analiziraju se kriptografske tehnike, uključujući simetrične i asimetrične kriptosustave, koje osiguravaju povjerljivost, integritet i neporecivost podataka u prijenosu. Ove kriptografske osnove omogućuju razvoj naprednijih sigurnosnih mehanizama koji su potrebni za učinkovitu zaštitu API-ja.

Poseban fokus stavljen je na tehnike dokazivanja posjedovanja kriptografskih ključeva, poput mTLS-a i DPoP-a, koje osiguravaju da samo ovlašteni korisnici mogu pristupiti zaštićenim resursima. Kroz usporedbu ovih tehnika, rad prikazuje njihove prednosti i ograničenja u različitim sigurnosnim scenarijima. Ove metode značajno smanjuju rizik od napada na komunikacijske kanale i neovlaštenog pristupa, što ih čini neophodnima u dizajnu sigurnih i pouzdanih API-ja.

Summary

This thesis explores key aspects of API security, emphasizing their role in protecting sensitive data and preventing unauthorized access. It analyzes the fundamental security mechanisms essential for controlling access to resources and safeguarding user information. Understanding these mechanisms enables better management of security risks in a dynamic environment.

Cryptographic techniques are examined, including symmetric and asymmetric cryptosystems, which ensure the confidentiality, integrity, and non-repudiation of data in transit. These cryptographic foundations facilitate the development of more advanced security mechanisms necessary for effective API protection.

Particular emphasis is placed on proof of possession techniques for cryptographic keys, such as mTLS and DPoP, which ensure that only authorized users can access protected resources. Through a comparison of these techniques, the thesis illustrates their advantages and limitations in various security scenarios. These methods significantly reduce the risk of attacks on communication channels and unauthorized access, making them essential in the design of secure and reliable APIs.

Životopis

Rođena sam 6. prosinca 1999. u Zagrebu. Nakon što sam završila osnovnu školu, upisala sam XV. gimnaziju u Zagrebu, gdje sam otkrila svoje zanimanje za matematiku. Godine 2018. upisujem prijediplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta.

Tijekom 2021. godine započela sam raditi kao junior developer u fintech industriji, što je dodatno usmjerilo moje profesionalne interese. Ova iskustva u praksi inspirirala su me da nastavim obrazovanje i 2022. godine upisujem diplomski studij Računarstvo i matematika na istom fakultetu.