Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Porovnanie API tokenov

Bakalárska práca

2023
Jitka Muravská

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Porovnanie API tokenov

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Richard Ostertág, PhD.

Bratislava, 2023
Jitka Muravská

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jitka Muravská
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Porovnanie API tokenov
Comparison of API Tokens

Anotácia: REST aplikačné rozhrania (API) často nebývajú čisto verejné. V takom prípade treba pri implementácii API riešiť aj jeho bezpečnosť. Väčšina schém zabezpečenia API používa token, ktorý je súčasťou jednotlivých požiadaviek. Tieto tokeny sú nejakým spôsobom spojené s identitou a autorizáciou používateľa. Aplikačné rozhranie prevezme požiadavku, extrahuje token, a podľa pravidiel prístupu rozhodne ako pokračovať.

Cieľom práce je porovnanie rôznych API tokenov (napríklad: OAuth 2.0, JWT, PASETO, Protobuf Tokens, Authenticated Requests, Macaroons, Biscuits, …). Prvým krokom bude zozbieranie a popísanie rôznych v praxi používaných API tokenov. Následne sa vykonajú porovnania ich výhod a nevýhod (napríklad rýchlosť, jednoduchosť použitia) na jednoduchej základnej aplikácii.

Vedúci: RNDr. Richard Ostertág, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 12.10.2022

Dátum schválenia: 13.10.2022 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie:

Abstrakt

Kľúčové slová:

Abstract

Keywords:

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Kapitola 1

Schémy zabezpečenia API a využitie tokenov

V tejto kapitole uvedieme viaceré známe prístupy riešenia autentifikácie a autorizácie. Stručne objasníme ako fungujú a aké sú ich výhody a nevýhody. Detailnejšie sa pozrieme na prístup využívajúci API tokeny a samotné tokeny a ich rozdelenie, ktorým sa venuje naša práca.

1.1 Zabezpečenie bez autentifikácie

V prípade, že je aplikačné rozhranie plne verejné, nepotrebuje žiadnu schému zabezpečenia. Ľubovoľný používateľ môže volať rozhranie bez predchádzajúcej autentifikácie a neexistuje spôsobom ako obmedziť prístup k volaniam rozhrania.

Jediným identifikátorom autora požiadavky je jeho IP adresa. To je však veľmi slabý identifikátor a nedáva nám veľké možnosti v obmedzení prístupu k rozhraniu.

Hlavnou výhodou tohto riešenia je jednoduchosť implementácie, rozhranie nepotrebuje uchovávať žiadne dáta o používateľoch a všetky požiadavky sú jednoduché, ich rýchlosť závisí len od rýchlosti samotného volania a rýchlosti siete.

Nevýhodou je samozrejme strata kontroly nad prístupom k rozhraniu, ktorá sa obmedzila buď na nejakú kontrolu na základe IP adresy alebo úplne vymizla. A jediným rozumných obmedzením je obmedzenie počtu požiadaviek za nejaký časový interval, či už pre konkrétne IP adresy alebo pre celé rozhranie.

1.2 Zabezpečenie bez schémy

Najjednoduchšie riešenie autentifikácie a autorizácie je posielanie prihlasovacích údajov v každej požiadavke na rozhranie. Klient jednoducho pripojí prihlasovacie údaje ku každej požiadavke a rozhranie si ich overí vo svojej databáze a v prípade úspechu vráti

požadované dáta.

Toto riešenie zjavne nie je vhodné ak sa niekde v rámci komunikácie nachádza nezabezpečené spojenie. Útočník, ktorý by takúto komunikáciu zachytil, by mal jednoduchý prístup k prihlasovacím údajom používateľa.

Používanie nezabezpečeného spojenia je všeobecne nebezpečné bez ohľadu na schému zabezpečenia a typ prenášaného údaju používaného na autorizáciu, preto ďalej v práci budeme predpokladať, že spojenia s každým koncovým bodom sú zabezpečené pomocou SSL/TLS.

Aj v prípade zabezpečeného spojenia však existujú zraniteľnosti [22]. Prihlasovacie údaje sú zriedkavo menený identifikátor a teda po ich odchytení sa dajú dlho zneužívať. Okrem samotného úniku citlivých údajov a z toho vyplývajúcich nepríjemností pre používateľa má tento prístup aj ďalšie nevýhody.

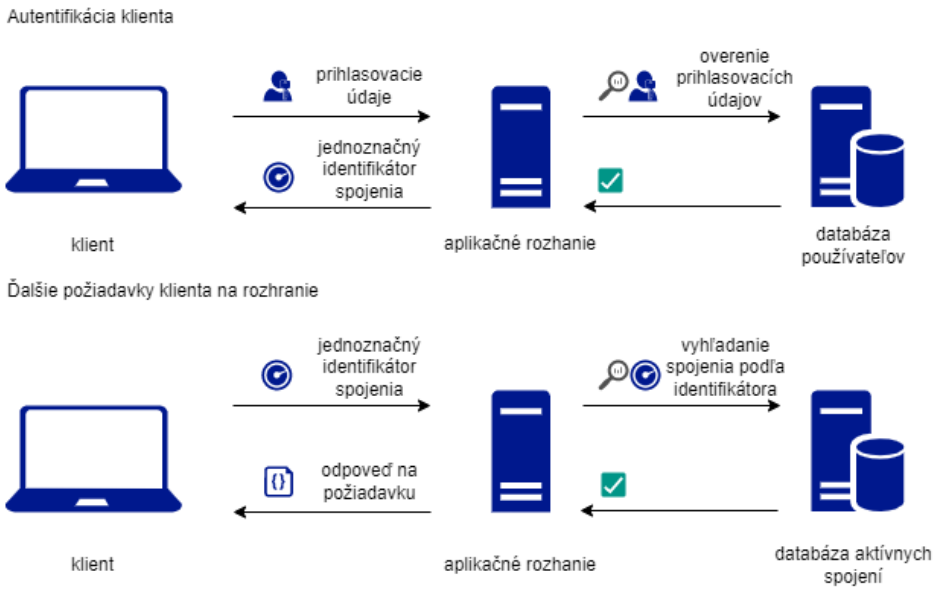
Útočník často nezneužíva získané heslá hneď, ale ukladá ich do databázy odchytených hesiel a následne túto databázu môže použiť pri útokoch. Tento prístup sa nazýva *slovníkový útok* a môže byť veľmi efektívny. [20]. Ľudia tiež často používajú podobné heslá na všetky svoje účty. Potom sa podľa istých vzorov odhalených v sade hesiel jedného používateľa dajú ľahšie uhádnuť ďalšie heslá. [10].

1.3 Schéma zabezpečenia s využitím identifikátora spojenia

Prvé riešenie, kde môžeme hovoriť o nejakej schéme zabezpečenia, nie len o existencii prihlasovacích údajov a ich overení pri každej požiadavke, je sledovanie každého aktívneho spojenia (angl. session) s rozhraním.

Pri prvej požiadavke sa používateľ autentifikuje voči serveru, ten si vytvorí a uloží záznam o spojení. Tento záznam môže obsahovať rôzne základné informácie a o spojení ako čas vytvorenia a informácie o používateľovi, s ktorým je spojenie vytvorené. Dôležitá časť záznamu je jednoznačný identifikátor spojenia v podobe náhodného reťazca. Následne klientovi vráti identifikátor spojenia. Všetky nasledujúce požiadavky klienta budú obsahovať tento identifikátor, podľa ktorého vie server určiť, ktorému používateľovi patria, teda či je autentifikovaný, prípadne či má právo vykonať dané volanie, ak rozhranie implementuje nejakú autorizačnú schému. Popísaná schéma je znázornená na obrázku 1.2.

Oproti predošlému riešeniu pribudla potreba manažovať spojenia, no veľkou výhodou je, že sa už neposielajú citlivé prihlasovacie údaje v každej požiadavke. Rozhranie však stále potrebuje pri každej požiadavke preložiť identifikátor na záznam o používateľovi.



Obr. 1.1: Autentifikačná schéma s použitím identifikátora spojenia.

1.4 Schéma zabezpečenia využívajúca API kľuče

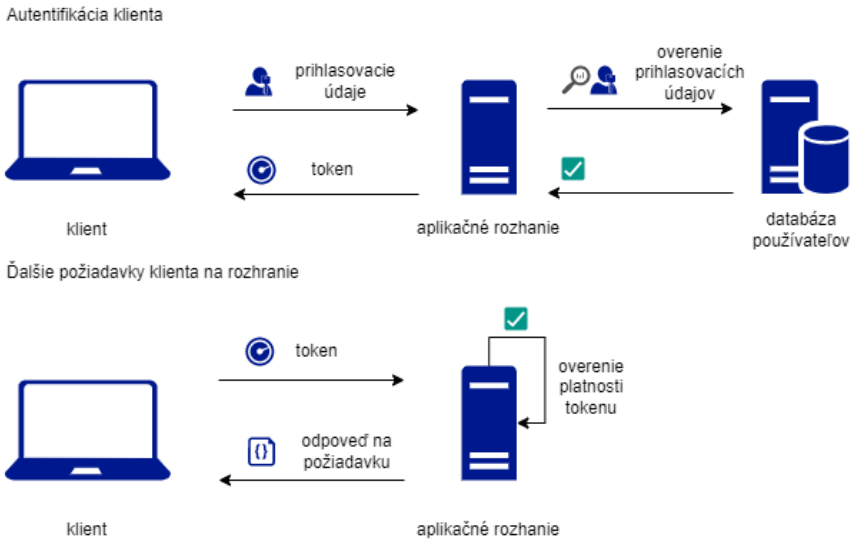
API kľúč je identifikátor používaný na pripojenie k rozhraniu. Identifikuje však aplikáciu alebo službu, nie konkrétneho používateľa. Ak potrebuje aplikácia používať rozhranie, ktoré nie je verejné a na autentifikáciu používa API kľúče, musí si najprv vyžiadať API kľuč a následne ho posielať s každou požiadavkou.

Tento kľúč môže byť zároveň spojený s právami umožňujúcimi vykonávať určité volania, ak dané rozhranie poskytuje rôzne úrovne prístupu.

Použitie API kľúčov nie je veľmi bezpečné, lebo kľúče majú neobmedzenú platnosť a ak príde k ich odchyteniu, jediný spôsob ako zvrátiť ich zneužitie je ich zneplatniť a vygenerovať nové. Ďalšia zraniteľnosť API kľúčov často nastáva pri ich uložení na nesprávne miesto.

Napríklad pri webovej aplikácii nie je bezpečné ukladať kľúč na klientskej strane, ale na serveri a odtiaľ vykonávať API volania. Rovnako pri mobilnej aplikácii nie je bezpečné ukladať kľúč priamo v aplikácii, lebo sa dá získať reverzným inžinierstvom. [5]

Preto sa vo všeobecnosti API kľúče nepoužívajú na ochranu kritických alebo citlivých volaní a zdrojov rozhrania. Naopak vhodné sú na zvýšenie kontroly nad požiadavkami na rozhranie. Je ľahké pomocou nich kontrolovať frekvenciu, či absolútny počet volaní a prípadne monetizovať ich používanie.



Obr. 1.2: Autentifikačná schéma s použitím tokenu.

1.5 Schéma zabezpečenia využívajúca API tokeny

API token (ďalej token) je identifikátor, ktorý slúži na autorizáciu prípadne aj identifikáciu používateľa pri prístupe k rozhraniu. Token môže mať viaceré podoby, jednotlivým typom a formátom sa venujú podkapitoly 1.6 a 1.7.

Tok v rámci schémy zabezpečenia rozhrania funguje podobne ako pri využití identifikátora spojenia, ktorý sme popísali v podkapitole 1.3. Líši sa najmä v tom, že token môže niesť pridanú informáciu a rozhranie vie jeho platnosť overiť bezstavovo. Na dosiahnutie tejto výhody je nutné použiť štruktúrovaný formát tokenu. Možné formáty tokenov detailnejšie popíšeme v podkapitole 1.7.

1.6 Typy tokenov

Tokeny využívané v schéme zabezpečenia rozhrania sa dajú rozdeliť podľa ich generického využitia na niekoľko typov. Nie v každom scenári máme explicitne dané aký token využiť, no niektoré sú často vhodnejšie ako iné. V tejto kapitole predstavíme viaceré typy tokenov a to prístupový (angl. access), nositeľský (angl. bearer), obnovovací (angl. refresh) a identifikačný token.

1.6.1 Prístupový token

Prístupový token je najzákladnejší a najčastejšie používaný typ tokenu. Tento token vygeneruje autentifikačný server po autentifikácii používateľa a ďalej slúži na autorizáciu v rozsahu ako sa určil v rámci autentifikácie konkrétneho používateľa. Prístupový

token môže mať rôzne formáty podľa toho ako funguje schéma zabezpečenia rozhrania.

Token je spojený s istými údajmi o používateľovi ako napríklad identifikátor a prístupové práva. Tieto údaje môžu byť uložené priamo v tokene alebo na serveri, v závislosti od toho aký formát tokenu je použitý. Zároveň je s tokenom spojený aj časový limit platnosti tokenu, ktorý býva kvôli bezpečnosti krátky. Po odchytení tokenu totiž môže útočník vystupovať v mene používateľa, ktorého údaje sú uložené v tokene.

1.6.2 Nositeľský token

Nositeľský token reprezentuje najčastejší spôsob použitia prístupového tokenu. Pri jeho použití má nositeľ tokenu prístup k rozhraniu bez ohľadu nato, kto je. Stačí aby v požiadavke poslal platný nositeľský token.

Následne rozhranie nekontroluje identitu používateľa, ale iba platnosť tokenu. Preto pri jeho využití treba prikladať väčší dôraz na bezpečnosť tokenu napríklad tak, že jeho platnosť bude veľmi krátka, napríklad 5 minút.

1.6.3 Obnovovací token

Ako sme už spomínali vyššie má prístupový token často relatívne krátky časový limit platnosti. Ak chce používateľ využívať rozhranie aj po jeho uplynutí, je potrebné aby získal nový prístupový token.

Na riešenie tohto problému existujú dva jednoduché spôsoby. Buď sa používateľ znova autentifikuje a server mu vygeneruje nový prístupový token alebo si klient, cez ktorého používateľ komunikuje s rozhraním bude pamätať prihlasovacie údaje používateľa a využije ich na znovuautentifikáciu používateľa. Oba prístupy nie sú ideálne, prvá možnosť je nekomfortná pre používateľa a nie je možná ak je klientom nejaký servis alebo iná aplikácia. Druhá možnosť je nebezpečná, lebo klient musí mať niekde uložené prihlasovacie údaje používateľa a tým sa zvyšuje riziko ich získania útočníkom.

Obnovovací token adresuje problém s krátkou platnosťou prístupových tokenov. Pri prvotnej autentifikácii používateľa server okrem prístupového tokenu vygeneruje aj obnovovací token. Oba tokeny vráti klientovi. Následne keď uplynie platnosť prístupového tokenu, klient pošle požiadavku o nový prístupový token pomocou obnovovacieho tokenu.

Schému s obnovovacími tokenmi využíva napríklad populárny autentifikačný protokol OAuth 2.0 [11].

1.6.4 Identifikačný token

Identifikačný token je špeciálny typ prístupového tokenu. V tomto prípade musí ísť o štruktúrovaný formát tokenu. Token obsahuje identifikátor používateľa a autorizačné

údaje. Navyše môže token obsahovať doplnkové údaje ako vydavateľa tokenu, čas platnosti tokenu, čas vydania tokenu a podobne.

Identifikačný token využíva napríklad protokol OpenID Connect (OIDC), čo je nadstavba protokolu OAuth 2.0 o identitu používateľa [24]. OIDC dokonca presne špecifikuje použitie JWT (JSON Web Token), ktorý viac predstavíme v kapitole 2.

1.7 Formáty tokenov

Už sme spomenuli, že existujú rôzne typy tokenov, no iba pri identifikačnom tokene je striktne daný formát tokenu. Pri ostatných typoch sa stretávame s rôznymi formátmi tokenu. Vo všeobecnosti rozoznávame dva formáty tokenov a to nepriehľadné (angl. opaque) a štruktúrované. Existujú aj hybridné formáty, ktoré kombinujú výhody obidvoch spomínaných typov, no pri nich sa dá hovoriť skôr o istých vzoroch ako o formátoch. Najpoužívanejšie sú fantómové a rozdelené (angl. split) tokeny.

1.7.1 Nepriehľadný token

Ide o najjednoduchší formát tokenu. Nepriehľadný token je náhodný reťazec znakov. Jednoduchý je v tom, že nenesie žiadnu pridanú informáciu pre rozhranie. Všetky metadáta o tokene si musí pamätať rozhranie.

Takýto prístup so sebou nesie významné výhody aj nevýhody. Výhodou je, že samotné vytvorenie tokenu je veľmi rýchle, rozhranie jednoducho vygeneruje náhodný reťazec. Navyše vďaka tomu, že nenesie žiadne pridané informácie, nenesie ani citlivé informácie o používateľovi, teda jeho zachytenie útočníkom nie je také nebezpečné. Nevýhodou však je, že rozhranie ho nevie bezstavovo overiť, teda ho musí napríklad vyhľadať v databáze platných tokenov a to je časovo náročné.

1.7.2 Štruktúrovaný token

Na rozdiel od nepriehľadného tokenu štruktúrovaný token obsahuje pridanú informáciu napríklad identifikáciu používateľa, jeho prístupové práva, čas platnosti tokenu, čas vydania tokenu a podobne.

Aby sa predišlo úniku citlivých informácií z tokenu, je token šifrovaný alebo zahešovaný. Využívajú sa rôzne symetrické aj asymetrické algoritmy. Všetky konkrétne protokoly, ktoré rozoberáme v tejto práci, používajú štruktúrované tokeny a v kapitole 2 sa venujeme ich vytváraniu a s ním spojenému šifrovaniu alebo hešovaniu tokenov.

Najväčšou výhodou štruktúrovaného tokenu je, že rozhranie ho môže bezstavovo overiť, lebo pozná kľúč, ktorým bol token zašifrovaný alebo zahešovaný, teda ho vie

dešifrovať a prečítať informácie, ktoré nesie. Zo získaných informácií vie rozhranie overiť platnosť tokenu a autorizovať používateľa.

1.7.3 Fantómový token

Ako sme naznačili v úvode podkapitoly, fantómový token je hybridný formát tokenu. Podľa tohto vzoru autentifikačný server vygeneruje dva tokeny, nepriehľadný token pre klienta a štruktúrovaný token pre rozhranie. Ďalej sa využíva API brána alebo reverzný proxy server (RPS), v ktorom sa uloží dvojica tokenov ako kľúč a hodnota. Kľúčom je nepriehľadný token, hodnotou je štruktúrovaný token.

Následne, keď klient pošle požiadavku na rozhranie, tak pridá nepriehľadný token. API brána alebo RPS ho preloží na štruktúrovaný token a tento štruktúrovaný token poskytne rozhraniu. Rozhranie ho môže bezstavovo overiť a využiť pridané informácie, ktoré nesie.

Brána alebo RPS síce musí vyhľadať záznam s dvojicou tokenov, kde je kľúčom poslaný nepriehľadný token, ale môže si výsledok uložiť do vyrovnávacej pamäte pre ďalšie požiadavky. Tým sa zvýši priepustnosť brány alebo RPS a zároveň ubudnú nároky na výkonnosť rozhrania.

Výhodou oproti obyčajnému štruktúrovanému tokenu je, že klient, teda prehliadač alebo iná aplikácia nedržia v pamäti štruktúrovaný token obsahujúci, aj keď zašifrované, citlivé informácie.

1.7.4 Rozdelený token

Rozdelený token má podobnú schému a výhodu ako fantómový token, no navyše ombedzuje štruktúrovaný token vydaný autentifikačným serverom tým, že musí obsahovať podpis chrániaci jeho autenticitu.

V schéme rozdeleného tokenu vydá autentifikačný server len jeden štruktúrovaný token. Podpis z tohto tokenu pošle klientovi a do vyrovnávacej pamäte brány alebo RPS zapíše celý token so zahešovaným podpisom ako kľúčom.

Požiadavky od klienta budú obsahovať ako token získaný podpis a brána alebo RPS ho zahešuje a preloží na štruktúrovaný token pomocou vyrovnávacej pamäte. Token následne spolu s požiadavkou pošle rozhraniu.

Kapitola 2

Šprecifikácia konkrétnych API tokenov

V tejto kapitole predstavíme v praxi využívané API tokeny, ktorými sa zaoberá naša práca. Uvedieme ich formát a základané charakteristiky ako ich vytvorenie, či validácia.

Ich využitie, výhody či nevýhody rozoberieme v kapitole 3.

2.1 JSON Web Token

Prvý token, ktorým sa budeme zaoberať je JSON web token (JWT) [16]. JWT vznikol ako súčasť JOSE [4] (JSON object signing and encryption) štandardov, čo je dokument vypracovaný pracovnou skupinou IETF (Internet Engineering Task Force) na základe aplikácií bezpečnostných mechanizmov v rámci vývoja softvéru. Tieto štandardy popisujú využitie a definujú požiadavky na objekty formátu JSON, ktoré sú bezpečné.

Definujú štandard pre bezpečný prenos JSON objektov medzi stranami, ktoré sú schopné ich overiť a dešifrovať. Zavádzajú tri základné formáty JSON objektov a to JWS (JSON Web Signature), JWE (JSON Web Encryption) a JWK (JSON Web Key), ktorým sa detailnejšie venujú ďalšie štandardy [15, 17, 14]. Prvé dva sú formáty zabezpečujúce bezpečnostné vlastnosti JSON objektov. Oba zabezpečujú autentickosť a integritu pomocou elektronických podpisov alebo hešovaného autentifikačného kódu (HMAC) [7], ďalej budeme hovoriť v oboch prípadoch o podpise tokenu. JWE navyše zabezpečuje aj dôvernosť a to šifrovaním obsahu JSON objektu. Posledný formát JWK je formát pre reprezentáciu kľúčov, ktoré sú použité v kryptografických algoritmoch využitých v JWS a JWE. Kryptografické algortimy a ich identifikátory sú definované JSON Web Algorithms (JWA) štandardom [13].

2.1.1 Štruktúra JWT

Samotný JWT je vpodstate iba serializácia JSON objektu chráneného JWS alebo JWE. Podľa štandardu JWT obsahuje tri samostatné časti oddelené bodkami - hlavičku, telo a podpis. Hlavička a telo sú serializované JSON objekty obsahujúce oprávnenia

(angl. claim) vo forme dvojíc kľúč, hodnota. Niektoré kľúče niektorých oprávnení sú definované v štandarde a teda by sa nemali používať pre žiadne iné hodnoty.

A to konkrétne v hlavičke najdôležitejšie sú *typ* a *alg*. Prvý určuje typ tokenu a druhý algoritmus, ktorý bol použitý na vytvorenie podpisu alebo vrámci šiforvania obsahu tokenu. Rôzne možnosti pre algortimy sú definované v JWA štandardoch.

Telo tvorí logický obsah tokenu, napríklad môže obsahovať oprávnenia týkajúce sa kontrétnej autentifikácie a používateľa, pre ktorého bol token vydaný. Dôležité Štandardom popísané kľúče sú napríklad *iss*, *sub*, *exp*, *nbf*, *iat*. Popisujú postupne vydavateľa tokenu, identifikátor používateľa, čas vypršania platnosti tokenu, čas, kedy sa token začne považovať za platný a čas vydania tokenu.

Do tela aj hlavičky sa môžu vkladať ľubovoľné iné oprávnenia, naprílad *admin*, *role*, *permissions*, určujúce oprávnenia používateľa.

2.1.2 Generovanie a validácia JWT

Hlavička a telo sa serializujú pomocou base64url kódovania [18]. V prípade JWE sa telo ešte šifruje. Následne sa obe časti podpíšu algoritmom definovaným v hlavičke a podpis sa zreťazí s hlavičkou a telom. Výsledný reťazec sa používa ako token.

Na overenie platnosti tokenu treba zvalidovať podpis. Podpis je vytvorený pomocou algoritmu definovanom v hlavičke. Teda pri validácii tokenu sa ako prvé dekóduje hlavička tokenu a z nej sa prečíta hodnota v kľúči *alg*.

Na základe hodnoty v kľúči *alg* sa určí algoritmus, ktorý bol použitý na podpis tokenu. Následne sa podpis zvaliduje.

Ak bol token vytvorený pomocou JWE, na prečítanie tela je potrebné ho najprv dešifrovať. V prípade využitia JWS je telo po dekódovaní priamo čitateľné. Následne môžeme overiť informácie o časovej platnosti tokenu, či právach používateľa a pod.

2.2 Platform Agnostic SEcurity TOken

Platform Agnostic SEcurity TOken (PASETO) je relatívne nový štandard tokenu navrhnutý v roku 2018 a je stále v štádiu RFC draftu [3]. Je inšpirovaný rodinov štandardov JOSE (JWT, JWS, JWE, JWK). Jednoducho povedané sa snaží zjednodušiť implementáciu a použitie kryptografických funkcií.

Rovanko ako JWT, PASETO serializuje JSON objekty a zaručuje rôzne bezpečnostné kvality pri ich prenose cez internet. Pôvodne bol PASETO navrhnutý s dvoma verziami *v1* a *v2* líšiacimi sa v použitých kryptografických algoritmoch. Dnes už existujú štyri verzie *v1*, *v2*, *v3* a *v4* popísané štandardom [2]. Každá verzia tokenu zaručuje autentickosť a integritu obsahu tokenu a to pomocou asymetrického šifrovania v zmysle

elektronického podpisu alebo pomocou hešovaného autentifikačného kódu (HMAC) [7]. Obe tieto možnosti budeme nazývať podpis a proces ich vytvárania podpisovanie.

2.2.1 Verzie PASETO

Ako sme spomenuli PASETO má štyri verzie. Každá verzia sa delí na dve ďalsie podľa ich využitia na lokálne a verejné. Lokálne tokeny majú zašifrované telo a tým zabezpečujú dôvernosť dát uložených v tele tokenu. Narozdiel od toho sú verejené tokene nešifrované a dáta v ich tele sú čitateľné pre kohokoľvek s prístupom k danému tokenu.

Každá verzia PASETO používa iný algoritmus na podpisovanie a prípadne šifrovanie tokenu v prípade lokálnych tokenov. Jednotlivé algortimy pre konkrétne verzie a ich použitie je popísané v špecifikácii [2].

Novšie verzie *v3* a *v4* prinášajú modernejšie kryptografické algortmy a pridávajú niektoré funkcionality. Napríklad verzia *v3* prináša nepopierateľnosť autorstva ako novú bezpečnostnú kvalitu. Dosahuje to dokonca bez predĺženia podpisu a to pomocou pridania verejného kľuča do tokenu pred vypočítaním podpisu [21]. Tiež zavádza podporu pre implicitné informácie, teda také informácie, ktoré nie sú uložené priamo v tokene, ale používajú sa pri výpočte podpisu. Teda sú to informácie potrebné pre validáciu tokenu, ale z nejakého dôvodu nie je vhodné ich vkladať priamo do tokenu. Napríklad môže ísť o citlivé interné dáta. Podrobná motivácia za zavedením nových verzií je popísaná v špecifikácii [2].

2.2.2 Štruktúra PASETO

PASETO sa skladá z troch alebo štyroch častí zreťazených bodkov. Časti postupne reprezentujú verziu, využitie, telo a pätu. Prvé tri časti sú povinné a päta je nepovinná, ale dovoľuje nám zapísať akékoľvek ďalšie informácie do tokenu.

- Verzia - reprezentuje verziu PASETO. Môže byť *v1*, *v2*, *v3* alebo *v4*.
- Využitie - určuje využitie tokenu ako lokálne alebo verejné. Možné hodnoty sú *local* alebo *public*.
- Telo - reprezentuje samotné dáta uložené v tokene. Podobne ako pri JWT ide o oprávnenia vo forme dvojíc kľúč hodnota a rovnako sú niektoré doležité kľúče sú definované špecifikáciou. [2]
- Päta - môže obsahovať ľubovoľné ďalšie informácie.

Telo a päta sú vo forme JSON objektu, ktorý je serializovaný pomocou *base64url* [18].

2.2.3 Generovanie a validácia PASETO

Pri vytváraní tokenu sa musíme najprv rozhodnúť pre verziu a využitie podľa toho aké bezpečnosté požiadavky máme na token. Následne vytvoríme telo tokenu obsahujúce informácie, ktoré chceme pomocou tokenu prenášať, napríklad informácie o vzniku tokenu, jeho platnosti, jeho autorovi, či určenému subjektu. Ďalej môžeme pridať ďalšie informácie do päty tokenu ako napríklad identifikátor kľúča kryptografickej funkcie.

Ak sme zvolili lokálne využitie, tak telo tokenu zašifrujeme. Následne vypočítame podpis z tela a päty tokenu. Šifrovanie aj podpisovanie sa deje pomocou kryptografickej funkcie vybraj podľa verzie a využitia.

Nakoniec všetky časti spojíme do jedného reťazca a oddelíme bodkami.

Validácia je tokenu je inverzný proces ku generovaniu. Najprv rozdelíme reťazec na časti a zistíme verziu a využitie tokenu a podľa toho vyberieme použitú kryptografickú funkciu. Následne zistíme, či je podpis tokenu platný pomocou adekvátnej kryptografickej funkcie a kľúča. Ak mal token lokálne využitie dešifrujeme jeho telo a skontrolujeme časovú plastnosť tokenu, ak to využívaná schéma podporuje a telo obsahuje informácie o platnosti tokenu.

2.3 Fernet

(Názov tohto tokenu je naozaj inšpirovaný známym nápojom. haha aka som vtipna :D) Pôvodne Fernet vznikol ako nástroj na zasielanie bezpečných správ v platforme cloudových služieb Heroku [8]. V súčasnosti už podľa špecifikácie [9] vzniklo veľa implementácií Fernetu v rôznych programovacých jazyok [**?**, 19], v rámci Heroku bol implementovaný v Ruby. Fernet bol dokonca vybratý PYCA (Python Cryptographic Authority) [23] ako štandard pre implementáciu symetrického šifrovania v Pythone.

Fernet je štruktúrovaný token, lebo v sebe nesie rôzne informácie, no nijak nešpecifikuje formát týchto informácií. Väčšina implementácií s nimi pracuje ako s obyčajným reťazcom znakov.

Token je navrhnutý s možnosťou pridania viacerých verzií, no v súčasnosti existuje len jedeina verzia. V tejto verzií je obsah tokenu zašifrovaný pomocou AES-128-CBC [6] a celý token je podpísaný pomocou HMAC-SHA256 [7]. Z toho vyplýva, že Fernet zaručuje autentickosť, integritu a dôvernosť.

2.3.1 Štruktúra Fernet

Fernet sa skladá z piatich zreťazených častí. Každá časť reprezentuje jednu informáciu o tokene. Jednotlivé časti sú nasledovné:

- Verzia - reprezentuje verziu Fernet tokenu, aktuálne existuje len jedna verzia a je reprezentovaná číslom 0x80. Zaberá vždy 8 bitov.
- Časová pečiatka - reprezentuje čas vytvorenia tokenu. Čas je zachytení ako počet sekúd od 1.1.1970 v UTC časovej zóne. Zaberá vždy 64 bitov.
- Inicializačný vektor (IV) - reprezentuje náhodný reťazec znakov, ktorý je použitý pri šifrovaní a dešifrovaní tokenu. IV musí byť unikátny a najmä nepredvídateľný pre každý token, preto sa generuje náhodnou funkciou. Zaberá vždy 128 bitov.
- Zašifrované telo - reprezentuje zašifrované dáta uložené v tokene. Môže mať premenlivú dĺžku, no vždy násobok 128 bitov.
- HMAC - reprezentuje výstup HMAC-SHA256 funkcie a zaberá vždy 256 bitov.

Ako vidíme, všetky časti tokenu okrem tela majú pevne danú dĺžku. Vďaka tejto vlastnosti nemusia byť zreťazené časti oddelené žiadnym špeciálnym symbolom, napríklad bodkou, lebo vieme jednoznačne oddeliť verziu, časovú pečiatku, IV aj HMAC a tým pádom aj zašifrované telo.

Pre jednoduché prenášanie je celý token zakódovaný pomocou *base64url* [18].

2.3.2 Generovanie a validácia Fernet

Pri generovaní Fernet tokenu sa využívajú dva kryptografické algoritmy, ktoré vyžadujú kľúč. Fernet definuje 256 kľúč, ktorý je rozdelený na dve 128 bitové časti. Prvá časť reprezentuje podpisový kľúč a druhá šifrovací kľúč.

Existuje iba jedna verzia tokenu, teda tú máme danú. Pre vygenerovanie tokenu, potrebujeme zaznamenaý aktuálny čas do časovej pečiatky a vygenerovať náhodný reťazec, ktorý bude slúžiť ako IV. Následne zašifrujeme telo tokenu pomocou šifrovacieho kľúča a IV. Ďalej vypočítame HMAC z predchádzajúcich častí tokenu pomocou podpisového kľúča. Nakoniec a všetky časti spojíme do jedného reťazca a zakódujeme pomocou textitbase64url [18].

Validácia tokenu spočíva v dekódovaní tokenu, rozdelení na časti a overení časovej pečiatky a HMAC. Potom dešifrujeme telo toknu pomocou šifrovacieho kľúča a IV. Následne prípadne overíme časovú plastnosť tokenu, ak telo obsahuje potrebné informácie pre overenie časovej platnosti ako vznik a doba platnosti tokenu.

2.4 Branca

(Áno naozaj tento token vznikol ako párodia na "lepší"fernet a teda Fernet-Branca). Motiváciou k vzniku Branca tokenu bolo modernizovanie Fernet tokenu. Branca má

podobnú štruktúru aj generovanie a validáciu ako Fernet. Branca sa najmä líši v tom, že využíva šifrovaciu a podpisovú funkciu XChaCha20-Poly1305 AEAD [1].

2.4.1 Štruktúra Branca

Branca sa podobne ako Fernet skladá z piatich zreťazených častí. Tieto časti sa však jemne líšia od častí Fernet tokenu a zakódované sú *base62* kódovaním [12].

- Verzia - reprezentuje verziu Branca tokenu, aktuálne existuje len jedna verzia a je reprezentovaná číslom 0xBA. Zaberá vždy 8 bitov.
- Časová pečiatka - reprezentuje čas vytvorenia tokenu. Čas je zachytení ako počet sekúd od 1.1.1970 v UTC časovej zóne. Zaberá vždy 32 bitov a je zapísaná ako číslo bez znamienka.
- Nonce (Čiraz :D) - reprezentuje náhodný reťazec znakov, ktorý využíva šifrovacia funkcia. Ide vpodstate o IV, ale využíva sa naozaj len raz, zatiaľčo IV sa v blokovej šifre využije viac krát. Zaberá vždy 192 bitov.
- Zašifrované telo - reprezentuje zašifrované dáta uložené v tokene. Môže mať ľubovoľnú dĺžku.
- Hešovaný autentifikačný kód - reprezentuje výstup funkcie Poly1305 a zaberá vždy 128 bitov.

2.4.2 Generovanie a validácia Branca

Vygenerujeme 192 bitový reťazec znakov, ktorý bude slúžiť ako nonce a zachytíme aktuálny čas do časovej pečiatky. Vyrobíme hlavičku zreťazením verzie, časovej pečiatky a nonce. Následne zašifrujeme telo tokenu pomocou šifrovacej funkcie, do ktorej vložíme tajný kľúč, nonce a ako dodatočnú informáciu použijeme hlavičku. Funkcia vráti zašifrované telo a hešovaný autentifikačný kód vypočítaný aj z hlavičky. Nakoniec všetky časti spojíme do jedného reťazca a zakódujeme pomocou *base62* kódovania [12].

Pri validácií tokenu ho ako prvé dekódujeme. Následne overíme, že prvý bajt je 0xBA a rozdelíme na jednotlivé časti. Dešifrujeme telo a overíme hešovaný autentifikačný kód pomocou fukncie XChaCha20-Poly1305 AEAD. Následne môžeme overiť napríklad časovú platnosť tokenu pomcou dodatočných informácií v tele tokenu a časovej pečiatky.

Kapitola 3

Teoretické porovnanie API tokenov

V tejto kapitole porovnáme teoretické parametre rôznych typov API tokenov ako bezpečnosť, škálovateľnosť, jednoduchosť implementácie a flexibilita.

Kapitola 4

Návrh a implementácia jednoduchého rozhrania

V tejto kapitole navrhneme a implementujeme jednoduché rozhranie s použitím viecerých API tokenov pre autentifikáciu a autorizáciu.

Kapitola 5

Porovnanie na jednoduchom rozhraní

V tejto kapitole porovnáme jednotlivé API tokeny na nami implementovanom rozhraní. Výsledkom budú naše pozorovania pri implementacii a použití rozhrania s jednotlivými typmi API tokenov.

Záver

Literatúra

[1] S. Arciszewski. Aead xchacha20 poly1305. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-xchacha-03.

[2] S. Arciszewski. Paseto specification. https://github.com/paseto-standard/paseto-spec.

[3] S. Arciszewski. Paseto: Platform-agnostic security tokens, Apríl 2018. https://paseto.io/rfc/.

[4] Richard Barnes. Use Cases and Requirements for JSON Object Signing and Encryption (JOSE). RFC 7165, Apríl 2014.

[5] Fallible. We reverse engineered 16k apps, here's what we found, Január 2017. https://hackernoon.com/we-reverse-engineered-16k-apps-heres-what-we-found-51bdf3b456bb#.io6e11q6n.

[6] Sheila Frankel, K. Robert Glenn, and Scott G. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, September 2003.

[7] Sheila Frankel and Scott G. Kelly. Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec. RFC 4868, May 2007.

[8] Harold Giménez. Fernet legacy specification. https://github.com/heroku/legacy-fernet.

[9] Harold Giménez. Fernet specification. https://github.com/fernet/spec/blob/f16a35d3cfd8cdb2d8c7f7d10ce6c4d6058b19d2/Spec.md.

[10] Weili Han, Zhigong Li, Minyue Ni, Guofei Gu, and Wenyuan Xu. Shadow attacks based on password reuses: A quantitative empirical analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(2):309–320, 2018.

[11] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, 2012.

[12] Kejing He, Xiancheng Xu, and Qiang Yue. A secure, lossless, and compressed base62 encoding. In *2008 11th IEEE Singapore International Conference on Communication Systems*, pages 761–765, 2008.

[13] Michael Jones. JSON Web Algorithms (JWA). RFC 7518, Máj 2015.

[14] Michael Jones. JSON Web Key (JWK). RFC 7517, Máj 2015.

[15] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Signature (JWS). RFC 7515, Máj 2015.

[16] Michael Jones, John Bradley, and Nat Sakimura. JSON Web Token (JWT). RFC 7519, Máj 2015.

[17] Michael Jones and Joe Hildebrand. JSON Web Encryption (JWE). RFC 7516, Máj 2015.

[18] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, Október 2006.

[19] Rodney Lorrimar. Haskell fernet implementation. `https://github.com/IamAmitE/FernetCpp/`.

[20] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[21] Thomas Pornin and Julien P. Stern. Digital signatures do not guarantee exclusive ownership, 2005. `http://www.bolet.org/~pornin/2005-acns-pornin+stern.pdf`.

[22] Agathoklis Prodromou. Tls security 6: Examples of tls vulnerabilities and attacks, Marec 2019. `https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/`.

[23] PYCA. Python cryptography library. `https://github.com/pyca/cryptography`.

[24] N. Sakimura, NRI, J. Bradley, Ping Identity, Microsoft, M. Jones, B. de Medeiros, Google, and C. Mortimore. Openid connect core 1.0 incorporating errata set 1, November 2014. `https://openid.net/specs/openid-connect-core-1_0.html#IDToken`.