Kriptografske metode autentifikacije i autorizacije

Karlović, Ratomir

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

RATOMIR KARLOVIĆ

Kriptografske metode autentifikacije i autorizacije

Diplomski rad

Pula, 2018.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

RATOMIR KARLOVIĆ

Kriptografske metode autentifikacije i autorizacije

Diplomski rad

JMBAG: 0303046143, redoviti student

Studijski smjer: Informatika

Predmet: Kriptografija

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: dr.sc. Siniša Miličić

Pula, listopad 2018.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Ratomir Karlović, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 30. listopada, 2018. godine

IZJAVA

o korištenju autorskog djela

Ja, Ratomir Karlović dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom Kriptografske metode autentifikacije i autorizacije koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 30. listopada, 2018. godine

Potpis

_____

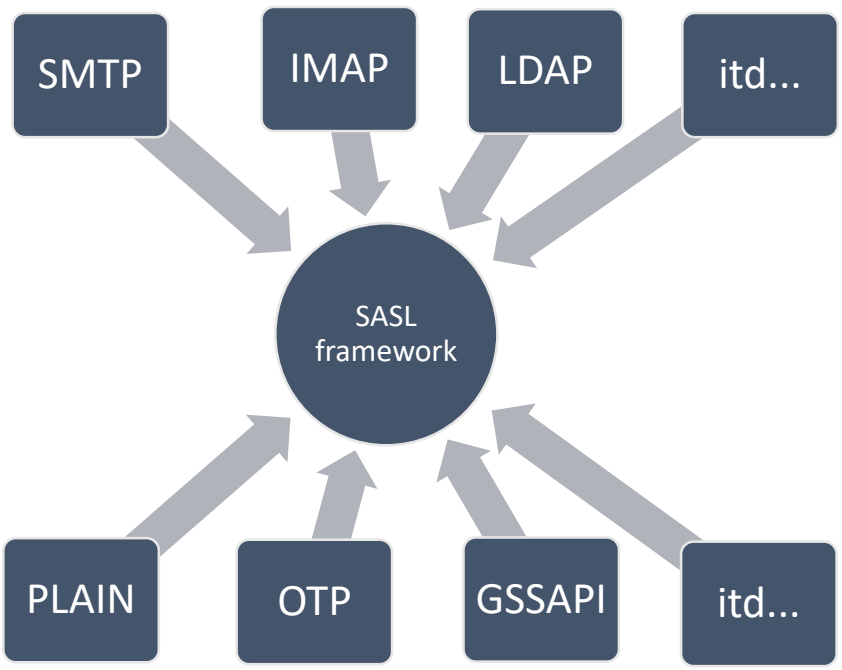
Sadržaj

1. Uvod

U ovom radu pisat će se o kriptografskim metodama autentikacije i autorizacije koje su kompatibilne sa SASL razvojnom cjelinom (*eng. framework*). Autentifikacija ili autentikacija je proces određivanja identiteta korisnika prilikom pristupanja nekom servisu, a autorizacija je proces utvrđivanja radnje koje korisnik može raditi. Specifičnost SASL frameworka je u tome što podržava rad s puno različitih metoda, ali njegova najveća prednost je u tome što je modularan i lako izmjenjiv za razliku od sličnih sigurnosnih programa. U prvom dijelu rada govorit će se o SASL frameworku i njegovim pravilima koja se moraju poštovati. Svi programi koji su povezani s kriptografijom moraju imati strogo definirana pravila i osmišljena rješenja za različite scenarije koji se mogu dogoditi. Nakon SASL frameworka govorit će se o SASL mehanizmima to jest metodama koje se koriste. Počinje se s najjednostavnijim mehanizmom Plain, koji je također najranjiviji, a završava se s kompleksnijim mehanizmima koji pružaju veću sigurnost kao OTP i SCRAM. U radu se ne govori samo o opisu mehanizama već i o njihovim prednostima i manama, kojim napadima podliježu te što se treba napraviti kako bi se obranili od takvih napada. Poslije toga dolazi praktična implementacija OAuth i OTP metode za koje se može reći da su danas najkorištenije i najsigurnije kriptografske metode autentikacije i autorizacije. Također, demonstrirat će se rad SCRAM-SHA1 autentikacije između servera i klijenta. Cilj implementacije je pokazati kako bez korištenja alata kao što je to Passport.JS, Firebase, Twillio i sličnih je moguće implementirati SASL autentikaciju. Za implementaciju SASL mehanizama u ovom radu je potrebno znanje JavaScript jezika, osnovne HTML, smartphone i korištenje Node.JS. U zaključku će biti izneseni zaključci o kriptografskim metodama autentikacije i autorizacije koji su nastali tijekom izrade rada.

2. Jednostavna autentikacija i sigurnosni sloj

Jednostavna autentikacija i sigurnosni sloj ili The Simple Authentication and Security Layer je framework koji omogućuje autentikaciju i servise za sigurnost podataka u konekcijski orijentiranim protokolima preko zamjenjivih mehanizama. SASL pruža strukturirano korisničko sučelje između protokola i mehanizma. Framework isto tako pruža protokol za osiguravanje slijeda izmjene protokola unutar sloja za sigurnost podataka. Sloj za sigurnost podataka može pružati integritet podataka, povjerljivost podataka i ostale servise. SASL dizajn je namijenjen da omogućava korištenje novih protokola sa postojećim mehanizmima bez potrebe za redizajniranjem istih. Također dopušta postojećim protokolima korištenje novih mehanizama bez potrebe za redizajniranjem protokola. SASL framework stvara apstraktni sloj između protokola i mehanizama (slika 1.).



Slika 1. - SASL framework

Izvor: Prilagođeno prema Melnikov A. i K. Zeilenga, *Simple Authentication and Security Layer (SASL)*, 2006., Dostupno na: RFC-Editor, (pristupljeno 21. kolovoza 2018.)

Pomoću korisničkog sučelja apstraktnog sloja je omogućeno da bilo koji protokol može koristiti bilo koji mehanizam. Generalno ovaj sloj sakriva pojedinosti protokola mehanizama i pojedinosti mehanizama od protokola, ali ne sakriva pojedinosti mehanizama od implementacije protokola. Različiti mehanizmi zahtijevaju različite informacije kako bi mogli pravilno raditi – neke od tih informacija koriste autentikaciju šifrom, druge koriste Kerberos "tickets", certifikate, itd. Kako bi se napravila autorizacija, kod implementacije servera generalno se koristi mapiranje identiteta između autentikacijskih identiteta čija forma je specifična za mehanizme i autorizacijskih identiteta čija forma je specifična za aplikacijski protokol.

Kako bi se koristio SASL, svaki protokol pruža metodu za identificiranje mehanizma koji će se koristi, metoda za razmjenu server-specific server-challenges i client-responses, i metodu za komuniciranje rezultata razmjene autentikacije. Svaki SASL mehanizam definira seriju serverskih izazova i klijentskih odgovora koji pružaju server autentikaciju i pregovora servise za sigurnost podataka.[1]

2.1. Koncept identiteta

U praksi autentikacija i autorizacija mogu uključiti više identiteta, moguće u više formi (jednostavno korisničko ime, Kerberos princip[2], X.500 Distinguished name[3] itd.) s različitim reprezentiranjem (ABNF-opisani UTF-8 enkodirani Unicode znakovi, BER-enkodirana imena). Tehnička dokumentacija često propisuje formu za identifikaciju i prikazivanje na mreži, ali različite forme za identifikaciju i/ili prikazi se koriste pri implementaciji. Kako se identiteti različitih formi uobičajno odnose jedni prema drugima je definirano lokalno. Forme i prikazi koji su se također koristili tijekom implementacije su lokalni.[4]

[1] Melnikov A. i K. Zeilenga, *Simple Authentication and Security Layer (SASL)*, 2006., str. 1, Dostupno na: RFC-Editor, (pristupljeno 21. kolovoza 2018.).
[2] Koristi proizvoljan broj elemenata za stvaranje korisničkog identiteta. Elementi se odvajaju s "/". Zadnji element je *realm*, koji većinom sadrži ime domene koje je napisano velikim slovima i odvojen je znakom "@".
[3] Za stvaranje korisničkog identiteta potrebno je koristiti šest elemenata. Puno ime, ime odjela unutar organizacije, ime organizacije. ime grada, ime države i dva slova države (ISO 3166-1 alpha-2).
[4] ibidem, str. 5.

SASL framework konceptualno koristi dva identiteta:

- Identitet asociran sa autentikacijom certifikata -> identitet autentikacije,
- Identitet za ponašanje -> identitet autorizacije.

SASL specifikacije mehanizma opisuju akreditacijske forme (Kerberos ticket, jednostavni korisničko ime/šifra, X.509 certifikat, itd.) koje se koriste pri autentikaciji klijenta uključujući sintaksu i semantiku autentikacijskih identiteta koji se prenose unutar akreditacijske forme kada je to potrebno. SASL specifikacije protokola opisuju forme za identitete koje se koriste pri autentikaciji i propisuje sintaksu i semantiku za nizove znakova (*eng. charachter string*) koji se koriste za predstavljanje i slanje autentikacijskih identiteta uz pomoć mehanizma.
Klijent pruža svoju akreditaciju (koja sadržava autentikacijski identitet) i opcionalno niz znakova (*eng. character string*) koji predstavlja autentikacijski identitet kao dio SASL izmjene. Kada string izostaje ili je prazan, klijent traži da se ponaša kao identitet koji se nalazi u akreditaciji (npr. korisnik zahtijeva da se ponaša kao autentikacijski identitet).
Osim što je server zadužen za verifikaciju klijentske akreditacije, on također mora provjeriti da li identitet koji je asociran s klijentskom akreditacijom ima dopuštenje da se ponaša kao autorizacijski identitet. SASL razmjena nije uspješna ako bilo koja ili obje verifikacije nisu uspješne (SASL razmjena također može biti neuspješna zbog drugih razloga).[5]

2.2. Razmjena autentikacije

Svaka razmjena autentikacije sastoji se od poruke klijenta serveru gdje se zahtijeva autentikacija preko određenog mehanizma. Nakon toga slijedi jedan ili više parova „izazova“ od servera koji zatim čeka odgovor klijenta koji će nakon toga primiti poruku od servera gdje će se indicirati rezultat razmjene autentikacije (autentikacija također može

[5] loc. cit.

biti napuštena.). U nastavku je prikana razmjena autentikacije na visokoj razini. Slovo K označava klijenta, a S server.

K: Zahtijeva razmjenu autentikacije

S: Započinje izazov

K: Započinje odgovaranje

<dodatni izazovi/odgovori>

S: Rezultat razmjene autentikacije

Ako je rezultat uspješan i sigurnosni sloj je dogovoren, onda je ovaj sloj postavljen. Neki mehanizmi specificiraju da prvi podaci koji se šalju u razmjeni autentikacije su od klijenta na server. Protokoli možda pružaju dodatne mogućnosti tj. mogu skratiti proces.

K: Zahtijeva razmjenu autentikacije + šalje prvi odgovor

<dodatni izazovi/odgovori>

S: Rezultat razmjene autentikacije

Kada mehanizam specifično specificira da prvi podaci koji se šalju potječu od klijenta, ali ako je polje nedostupno ili neiskorišteno, zahtjev klijenta sadrži prazan izazov. [6]

K: Zahtijeva razmjenu autentikacije

S: Prazan izazov

K: Započinje odgovor

<dodatni izazovi/odgovori>

S: Rezultat razmjene autentikacije

U slučaju da se koristi mehanizam koji ne dozvoljava klijentu da šalje podatke prvi, razmjena autentikacije neće biti uspješna. Neki mehanizmi specificiraju da server mora

[6] ibidem, str. 6-7.

poslati dodatne podatke klijentu kada indiciraju da je autentikacija uspješna. Protokoli mogu pružiti mogućnost opcionalnog dodavanja podatkovna polja u poruci gdje bi se nalazili dodatni podaci koje određeni mehanizmi zahtijevaju. Ako mehanizam radi na takvom principu, protokol će pružiti dodatno polje za podatke za odlazeću poruku sa servera te će server zatim iskoristiti to polje i skratiti proces.

K: Zahtijeva razmjenu autentikacije

S: Započinje izazov

K: Započinje odgovaranje

<dodatni izazovi/odgovori>

S: Rezultat razmjene autentikacije uz dodatne podatke za uspjeh

Kada mehanizam specificira da server mora vratiti dodatne podatke klijentu s uspješnim obavljanjem autentikacije, ali to je polje nedostupno ili neiskorišteno, dodatni podaci se šalju kao „izazov“ čiji odgovor je prazan. Nakon što server primi ovakav odgovor, server indicira uspješan rezultat.[7]

K: Zahtijeva razmjenu autentikacije

S: Započinje izazov

K: Započinje odgovaranje

<dodatni izazovi/odgovori>

S: Dodatna razmjena podataka „izazova“

K: Prazan odgovor

S: Rezultat razmjene autentikacije

Ukoliko mehanizam specificira da prvi podaci koji se razmjenjuju dolaze od klijenta i šalju se dodatni podaci klijentu s indikacijom uspješnog obavljanja autentikacije. Ako protokol

[7] loc. cit.

podržava sve navedene radnje, cijeli proces izmjene će se svesti na dva kruga izmjene podataka.[8]

K: Zahtijeva razmjenu autentikacije + šalje dodatne odgovore

<dodatni izazovi/odgovori>

S: Rezultat razmjene autentikacije s dodatnim informacijama u slučaju uspjeha

Umjesto:
K: Zahtijeva razmjenu autentikacije

S: Prazan izazov

K: Započinje odgovaranje

<dodatni izazovi/odgovori>

S: Dodatni izazovi

K: Prazan odgovor

S: Rezultat razmjene autentikacije

2.3. Imenovanje mehanizama

Ime SASL mehanizama je niz znakova od 1 do 20 znakova dužine koji se sastoje od ASCII velikih slova, brojki, crtica i/ili podcrtane linije.

Rad/komunikacija mehanizama je određena protokolom koji se koristi. Uobičajeno, protokol specificira da server reklamira klijentu dostupne mehanizme koje taj protokol koristi. Tada će klijent odabrati „najbolji“ mehanizam koji može koristiti s liste koju je dobio od servera. Ova komunikacija nije zaštićena prethodno navedenim metodama pa je zbog toga podložna napadima ukoliko se ne zaštiti nekim drugim načinima. Kako bi se detektirao napad, protokol može dodati zaštitu za integritet podataka kada se odabire

[8] loc. cit.

mehanizam za korištenje kako bi klijent mogao otkriti promjene na listi mehanizama dostupnih na serveru. Ovakvim napadima cilj je dodati mehanizam koji server ne podržava pa u slučaju da klijent odabere upravo taj mehanizam, napadač bi došao do informacija koje klijent želi poslati na server.[9]

2.4. Zahtjev za autentikacijom

Zahtjev za autentikaciju šalje klijent kada traži autentikaciju preko nekog specificiranog mehanizma. Klijent šalje poruku koja sadrži ime mehanizma serveru. Pojedinosti poruke ovise o specifikacijama protokola. Ime mehanizma nije zaštićeno samim mehanizmom pa je zbog toga na meti napada ako integritet nije zaštićen drugim načinima. Kasnije ovisno o tome kako je mehanizam definiran, klijent šalje prvi podatke serveru.

Mehanizam je definiran tako da omogućuje klijentu slanje podatke i zahtjev poruke protokola, koji sadrži opcionalno polje za odgovor. Klijent u njemu može uključiti odgovor na prvi upit u poruci autentikacijske razmjene.[10]

2.5. Izazovi i odgovori

Razmjena autentikacije uključuje jednu ili više interakcija između servera koji postavlja izazov i klijenta koji šalje odgovore na zadane izazove. Ova interakcija je također definira mehanizmom. Odgovori i izazovi su obuhvaćeni unutar poruke protokola, koja je definirana samim protokolom. Kroz izazove i odgovore mehanizam može:

- autenticirati klijent serveru,
- autenticirati server klijentu,
- poslati autorizacijski identitet preko stringa,
- dogovoriti sigurnosti sloj,
- ponuditi ostale usluge.

[9] ibidem, str. 7.
[10] ibidem, str. 8.

Dogovaranje sigurnosnog sloja može uključivati pregovaranje oko sigurnosnih usluga koje će biti dostupne u sloju te definiranje maksimalne veličine zaštićenog teksta kojeg svaka strana može primiti. Nakon što klijent primi izazov, klijentski mehanizam može poslati odgovor ili zaustaviti razmjenu.[11]

2.6. Autorizacijski string za identifikaciju

Autorizacijski string za identifikaciju je sekvenca koja se sastoji od nula ili više unicode znakova, bez NUL(U+0000) znaka, koji predstavljaju identitet koji klijent može koristiti. U slučaju da autorizacijski string za identifikaciju nedostaje, klijent traži identitet kojeg će server asocirati pomoću klijentske akreditacije. Prazan string je ekvivalentan praznom stringu za identifikaciju. Ukoliko string nije prazan, to indicira da klijent želi dobiti identitet koji je predstavljen pomoću stringa. U tom slučaju, forma identiteta koja se nalazi u stringu, kao i sintaksa i semantika stringa, je određena prema protokolu. Dok je enkodiranje koje se koristi za transfer autorizacijskog stringa za identifikaciju specificiran mehanizmom. Od mehanizma se također očekuje da može prenijeti cijeli unicode sadržaj.[12]

2.7. Prekid razmjene autentikacije

Klijent ili server mogu odlučiti prekinuti razmjenu autentikacije ukoliko više ne mogu nastaviti ili započeti razmjenu (npr. nemaju sve podatke koji su potrebni za nastavak) ili žele zaustaviti razmjenu (npr. zbog toga što se druga strana lažno predstavlja). Klijent može zaustaviti razmjenu slanjem poruke (pojedinosti poruke ovise o protokolu koji se koristi) serveru gdje indicira da je razmjena zaustavljena. Zbog protokola server će možda morati odgovoriti na poruku klijenta o prekidu razmjene autenfikacije. Također, server

[11] ibidem, str. 9.-10.
[12] ibidem, str. 10.

može prekinuti razmjenu autentikacije slanjem poruke čije pojedinosti su definirane korištenim protokolom. Poruka mora indicirati da je razmjena autenfikacije prekinuta.[13]

2.8. Rezultati autentikacije

Na kraju razmjene autentikacije server šalje poruku sa pojedinostima klijentu u kojoj su vidljivi rezultati autentikacije.

Rezultat nije uspješan ukoliko:

- je razmjena autentikacije zbog bilo kojeg razloga prekinuta;
- klijentsku akreditaciju nije moguće provjeriti;
- server ne može asocirati identitet pomoću klijentske akreditacije;
- identitet asociran s klijentskom akreditacijom nije autoriziran da se ponaša kao što taj identitet zahtijeva;
- dogovoreni sigurnosni sloj nije definiran ili dovoljno dobar;
- server ne želi pružiti uslugu zbog bilo kojeg razloga.

Protokoli mogu sadržavati dodatna podatkovna polja (*eng. data field*) u ovoj poruci. Ta polja mogu biti unutar poruke jedino kao dodatna polja ukoliko je autentikacija uspješna. Ukoliko je rezultat uspješan i sigurnosni sloj dogovoren, tada se taj sloj implementira/instalira. U slučaju da rezultat nije uspješan ili sigurnosni sloj nije dogovoren, svi slojevi koji su uspostavljeni do tada ostaju. Poruka kojom server daje do znanja koji je rezultat autentikacije, također, pomaže klijentu da razlikuje greške koje se mogu riješiti ponovnim slanjem klijentske akreditacije, greške koje se riješavaju pokušavanjem kasnije i greške koje zahtijevaju da klijent kontaktira sistem administratora za rješenje. Ovakav pristup je naročito koristan tijekom redovnog održavanja servera jer smanjuje troškove za korisničkom podrškom. Također je važno da se server može

[13] loc. cit.

konfigurirati tako da se poslana poruka neće razlikovati između važećeg korisnika s nevažećom akreditacijom i nevažećeg korisnika.[14]

2.9. Sigurnosni slojevi

SASL mehanizmi mogu pružiti široku lepezu sigurnosnih usluga u sigurnosnim slojevima. Tipične usluge sastoje se od integriteta podataka (*eng. data integrity*) i povjerljivosti podataka (*eng. data confidentality*). SASL mehanizmi koji ne pružaju sigurnosni sloj tretiraju se kao da sigurnosni sloj nije dogovoren. Ukoliko je sigurnosni sloj dogovoren u protokolu kod razmjene autentikacije, sloj je instaliran od strane servera nakon što potvrdi rezultat razmjene autentikacije, a klijent ga instalira nakon primanja rezultata autentikacije. U oba slučaja, sloj se instalira prije prijenosa daljnjih protokol podataka. Točna pozicija nakon koje sigurnosni sloj ima efekt u protokolu je definirana samim protokolom. Jednom kada sigurnosni sloj počne raditi u protoku podataka unutar protokola on nastavlja raditi sve dok veza nije zatvorena ili novi sigurnosni sloj nije dogovoren. Dok sigurnosni sloj radi, on obrađuje podatke unutar protokola u zaštićene podatke. Ukoliko u bilo kojem trenutku sigurnosni sloj ne može ili ne želi nastaviti obradu podataka, transfer podataka mora biti zatvoren. Ako sigurnosni sloj ne može dekodirati dobivene podatke veza mora biti prekinuta isto kao u prethodnom slučaju. U oba slučaja veza bi trebala biti prekinuta na primjeren način. Svaki paket podataka nakon što je obrađen šalje se uspostavljenom vezom kao sekvenca okteta na čijem početku se nalazi polje sa četiri okteta, koje je sukladno s konvencijom o rasporedu bajtova u mreži, koji predstavlja dužinu paketa (*eng. buffer*). Dužina paketa zaštićenih podataka ne smije biti veći od dužine koja druga strana očekuje. Ukoliko se dogodi da primatelj primi takav paket, trebao bi prekinuti vezu jer ovo može biti znak napada. Maksimalna veličina je fiksirana mehanizmom, tako da je veličina dogovorena ili je specificirana samim mehanizmom.[15]

[14] ibidem, str. 11.
[15] ibidem, str. 12.

2.10. Višestruka autentikacija

Ukoliko to nije eksplicitno dopušteno u protokolu, što će biti napisano u tehničkoj dokumentaciji samog protokola, samo jedna uspješna razmjena SASL autentikacije se može napraviti u sesiji protokola. U tom slučaju kada je razmjena autentikacije uspješno završena svi daljnji pokušaji za inicijalizacijom razmjene autentikacije će biti bezuspješna. Ukoliko je višestruka SASL razmjena autentikacije dopuštena u protokolu onda SASL sigurnosni slojevi ne smiju raditi istovremeno. Ukoliko je dopuštena višestruka autentikacija i nakon dogovora je selektiran drugi sloj, onda će u tom slučaju drugi sloj zamijeniti prvi. U slučaju da je višestruka autentikacija dopuštena i nije odabran niti jedan sigurnosni sloj, onda će originalni sigurnosni sloj biti odabran. Kada su dopušteni višestruki SASL pregovori unutar protokola, utjecaj neuspješne razmjene autentikacije nakon prethodno uspostavljene autentikacije i autorizacije specifičan je za svaki protokol. Tehnička specifikacija protokola bi trebala objasniti da li će prethodna autentikacija i autorizacija ostati na snazi ili će ono biti promijenjeno. Bez obzira na specifikacije protokola, prethodno dogovoreni sigurnosni sloj će ostati aktivan.[16]

[16] ibidem, str. 13.

3. Zahtjevi protokola

Ukoliko neki protokol želi pružati SASL usluge on mora poštovati određene specifikacije tj. njegova specifikacija mora pružati obavezne informacije. Protokol mora imati ime usluge kako bi mogao biti selektiran iz registra usluga za Generic Security Service Application Program Interface (GSSAPI) host-based forme za imena usluga. Ovaj registar se dijeli između GSSAPI i SASL mehanizama. Protokol mora pružiti detalje bilo kojeg mehanizma koji radi pregovaranje koje protokol pruža.

Protokol bi trebao specificirati način na koji klijent može otkriti, prije inicijalizacije razmjene i poslije instaliranja sigurnosnog sloja koji je dogovoren nakon razmjene, imena SASL mehanizama koji su dostupni klijentu na tom serveru. Ovo je važno imati poslije instaliranja sigurnosnog sloja zbog otkrivanja downgrade napada.

Definicija svih poruka potrebnih za razmjenu autentikacije mora biti specificirana protokolom. Ona mora sadržavati poruku koja započinje razmjenu autentikacije. Ova poruka mora sadržavati polje namijenjeno za ime mehanizma kojeg je klijent odabrao. Poruka bi trebala sadržavati opcionalno polje za prvi odgovor klijenta. Ako je poruka definirana s ovakvim poljem onda se mora specificirati kako će se razlikovati slučaj kada klijent pošalje prazno polje i kada klijent uopće ne pošalje poruku. Ovo polje mora biti sposobno prenijeti sekvencu okteta.

Svaka poruka koja se desi u komunikaciji kada server postavlja izazove, a klijent odgovora mora biti sposobna prenijeti sekvencu okteta[17].

Poruka koja indicira rezultat autentikacije bi trebala imati opcionalno polje za prenošenje dodatnih podataka s uspješnim rezultatom razmjene autentikacije. U slučaju da je poruka definirana s ovakvim poljem onda se mora specificirati kako će se razlikovati poruke sa praznim dodatnim poljem od poruka bez dodatnih podataka. Ova poruka također mora podržavati prenošenje sekvence okteta.[18]

[17] 1 oktet = 1 byte, 1 byte = 1 bit
[18] ibidem, str. 13.-14.

Sintaksa i semantika string varijabli za identifikaciju mora postojati kako bi se izbjegli problemi interoperabilnosti zbog različitih normalizacija. Moraju biti točno definirani kada i gdje će stringovi biti obrađeni, uključujući sve normalizacije (npr. SASLPrep, provjera da li se koriste zabranjeni znakovi), zbog usporedbi i drugih funkcionalnosti kako bi se osigurala ispravna funkcionalnost. Savjetuje se da se koriste već postojeće forme za identitete i postojeće reprezentacije za stringove.

Svaka opcija protokola koja omogućava klijentu i/ili serveru da prekinu razmjenu autentikacije mora biti detaljno opisana. Protokoli koji podržavaju višestruku autentikaciju uobičajeno dozvoljavaju klijentu da prekine razmjenu autentikacije koja je u tijeku tako da inicijalizira novu razmjenu autentikacije. Protokoli koji podržavaju višestruku autentikaciju možda će tražiti od klijenta da zatvori vezu sa serverom i počne iz početka kako bi prekinuo autentikaciju u tijeku. Protokoli tipično dozvoljavaju serveru da prekine razmjenu autentikacije koja se odvija tako da vrate poruku s neuspješnim rezultatom autentikacije.

Potrebno je identificirati gdje će točno novo dogovoreni sigurnosni sloj početi djelovati kod klijenta i servera. Uobičajeno je da sigurnosni sloj počne raditi sa prvim oktetom koji server šalje klijentu i sa prvim oktetom kojeg klijent pošalje serveru.

Ukoliko protokol podržava druge sigurnosne slojeve kao što je to Transport Layer Security (TLS), mora se specificirati redoslijed kojim se sigurnosni slojevi primjenjuju na podatke u protokolu. Za protokol koji podržava TLS i SASL sigurnosne slojeve, specifikacija bi mogla izgledati kao jedan od navedenih primjera: [19]

- SASL sigurnosni sloj se uvijek primjenjuje prvi na podatke koji se šalju pa zbog toga se primjenjuje zadnji na primljene podatke;
- SASL sigurnosni sloj se uvijek primjenjuje zadnji na podatke koji se šalju pa zbog toga se primjenjuje prvi na primljene podatke;
- Slojevi se primjenjuju u redoslijedu kojim su oni instalirani;
- Slojevi se primjenjuju u obrnutom redoslijedu u kojem su instalirani;
- TLS i SASL sigurnosni sloj ne mogu biti implementirani u isto vrijeme.

[19] ibidem, str. 15.

Specifikacija mora pokazati da li protokol podržava višestruku autentikaciju. Ukoliko podržava, onda protokol mora detaljno opisati efekt neuspjele razmjene autentikacije na prethodno uspostavljene autentikacije i autorizacijska stanja.

Specifikacija protokola bi trebala izbjegavati govoriti o zahtjevima implementacija koje bi ometale zamjenu mehanizama. Generalno, specifikacija protokola bi trebala biti neutralna prema mehanizmima. Postoje neke iznimke ovoj preporuci. Ne treba detaljizirati detalje kako se akreditacija (što je specifično za svaki mehanizam) koristi unutar protokola. Ne treba iznositi detalje kako se autentikacijski identiteti (što je specifično za svaki mehanizam) i autorizacijski identiteti (što je specifično za svaki protokol) ponašaju jedan prema drugome. Također bi se trebalo izbjegavati detaljno govoriti o mehanizmima koji se mogu koristiti unutar određenog protokola.[20]

[20] ibidem, str. 16.

4. Zahtjevi mehanizma

Svi SASL mehanizmi moraju u svojoj specifikaciji imati određene informacije. Svaki mehanizam mora imati ime koje mora biti registrirano. Mehanizam mora imati definirane izazove servera i odgovore klijenta kod razmjene autentikacije.

Mehanizam mora naznačiti da li je mehanizam klijent na prvom mjestu (*eng. client-first*), varijabilni ili server na prvom mjestu (*eng. server-first*). U slučaju da je SASL mehanizam definiran kao klijent na prvom mjestu koji ne pošalje inicijalni odgovor u zahtjevu za autentikaciju onda prvi izazov koji server pošalje mora biti prazan (EXTERNAL mehanizam je primjer ovakvog mehanizma). Ako je SASL mehanizam definiran kao varijabilni, onda specifikacije moraju definirati kako se server ponaša kada je klijentov odgovor izostavljen (DIGEST-MD5 mehanizam je primjer takvog mehanizma). Kada je SASL mehanizam definiran kao server na prvom mjestu, onda klijent ne smije poslati inicijalni odgovor u autentikacijskom zahtjevu za razmjenu (CRAM-MD5 mehanizam je primjer ovakvog mehanizma, o njemu više u 8. poglavlju).

Mehanizam mora indicirati da li se od servera očekuje da pošalje dodatne podatke kada je autentikacijska razmjena uspješna. Ukoliko server šalje dodatne podatke onda se, ako se oni šalju kao dodatni izazov, u specifikaciji mora naznačiti da je odgovor na ovaj izazov prazan.

SASL mehanizmi bi trebali biti dizajnirani tako da smanje broj izazova i odgovora potrebnih za izvršenje zamjene autentikacije.

Treba naznačiti da li je mehanizam sposoban prenijeti stringove za autorizacijske identitete. Dok stari mehanizmi nisu sposobni prenijeti autorizacijske identitete (za ove mehanizme, autorizacijski identiteti su uvijek prazan string). Mehanizmi koji su nedavno definirani bi trebali biti sposobni prenijeti autorizacijski string. Mehanizmi ne bi smjeli nikad ne moći prenijeti string za autorizacijske identitete ili prazan autorizacijski identitet.[21]

[21] loc. cit.

Mehanizmi koji su sposobni prenijeti string za autorizacijski identitet moraju biti sposobni prenijeti praznu sekvencu Unicode znakova, bez onih koji sadrže NUL znak. Mehanizmi bi trebali koristiti UTF-8 transformacijski format. Specifikacije mehanizma moraju objasniti kako bilo koji Unicode kod, specifičan za mehanizam, koji se može pojaviti u stringu za autorizacijski identitet izbjegava dvosmislenost tijekom dekodiranja stringa za autorizacijski identitet. Uobičajeno mehanizam koji ima posebne znakove zahtijeva da se ti posebni znakovi izbjegavaju ili da se enkodiraju u znakove stringa (nakon što se enkodira u specifični Unicode transformacijski format) koristeći shemu za enkodiranje podataka kao što je to Base64. Specifikacija mehanizma mora opisati da li mehanizam nudi sigurnosni sloj. U slučaju da mehanizam ima sigurnosni sloj, onda on mora biti detaljno opisan skupa s ostalim uslugama koje su ponuđene unutar sigurnosnog sloja i kako se te usluge mogu implementirati. Ukoliko osnovna kriptografska tehnologija koju mehanizam koristi podržava integritet podataka, onda mehanizam mora zaštititi prijenos autorizacijskom identiteta i pregovaranje sigurnosnog sloja.

Kako bi se izbjegla interoperabilnost zbog različitih normalizacija, kada mehanizam poziva znakovne podatke (ne odnosi se na string za autorizacijski identitet) da se koriste kao unos za kriptografsku funkciju i/ili funkciju za uspoređivanje, specifikacija mora detaljno opisati kako i kada (klijent ili server) se znakovni podaci pripremaju, uključujući i normalizaciju, za unos u funkciju kako bi se osigurala ispravna operacija. Za jednostavna korisnička imena i/ili šifre u autentikacijskim akreditacijama, SASLprep[22] bi trebao biti specificiran kao algoritam za pripremu podataka. Mehanizmi ne bi trebali koristiti string za autentikacijski identitet u generiranju bilo kojih dugoročnih kriptografskih ključeva ili hasheva jer nije obavezno da string za autentikacijski identitet treba biti standardiziran. Dugoročno u ovom kontekstu znači trajanje duže od razmjene autentikacije u kojem su oni generirani. Različiti klijenti (od istog ili različitog protokola) mogu pružiti različite stringove za autorizacijske identitete koji su semantički jednaki. Korištenje stringova za autentikacijski identitet u generiranju kriptografskih ključeva najvjerojatnije će dovesti do interoperabilnosti i drugih problema.

[22] ibidem, str. 17.-18.

5. Razmatranje sigurnosti

Mnogi postojeći SASL mehanizmi ne pružaju dovoljnu zaštitu protiv pasivnih napada, a kamoli aktivnih napada u razmjeni autentikacije. Mnogi postojeći mehanizmi uopće ne nude sigurnosne slojeve. Pretpostavlja se da će budući SASL mehanizmi pružati jaku zaštitu protiv pasivnih i aktivnih napada u razmjeni autentikacije i sigurnosne slojeve s jakom osnovnom zaštitom podataka kao što je to integritet podataka i povjerljivost podataka. Također se pretpostavlja da će u budućnosti mehanizmi imati napredniju zaštitu podataka kao što je to re-keying.

SASL framework je podložan na downgrade napade. Postoje različiti načini s kojima se oni mogu spriječiti ili detektirati no više o tome u potpoglavlju 5.2. U određenim slučajevima je prikladno koristiti zaštitu podataka koju SASL ne pruža, kao što je to TLS, za zaštitu protiv downgrade napada na SASL. Korištenje vanjskih usluga za zaštitu podataka je važno kada dostupni mehanizmi ne nude adekvatnu zaštitu integriteta i/ili povjerljivosti podataka razmjene autentikacije i/ili podataka protokola. U nastavku će biti navedeni aktivni napadi koji se događaju.[23]

5.1. Hijack napad

Kada klijent odabere SASL sigurnosni sloj minimalno sa zaštitom integriteta podataka, takva zaštita je dovoljna kao protu-mjera protiv aktivnog napada otmice veze i preoblikovanje podataka protokola koji se šalju nakon uspostavljenog sigurnosnog sloja. Implementacija bi trebala zatvoriti vezu kada sigurnosna usluga u SASL sigurnosnom sloju prijavi manjak integriteta u podacima protokola.[24]

5.2. Downgrade napadi

Jako je važno da se bilo koji sigurnosno osjetljivi pregovori protokola izvršavaju nakon instalacije sigurnosnog sloja sa zaštitom integriteta podataka. Protokol bi trebao biti

[23] ibidem, str. 18.
[24] ibidem, str. 19.

dizajniran tako da pregovaranje koje je napravljeno prije ove instalacije treba biti pregledano nakon što se instalacija završi zbog toga što su pregovaranja kod SASL mehanizama sigurnosno osjetljivi. Kada klijent pregovara autentikacijski mehanizam za server i/ili druge sigurnosne usluge, moguće je da aktivni napadač promijeni listu mehanizama koju server prezentira ili listu mehanizama koju klijent šalje u svom odgovoru. Kako bi se zaštitilo protiv ovakvih napada, implementacija ne bi trebala reklamirati mehanizme i/ili usluge koje ne zadovoljavaju njihove minimalne sigurnosne uvjete. Ne treba započeti ili nastaviti razmjenu autentikacije koja ne zadovoljava minimalne sigurnosne uvjete. Također treba provjeriti da li je završena razmjena autentikacije rezultat sigurnosnih servisa koji zadovoljavaju minimalne sigurnosne uvjete. Svaku od prethodno navedenih stavki treba posebno provjeriti da li zadovoljava sigurnosne uvjete. Kako bi se detektirao downgrade napad kod manje sigurnih podržanih sigurnosnih mehanizama, klijent može otkriti SASL mehanizme koje server nudi prije SASL razmjene autentikacije i poslije razmjene pregovora SASL sigurnosnog sloja (koji minimalno podržava zaštitu integriteta podataka) koji je već instaliran kroz mehanizam protokola. Ako klijent otkrije da lista sa zaštićenim integritetom, koju je dobio nakon što je sigurnosni sloj instaliran, sadržava jači mehanizam od onih na prethodnoj listi koju je dobio, klijent treba pretpostaviti da je prethodno dobivena lista modificirana od strane napadača i treba zatvoriti vezu sa serverom jer je ona ugrožena. Klijentska inicijalizacija SASL zamjene, uključujući odabir mehanizma nije zaštićena i zbog toga je podložna napadima od strane aktivnih napadača. Jako je važno za bilo koji novi SASL mehanizam da bude dizajniran tako da aktivni napadači ne mogu napraviti uspješan napad modificiranjem imena SASL mehanizama i/ili izazova i odgovora zbog autentikacije sa slabim sigurnosnim svojstvima. Višerazinski sigurnosni pregovori su podložni downgrade napadima. Kod dizajniranja protokola treba izbjegavati pružanje sigurnosnih pregovaranja na višoj razini o sigurnosnim uslugama protokola kao što su pregovori iznad SASL mehanizama. Također treba izbjegavati pregovore na nižim razinama tj. svi pregovori koji su ispod SASL mehanizama.[25]

[25] loc. cit.

5.3. Replay napad

Neki mehanizmi mogu biti podložni replay napadima. Takvi mehanizmi nisu zaštićeni eksternim sigurnosnim uslugama za zaštitu podataka. Primjer eksterne sigurnosne usluge za zaštitu podataka je TLS.[26]

5.4. Truncation napad

Većina postojećih SASL sigurnosnih slojeva ne pružaju zaštitu protiv truncation napada. U truncation napadu aktivni napadač prouzroči prekid sesije protokola, prouzrokujući skraćivanje/oštećivanje podataka (*eng. truncation*) sa zaštićenim integritetom što može uzrokovati da jedan ili oba protokola (klijent i server) daju beneficije napadaču. Ovakvi napadi su relativno lagani za obraniti unutar aplikacijske razine (*eng.application-level*) protokola koji su orijentirani prema konekciji. Protokol se može obraniti od ovakvih napada tako da se pobrine da svaka sesija ima mehanizam za zatvaranje sesije te da imaju zaštitu integriteta podataka.[27]

5.5. Ostali aktivni napadi

Kada je sigurnosni sloj dogovoren preko protokola razmjene autentikacije, primatelj bi se trebao adekvatno ponašati ako dobije pakete sa zaštićenim podacima veće veličine od definirane tj. dogovorene maksimalne veličine. U tom slučaju ne treba alocirati dovoljnu količinu memorije za taj paket jer se može desiti „out of memory“ stanje. Ako primatelj detektira velike blokove, onda treba zatvoriti vezu.[28]

[26] ibidem, str. 20.
[27] loc. cit.
[28] loc. cit.

5.6. Pasivni napadi

Puno mehanizama je podložno različitim pasivnim napadima. Pasivni napad može uključivati jednostavno prisluškivanje nesigurnih akreditacijskih informacija. Također, može i uključivati online i offline napade na rječnike zaštićenih akreditacijskih informacija.[29]

5.7. Re-keying

Sigurno ili administrativno dopušteno trajanje životnog ciklusa sigurnosnog sloja SASL mehanizma je ograničeno tj. ima konačno vrijeme. Kriptografski ključevi slabe što se više koriste i što su stariji. Što više vremena i/ili šifriranog teksta kriptoanaliza ima nakon prvog korištenja ključa, lakše je napraviti napad na ključ.

Administrativna ograničenja na životnom ciklusu sigurnosnog sloja može imati formu vremenskih ograničenja izraženim X.509 certifikatom, Kerberos V kartom ili direktorijima, što je često i poželjno. U praksi, čest efekt administrativnog ograničenja životnog ciklusa je taj da aplikacija može pronaći sigurnosni sloj prestane li raditi tijekom aplikacijskih operacija protokola, npr. prilikom transfera velikih količina podataka. Kao rezultat ovoga, veza će biti prekinuta što će kao posljedicu imati loše korisnički iskustvo.

Re-keying (proces prepoznavanja ključa) je način adresiranja oslabljenih kriptografskih ključeva. SASL framework ne pruža re-keying, ali SASL mehanizmi mogu. Dizajneri budućih SASL mehanizama trebaju uzeti u obzir pružanje re-keying usluga.

Implementacije koje žele napraviti re-keying na SASL sigurnosnim slojevima gdje mehanizmi ne pružaju re-keying trebaju ponovno autenticirati isti ID i zamijeniti zastarjeli ili uskoro zastarjeli sigurnosni sloj. Ovaj pristup zahtjeva podršku za ponovnu autentikaciju u aplikacijskom protokolu (višestruka autentikacija).[30]

[29] loc. cit.
[30] ibidem, str. 21.

5.8. Ostali aspekti

Dizajneri protokola i implementatori trebaju razumjeti sigurnosne aspekte mehanizama kako bi mogli odabrati mehanizam koji zadovoljava njihove potrebe.

Kod distribuirane serverske implementacije treba biti oprezan koliku razinu povjerenja se daje drugoj strani. Posebno se to odnosi na autentikacijske tajne koje se jedino povjeravaju strani za čiju sigurnost se ne sumnja i vjeruje se da će ona upravljati i koristiti te tajne na prihvatljiv način strani koja joj je dala pristup tim tajnama. Aplikacije koje koriste SASL pretpostavljaju da SASL sigurnosni sloj pruža zaštitu integriteta podataka koja je sigurna pod napadima kada napadač odabere tekst koji je pod zaštitom sigurnosnog sloja. Slično tome, aplikacije pretpostavljaju da je SASL sigurnosni sloj siguran i onda kada napadač može manipulirati sa šifriranim tekstom koji je nastao kao rezultat sigurnosnog sloja. Od novih SASL mehanizama se očekuje da zadovoljavaju ovakva očekivanja.

Zbog toga što Unicode sadržava veliki broj znakova i sadržava znakove različitih pisama, nepravilno korištenje takvih znakova može ostaviti sustav ranjivim na napade. Upravo zbog toga se trebaju poštovati Unicode sigurnosni aspekti. Oni se primjenjuju na stringove za autentikacijski identitet. Isto vrijedi i za UTF-8 sigurnosne aspekte kada se koristi UTF-8 te kada se koriste SASLprep i StringPrep.[31]

U sljedećem poglavlju govorit će se o IANA aspektima.

[31] loc. cit.

6. Registar SASL mehanizama

Registar SASL mehanizama održava IANA tj. Internet Assigned Numbers Authority. Registar se trenutno nalazi na službenim stranicama IANA pod SASL mehanizma. Svrha ovog registra nije jedino da održava jedinstvenost vrijednosti koje se koriste za imenovanje SASL mehanizama, već i da pruža reference za tehničku specifikaciju koja detaljno opisuje svaki SASL mehanizam dostupan na internetu za korištenje. Ne postoji konvencija za imenovanje SASL mehanizama. Bilo koje ime koje poštuje sintaksu SASL mehanizma može biti registrirano.[32]

6.1. Procedura registracije imena mehanizma

IANA će registrirati novo ime SASL mehanizma prema principu „First Come First Server“, kao što je definirano u BCP 26. To znači da ukoliko su svi aspekti ispoštovani, mehanizam koji je prvi registriran dobit će željeno ime, a ako se pojavi mehanizam s istim imenom koji je također ispunio sve uvjete, on će morati promijeniti ime jer se registrirao kasnije. IANA ima pravo odbiti lažne registracijske zahtjeve, ali neće napraviti pregled napravljenih tvrdnji u registracijskoj formi. Registracija SASL mehanizma se zahtijeva slanjem sljedeće forme:

Predmet: Registracija SASL mehanizma X

Ime SASL mehanizma (ili prefiks njegove obitelji):

Sigurnosni aspekti:

Objavljena specifikacija (Preporučeno):

Adresa osobe i elektronička adresa za daljnju komunikaciju:

Namjena korištenja; (Jedna od: Opća, Ograničeno korištenje ili Zastarjelo)

Vlasnički/Change controller:

[32] ibidem, str. 22.

Bilješke/Komentari: (Bilo koja informacija koju autor smatra bitnom se dodaje ovdje)

Nakon što je registracijska forma pravilno ispunjena ona se šalje preko elektroničke adrese u IANA na njihovu službenu adresu iana@iana.org. Ova registracijska procedura ne zahtjeva pregled od strane visokokvalificirane osobe, autore SASL mehanizama se potiče da traže od zajednice da pregleda i komentira njihov mehanizam kad god je to moguće. Autori mogu tražiti mišljenje zajednice tako što će objaviti specifikacije protokola kojeg žele predložiti kao Internet-Draft. SASL mehanizam koji je namijenjen za široku upotrebu trebao bi biti standardiziran kroz normalnu IETF proceduru kada za to bude vrijeme. [33]

6.2. Procedura registracije imena obitelji mehanizma

Kao kod registracije imena mehanizma, isto tako ne postoji generalna konvencija za imenovanje. Međutim, specifikacije mogu imati rezervirani dio imena SASL mehanizma čija namjena je za lakše prepoznavanje sličnih SASL mehanizama, „obitelj“ SASL mehanizama. Svaka obitelj SASL mehanizama se prepoznaje po jedinstvenom prefiksu, kao što je to X-. Registracija nove obitelji SASL mehanizma zahtijeva pregled visoko kvalificiranog stručnjaka, kao što je to definirano u BCP 26. Registracija SASL obiteljskog imena zahtijeva ispunjavanje sljedeće forme:

Predmet: Registracija imena obitelji SASL mehanizma X

SASL ime obitelji (ili prefiks za obitelj):

Objavljena specifikacija (preporučeno):

Namjena korištenja; (Jedna od: Opća, Ograničeno korištenje ili Zastarjelo):

Osobna adresa i elektronička adresa za daljnju komunikaciju:

Promjena vlasnika(*eng. change controller*):

[33] ibidem, str. 23.

Bilješke/Komentari: (Bilo koja informacija koju autor smatra bitnom se dodaje ovdje)

Kada je forma ispunjena ona se šalje na elektroničnu adresu od IETF SASL na ietf-sasl@imc.org i elektroničnu adresu IANA na iana@iana.org. Nakon dva tjedna testiranja zajednice, visoko kvalificirani stručnjak odlučuje da li je zahtjev za registraciju imena primjeren i nakon toga odlučuje da li će odobriti ili odbiti zahtjev tako što će poslati obavijest autoru, IANA i listi elektroničkih adresa.

Pregled bi se trebao fokusirati na pogodnosti predloženog imena obitelji i prikladnost predloženog imena i registracijskog plana za postojeće i buduće mehanizme koji bi se svrstali pod tu obitelj. Opseg pregleda može zahtijevati povezane aspekte koje pruža bilo koja tehnička specifikacija, kao što je to IANA Considerations sekcija. Međutim pregled je usko fokusiran na prikladnost zahtjeva za registraciju, a ne na pruženu tehničku specifikaciju. Autori se potiču na traženje mišljenja zajednice tako da objave tehničku specifikaciju kao Internet-Draft i traženje komentara na IETF listi elektroničkih adresa.[34]

6.3. Komentari o registraciji SASL mehanizma

Komentari u registraciji SASL mehanizma ili SASL obitelji trebaju prvi biti poslani vlasniku mehanizma ili obitelji i/ili na mailing listu tj. na ietf-sasl@imc.org. Komentatori mogu nakon razumnog pokušaja kontaktiranja vlasnika, zatražiti od IANAe da pripiše njihov komentar registraciji SASL mehanizma. Nakon toga IANA odlučuje želi li ili ne želi pripisati komentar u registraciju SASL mehanizma.[35]

6.4. Promjena vlasnika

Jedno kada je registracija SASL mehanizma objavljena od strane IANA, autor može zahtijevati promjenu definicije. Zahtjev za promjenu ima istu proceduru kao procedura za registraciju.

[34] ibidem, str. 24.
[35] ibidem, str 25.

Vlasnik SASL mehanizma može svoje odgovornosti za SASL mehanizam prenijeti na drugu osobu ili agenciju tako da informira IANA. Ovo se može napraviti bez prethodne rasprave ili pregleda. IESG može ponovno dodijeliti odgovornosti za SASL mehanizam. Najčešći slučaj kada se ovako nešto desi je kada treba odobriti promjene koje treba napraviti na mehanizmu, ali vlasnik je preminuo, nije ga moguće kontaktirati ili zbog nekog drugog razloga nije u mogućnosti napraviti promjene koje su važne za zajednicu.

SASL registracija mehanizma se ne može izbrisati, ali u slučaju kada se vjeruje da neki mehanizam više nije primjeren za korištenje njegovo se polje u registraciji za namjenu mijenja u Zastarjela (*eng. Obsolete*). Takav mehanizam će biti jasno označen na listi koju objavljuje IANA. IESG se smatra vlasnikom svih SASL mehanizama koji se nalaze na IETF. [36]

[36] loc. cit.

7. PLAIN SASL Mehanizam

Višenamjenske šifre otvorenog teksta (*eng. clear-text*) su jednostavne i kompatibilne skoro sa svim postojećim autentikacijskim bazama podataka operacijskih sustava i korisne su za nesmetan prijelaz na sigurnije autentikacijske mehanizme koji su bazirani na šiframa. Mana PLAIN SASL mehanizma je u tome što je on neupotrebljiv na internetu jer povjerljivost podataka nije osigurana. PLAIN mehanizam ne pruža sigurnosni sloj. Ovaj mehanizam se nikad ne bi trebao koristiti bez adekvatne zaštite podataka zbog toga što mehanizam ne pruža zaštitu integriteta podataka i zaštitu privatnosti podataka. Mehanizam je namijenjen korištenju sa zaštitom podataka aplikacijskog sloja protokola i uslugama TLSa.

U komunikaciji od klijenta do servera, mehanizam se sastoji od jedne poruke, UTF-8 stringa koji je enkodiran Unicode znakovima. Klijent prezentira autorizacijski identitet (identitet s kojim se želi ponašati), kojeg prati NUL (U+0000) znak, nakon kojeg dolazi autentikacijski identitet (identitet čija će šifra biti korištena), kojeg ponovno prati NUL znak nakon kojeg dolazi clear-text šifra. Kao i s drugim SASL mehanizmima, klijent ne daje autorizacijski identitet kada želi da server napravi identitet na osnovu akreditacije i koristi ga kao autorizacijski identitet.[37]

Formalna gramatika za klijentske poruke koristeći Augmented BNF glasi:

message = [authzid] UTF8NUL authcid UTF8NUL passwd

authcid = 1*SAFE; mora biti sposoban primiti do 255 okteta

authzid = 1*SAFE; mora biti sposoban primiti do 255 okteta

passwd = 1*SAFE; mora biti sposoban primiti do 255 okteta

UTF8NUL = %x00; UTF-8 enkodiran NUL znak

SAFE = UTF1 / UTF2 / UTF3 / UTF4; bilo koji UTF-8 enkodiran Unicode znak osim NUL

[37] Zeilenga K., *The PLAIN Simple Authentication and Security Layer (SASL) Mechanism*, 2006., str. 1.-2. Dostupno na: RFC-Editor, (pristupljeno 22. kolovoza 2018.).

Autorizacijski identitet (authzid), autentikacijski identitet (authcid), šifra (passwd) i NUL znak bit će transportirani kao UTF-8 enkodirani stringovi Unicode znakova. Kako se NUL znak koristi kao oznaka za početak i kraj izjave, on se ne smije pojaviti u authzid, authcid ili šifri.

Forma authzid je specifična i ovisi o aplikacijskoj razini protokola SASL profila. Authcid i passwd imaju slobodnu formu (*eng. form free)*. Korištenje nevidljivih znakova ili znakova koje korisnik ne može utipkati preko tipkovnice se ne podržava (iako ih je moguće unijeti). Serveri moraju biti sposobni prihvatiti authzid, authcid i passwd veličine do 255 znakova. UTF-8 enkodiranje Unicode znakova može biti dugo do 4 okteta.

Nakon primitka poruke, server će provjeriti dane podatke, koje je dobio u poruci, autentikacijski identitet (authcid u pseudo kodu) i šifru (passwd u pseudo kodu) s autentikacijskim sustavom baze podataka i nakon toga će provjeriti da li autentikacijska akreditacija dopušta klijentu ovlasti autorizacijskog identiteta (authzid u pseudo kodu). Ako su oba koraka uspješna, korisnik je uspješno autenticiran.

Stringovi autentikacijskog identiteta i string šifre, uključujući string autentikacijskog identiteta baze podataka i string šifre, moraju biti obrađeni prije nego što se iskoriste u verifikacijskom procesu. Profil SASLPrep od StringPrep algoritma je preporučen algoritam za obradu. SASLPrep je preporučeni algoritam jer poboljšava usporedbu i poboljšava usporedbu kako bi se ona izvršavala kako se to od nje očekuje. SASLPrep algoritam nije obavezan, pa je tako dopušteno koristiti druge algoritme ili ne koristiti niti jedan ovisno o situaciji. Na primjer, korištenje različitih algoritama je možda potrebno za uspješan rad servera s vanjskim sustavom.

Kada se obrađuje dobiveni string koristeći SASLPrep, dobivene stringove se tretiraju kao „query“ stringove i nedodijeljenim točkama koda je dopušteno pojaviti se u rezultatu. Kada se obrađuju stringovi baze podataka koristeći SASLPrep, string baze podataka se tretira kao „spremljeni“ string te je nedodijeljenim točkama koda zabranjeno pojavljivanje u njihovom rezultatu.

Bez obzira na algoritam koji se koristi, ako je rezultat bespovratne funkcije (npr. hash) nekog stringa spremljen, string mora biti obrađen prije nego što je unesen u tu funkciju.

Također bez obzira na koji se algoritam koristi, ako je obrada neuspješna ili rezultat je prazan string, verifikacija neće biti uspješna. Kada autorizacijski identitet nije poslan, server izvodi autorizacijski identitet na temelju pripremljenog reprezentacijskog stringa za autentikacijski identitet. To osigurava da izvođenje različitih reprezentacija autentikacijskog identiteta stvara isti autorizacijski identitet. Server može koristiti akreditaciju za inicijaliziranje bilo koje nove autentikacijske baze podataka, na primjer baze koje su prikladne za CRAM-MD5 ili DIGEST-MD5.[38]

7.1. Pseudo kod

Prikazani pseudo kod pokazuje verifikacijski proces, koji koristi hash šifru i SASLPrep funkciju, o kojem je bilo govora u prethodnoj sekciji. Ovaj pseudo kod nije konačan i ovisno o vremenu, može biti promijenjen.

```
boolean Verify(string authzid, string authcid, string passwd) {

string pAuthcid = SASLprep(authcid, true); # prepare authcid

string pPasswd = SASLprep(passwd, true);   # prepare passwd

if (pAuthcid == NULL || pPasswd == NULL) {

return false;     # preparation failed

}

if (pAuthcid == "" || pPasswd == "") {

return false;     # empty prepared string

}
```

Kod 1 - Prikaz verifikacijskog procesa

[38] ibidem, str. 3.-4.

```
storedHash = FetchPasswordHash(pAuthcid);
    if (storedHash == NULL || storedHash == "") {
      return false;     # error or unknown authcid
    }
    if (!Compare(storedHash, Hash(pPasswd))) {
      return false;     # incorrect password
    }
    if (authzid == NULL ) {
      authzid = DeriveAuthzid(pAuthcid);
      if (authzid == NULL || authzid == "") {
          return false; # could not derive authzid
      }
    }
    if (!Authorize(pAuthcid, authzid)) {
      return false;     # not authorized
    }
    return true;
  }
```

Kada je drugi parametar SASLprep funkcije istinit (true), onda to znači da su dozvoljene nedodijeljene točke koda u ulazu. Kada je SASLprep funkcija pozvana da obradi šifru to jest pripremi šifru za izradu hasha[39], drugi parametar mora biti lažan (false). Drugi parametar u Authorize se ne dobiva s ovim kodom. Treba se konzultirati sa SASL profilom aplikacijske razine kako bi odlučile kakve pripreme su potrebne i da li su one uopće potrebne. Također treba napomenuti da DeriveAuthzid i Authorize funkcije, bez obzira jesu li implementirane kao jedna ili dvije funkcije ovisno o tome kako su dizajnirane ili implementirane od strane mehanizma, zahtijevaju znanje i razumijevanje detaljne specifikacije implementacije mehanizama kako bi se ove funkcije implementirale. Rezultat Authorize funkcije je ovisan o detaljima lokalnog modela autorizacije i njegove politike. Obje funkcije mogu biti ovisne o drugim faktorima.[40]

7.2. Primjer

Prvi primjer pokazuje kako se PLAIN mehanizam može koristiti za autentikaciju.

Kod 2 - Prikaz korištenja Plain mehanizma

```
S: * ACAP (SASL "CRAM-MD5") (STARTTLS)

K: a001 STARTTLS

S: a001 OK "Begin TLS negotiation now"

< TLS pregovori, nastavak se odvija unutar TLS sloja >

S: * ACAP (SASL "CRAM-MD5" "PLAIN")

K: a002 AUTHENTICATE "PLAIN"

S: + ""

K: {21}

K: <NUL>tim<NUL>tanstaaftanstaaf

S: a002 OK "Authenticated"
```

[39] Hash se dobiva korištenjem matematičke funkcije kako bi se zaštitili podaci, tako da samo onaj kojem je namijenjena poruka može znati značenje poruke.
[40] ibidem, str. 5.

Na primjeru se može vidjeti da server i klijent uspostavljaju TLS vezu te se nakon toga vode TLS pregovori. Ukoliko TLS mora izvršiti dodatne korake onda se oni izvršavaju za vrijeme pregovora i poslije njih. Nakon što je klijent odabrao PLAIN mehanizam, klijent šalje autorizacijski identitet (21), kojeg prati NUL(U+0000) znak, nakon kojeg dolazi autentikacijski identitet (identitet čija će šifra biti korištena), kojeg ponovno prati NUL znak nakon kojeg dolazi clear-text šifra. Nakon toga server radi provjeru i ukoliko je ona dobra server vraća poruku o uspješnoj autentikaciji.

Sljedeći primjer pokazuje interakciju servera i klijenta kada klijent pošalje identitet drugog korisnika. Također u ovom primjeru se koristi protokol kako bi se smanjio broj potrebnih koraka i ubrzala autentikacija.[41]

```
S: * ACAP (SASL "CRAM-MD5") (STARTTLS)

C: a001 STARTTLS

S: a001 OK "Begin TLS negotiation now"

<TLS pregovori, nastavak se odvija unutar TLS sloja >

S: * ACAP (SASL "CRAM-MD5" "PLAIN")

C: a002 AUTHENTICATE "PLAIN" {20+}

C: Ursel<NUL>Kurt<NUL>xipj3plmq

S: a002 NO "Not authorized to requested authorization identity"
```

Kod 3 - Interakcija servera i klijenta

Na primjeru je moguće vidjeti da nakon što klijent pošalje poruku kako bi se identificirao, server šalje negativan odgovor nakon što je izvršio provjeru.

[41] ibidem, str. 6.

7.3. Sigurnosni aspekti

Kako PLAIN mehanizam ne pruža zaštitu integriteta i povjerljivosti podataka, on se ne bi trebao koristiti bez neke vanjske zaštite podataka koja će pružiti dovoljnu visoku razinu sigurnosti podataka ovisno o situaciji. Preporuča se korištenje servisa kao što je to TLS koji nudi puno aplikacijskih slojeva protokola. Generalno, PLAIN mehanizam nikad se ne bi trebao naći na listi mehanizama koje server šalje klijentu, ako se s tim mehanizmom ne koristi adekvatna zaštita podataka. Kada se koristi PLAIN mehanizam, server dobiva mogućnost da se predstavlja kao korisnik prema svim drugim servisima s istom šifrom bez obzira na enkripciju TLS ili nekog drugog mehanizma za zaštitu povjerljivosti podataka. Mnogi autentikacijski mehanizmi imaju sličnu slabost, no, bolji to jest jači SASL mehanizmi imaju rješenje za ovaj problem. Klijente se potiče da imaju operativni način rada gdje svi mehanizmi koji će najvjerojatnije pokazati šifru korisnika serveru budu onesposobljeni.[42]

[42] ibidem, str. 6.-7.

8. IMAP/POP AUTHorize ekstenzija za jednostavni izazov/odgovor

Postojeći predloženi standardi specificiraju AUTHENTICATE mehanizam kao IMAP4 protokol, a paralelni AUTH mehanizam za POP3 protokol. AUTHENTICATE mehanizam je napravljen da bude proširiv. Četiri metode koje koristi su sasvim jake i zahtijevaju podršku određene sigurnosne infrastrukture. Osnovna specifikacija POP3 također ima laki mehanizam za izazov-odgovor pristup koji se zove APOP. Njega se asocira s većinom rizika za takve protokole. On zahtijeva da klijent i server imaju pristup zajedničkim tajnama u formi čistog teksta. CRAM nudi metodu koja izbjegava korištenje spremljenog čistog teksta dok zadržava jednostavnog APOP algoritma koristeći jedino MD5. Trenutačno IMAP nema puno usluga koji se mogu usporediti s APOPom. Jedina alternativa jakom mehanizmu u IMAP-AUTH je korištenje čistog teksta za korisničko ime i šifre koje podržava naredba LOGIN u IMAPu.[43]

8.1 Challenge-Response Authentication Mechanism (CRAM)

CRAM-MD5 je mehanizam koji je sličan APOPu i PPP CHAPu. On se može koristiti s IMAPom i POP3em. Ovaj mehanizam ima prednosti nad alternativama zbog toga što ne zahtijeva od servera da čuva informacije o „prijavama“ elektroničke pošte prema prijavi. Iako mehanizmi koji zahtijevaju takav način rada se mogu smatrati sigurnijima, protokolima kao što je to IMAP, koji imaju nekoliko otvorenih veza istovremeno između klijenta i servera, ovakav način rada ne odgovara jer ima previše zahtjeva.

Podaci enkodirani u prvom odgovoru sadrže string s nasumičnim brojkama, vremenskom oznakom i ime potpuno kvalificiranog primarnog serverskog hosta. Sintaksa podataka koji nisu enkodirani moraju odgovarati pravilima POP3. Klijent uzima u obzir podatke koje je dobio i onda šalje svoj odgovor serveru sa stringom koji sadržava korisničko ime, razmak i digest. Digest se računa tako što se koristi MD5 algoritam gdje je ključ zajednička tajna i vremenska oznaka koja je dobivena sa prvom komunikacijom. Ključ je string koji je jedino poznat klijentu i serveru te je zbog toga zajednička tajna. Digest parametar je 16-oktetna

[43] Klensin J., Catoe R. i P. Krumviede, *IMAP/POP AUTHorize Extension for Simple Challenge/Response*, 1997., str. 1., Dostupno na: RFC-Editor, (pristupljeno 22. kolovoza 2018.).

vrijednost koja se šalje u heksadekadskom formatu koristeći male znakovi iz ASCIIa. Kada server zaprimi odgovor klijenta, on provjerava digest vrijednost koju je dobio. Ako je on točan, server smatra da je klijent odgovorio dobro i da je sada autenticiran. MD5 je izabran zbog toga što pruža bolji sigurnost kod autenticiranja s manjim porukama te zbog toga što eliminira spremanje tajni na serveru u čistom tekstu. MD5 čuva posredujuće rezultate koje se još naziva sadržajem (*eng. contexts*). CRAM ne podržava sigurnosni mehanizam.[44]

8.2. Primjer

Primjer pokazuje korištenje CRAM mehanizma s IMAP4 AUTHENTICATE naredbom (IMAP-AUTH). Za enkodiranje izazova i odgovora koristi se base64 koji je dio IMAP4 AUTHENTICATE naredbe, a ne dio CRAMa.

S: * OK IMAP4 Server

C: A0001 AUTHENTICATE CRAM-MD5

S: + PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+

C: dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw

S: A0001 OK CRAM autentikacija uspješna

U ovom primjeru, zajednička tajna je string „tanstaaftanstaaf“. Samim time digest se računa na sljedeći način:

MD5((tanstaaftanstaaf XOR opad),

MD5((tanstaaftanstaaf XOR ipad),

<1896.697170952@postoffice.reston.mci.net>))

[44] ibidem, str. 2.

Vrijednosti ipad i opad su definirani kao što je to navedeno u KEYED-MD5 te string koji je pokazan u izazovu je 1896.697170952@postoffice.reston.mci.net enkodiran s base64. Zajednička tajna je produžena na dužinu od 64 bajtova. Razlika u dužini se nadopunjuje nulama. Ukoliko je zajednička tajna duža od 64 bajta, MD5 digest zajedničke tajne se koristi kao 16 bajt ulaz za MD5 računanje.[45]

Rezultat računanja digest vrijednosti u heksadekadskom zapisu:

b913a602c7eda7a495b4e6e7334d3890

Ta vrijednost se potom enkodira s base64 kako bi se zadovoljili kriteriji IMAP4 AUTHENTICATE naredbe(ili slične POP3 AUTH naredbe), što će rezultirati izrazom:

dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw

8.3. Sigurnosni aspekti

Korištenjem CRAM autentikacijskog mehanizma dobiva se identitet podrijetla (*eng. origin identification*) i „replay“ zaštita prema sesiji. Server koji implementira naredbu sa čistim tekstom i ovaj način autentikacije, ne bi nikad trebao dopustiti korištenje obiju metoda jednom korisniku. Dok je spremanje konteksta na serveru marginalno bolje nego spremanje zajedničkih tajni u čistom tekstu kako to zahtijevaju CHAP i APOP, to nije dovoljno kako bi se zaštitile tajne ako je server ugrožen. Zbog toga server koji sprema tajne ili metapodatke mora biti zaštićen do razine vrijednosti tih podataka, štiti informacije o identitetima i elektroničkim sandučićima prema njihovoj potencijalnoj vrijednosti. CRAM-MD5 se često koristi kao zamjena to jest nadogradnja PLAIN mehanizma koji koristi čisti tekst. [46]

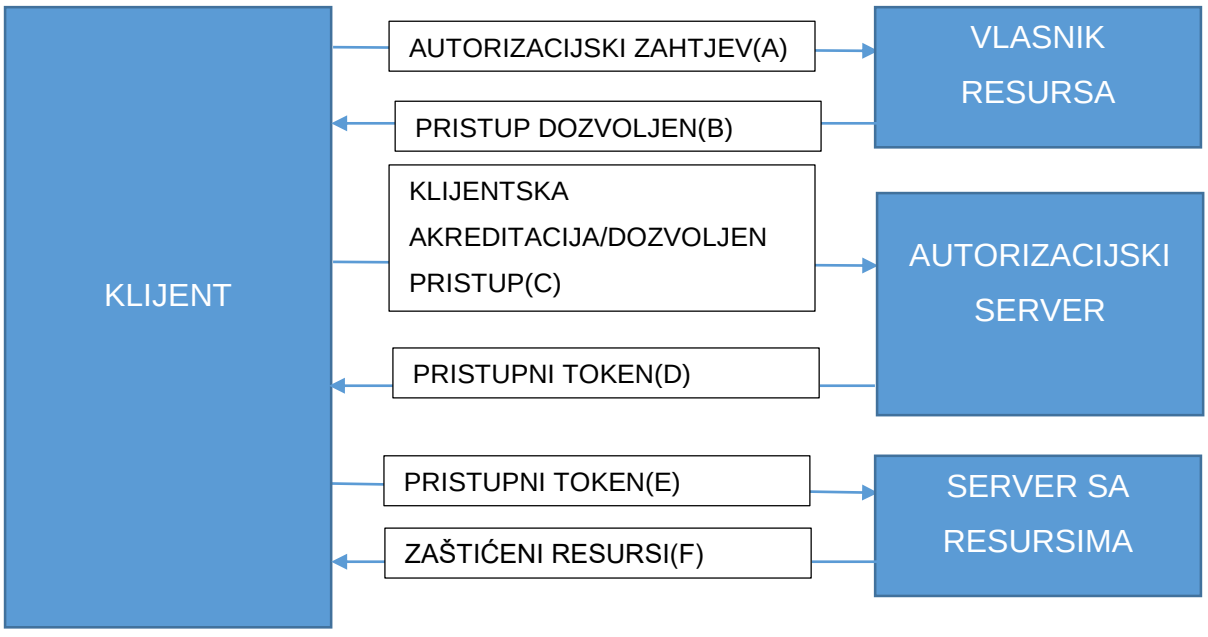
[45] ibidem, str. 3.
[46] ibidem, str. 4.-5.

9. OAuth

OAuth dozvoljava aplikaciji koja ne dolazi od strane klijenta ni servera, već treće strane, da dobije limitiran pristup zaštićenim resursima ili na osnovni vlasnika resursa tako što će dobiti dozvolu nakon interakcije ili dozvoljavanju aplikaciji koja dolazi od treće strane da dobije pristup u svoje ime.

Klijenti obično spremaju dugoročnu korisničku akreditaciju, međutim to vodi velikoj sigurnosnoj ranjivosti gdje na primjer može doći do curenja informacija. Velika prednost OAuth kod tih klijenata je u tome što je šifra zamijenjena zajedničkom tajnom s većom entropijom, npr. token. Token tipično pruža limitiran pristup i njime se može upravljati ili ga ukinuti odvojeno od korisnikove dugoročne šifre.

Na sljedećoj slici može se vidjeti kako radi OAuth kada je integriran unutar SASL frameworka.



Slika 2 - Prikaz rada OAuth mehanizma

Izvor: Prilagođeno prema Mills W., Showalter T. i H. Tschofenig, A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth, 2015., Dostupno na: RFC-Editor, (pristupljeno 24. kolovoza 2018.)

U prvom koraku (A), klijent traži autorizaciju od vlasnika resursa. Autorizacijski zahtjev može biti direktno poslan vlasniku resursa kao što je to pokazano ili indirektno preko autorizacijskog servera kao posrednika.

Klijent dobiva autorizacijska prava u drugom koraku (B) što je zapravo akreditacija koja predstavlja autorizaciju vlasnika resursa.

U sljedećem koraku (C) klijent traži pristupni token tako što se autenticira autorizacijskom serveru i pokazuje autorizacijska prava koja je dobio u drugom koraku od vlasnika resursa.

Klijent traži zaštićene resurse (D) od servera na kojem se oni nalaze i radi autentikaciju sa pristupnik tokenom.

U zadnjem koraku (F) server provjerava pristupni token i ako je on dobar, onda je autentikacija uspješna.[47]

9.1. Specifikacija OAuth SASL Mehanizama

SASL se koristio kao autentikacijski framework kod različitih aplikacijskih slojeva protokola. Dva takva mehanizma su OAUTHBEARER i OAUTH10A OAuth SASL mehanizma. OAUTHBEARER korisni TLS kako bi pružio sigurniju interakciju između klijenta i servera s resursima unutar protokola. OAUTH10A koristi OAuth 1.0a MAC tokene za razliku od OAUTHBEARER koji koristi OAuth 2.0 tokene.

Ovi mehanizmi ne pružaju zaštitu integriteta podataka niti njihovu povjerljivost kod slanja aplikacijskih poruka nakon autentikacije. Ukoliko je potrebna takva zaštita onda se treba koristiti TLS ili slična rješenja. Konkretno kod prethodno navedenog mehanizma, OAUTHBEARER mora koristiti TLS, a TLS se predlaže za korištenje kod implementaciji OAUTH10A. Oba mehanizma se inicijaliziraju nakon što klijent napravi prvi korak, gdje će server uvijek odgovarati na poruke klijenta.

[47] Mills W., Showalter T. i H. Tschofenig, *A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth*, 2015., str. 3.-5., Dostupno na: RFC-Editor, (pristupljeno 24. kolovoza 2018.).

Kada klijent ima i ispravno koristi valjani token procedura je jednostavna. Ona se odvija u dva koraka:

1. Klijent šalje valjan i ispravan prvi klijentski odgovor,
2. Server potvrđuje da je autentikacija uspješna.

U slučaju kada autentikacija nije uspješna, server šalje poruku s neispravnim rezultatom, nakon čega klijent mora poslati dodatnu poruku serveru kako bi dopustio serveru da završi razmjenu. Neki protokoli i uobičajene SASL implementacije ne podržavaju slanje SASL poruke i finaliziranje SASL pregovora. Dodatna klijentska poruka u slučaju pogrešne autentikacije služi za rješavanje ovog problema. Tada razmjena teče na sljedeći način:

1. Klijent šalje krivi prvobitni klijentski odgovor,
2. Server odgovara sa porukom greške,
3. Klijent šalje dodatnu poruku koja nema značenje (važne podatke),
4. Server zaustavlja autentikaciju.

9.2. Odgovaranje servera

Server provjerava odgovor ovisno o specifikacijama OAuth pristupnog tokena kojeg koristi. Server mora provjeriti cijeli odgovor od klijenta prije nego što generira serverski odgovor, ovo uključuje validacijske korake koji su navedeni u specifikacijama svakog OAuth pristupnog tokena. Međutim mogu se dodati dodatni validacijski koraci ukoliko je to potrebno ovisno o pojedinostima aplikacijskog protokola kojeg se koristi. Server odgovara uspješnom verificiranom klijentu tako što će završiti SASL pregovore. Autenticirani identitet kojeg je prijavio SASL mehanizam je identitet koji je sigurno napravljen za klijenta s OAuth akreditacijom. Aplikacija, ne SASL mehanizam, na osnovu politike osnovnog pristupa odlučuje da li je identitetu kojeg je prijavio mehanizam dozvoljen pristup zatraženim resursima.[48]

[48] ibidem, str. 8.

9.3. OAuth identifikatori unutar SASL konteksta

U OAuth frameworku klijent može biti autenticiran od strane autorizacijskog servera i vlasnik resursa se autenticira autorizacijskom serveru. OAuth pristupni token može sadržavati informacije o autentikaciji vlasnika resursa i o klijentu. Samim time on čini te informacije dostupnim serveru s resursima. Ako aplikacija traži oba identifikatora, programeri moraju pružiti način na koji će se obavljati komunikacija između SASL mehanizma i aplikacije.[49]

9.4. Sigurnosni aspekti

OAuth 1.0a i OAuth 2.0 dozvoljavaju različite varijante implementacijskih scenarija pa zbog toga sigurnosna svojstva variraju ovisno o kakvoj je implementaciji riječ. Zbog toga developeri trebaju dobro razumjeti sigurnosne potrebe na osnovi procjene prijetnji koja radi prije odabira specifičnog SASL OAuth mehanizma. OAUTHBEARER je mehanizam koji koristi OAuth 2.0 bearer token. On se oslanja na aplikacije koje koriste TLS kako bi zaštitio OAuth 2.0 bearer token razmjenu. Ukoliko se ne koristi TLS na aplikacijskom sloju, ova metoda je kompletno nesigurna. To je razlog zbog kojeg je obavezno korištenje TLSa kada se odabire ovaj autentikacijski mehanizam. S druge strane OAUTH10A je mehanizam koji koristi OAuth 1.0a MAC token. On podržava samo klijentsku autentikaciju. Ukoliko se želi imati serversku autentikaciju onda će ona biti pružena od aplikacije koja se nalazi ispod SASL sloja. U svakom slučaju preporučuje se korištenje TLSa.

Pristupni token nije jednak dugoročnoj korisničkoj šifri. Zbog toga treba obratiti pažnju kada se OAuth akreditacija koristi za radnje kao što je to promjena šifre. Server s resursima treba osigurati da radnje koje se odvijaju unutar autenticiranog kanala primjerene ovisno o jakosti akreditacije koja se koristi.

Moguće je da će se SASL koristiti za autenticiranje veze, te životni ciklus veze može nadživjeti ciklus pristupnog tokena koji je korišten za autenticiranje te iste veze. Ovo je

[49] ibidem, str. 9

uobičajeni problem u aplikacijskim protokolima gdje veza ima duži životni ciklus, a ne problem u mehanizmu. Server s resursima može isključiti klijenta ovisno o suglasnosti s aplikacijskim protokolom.

Smanjivanje životnog ciklusa pristupnog tokena daje bolje sigurnosne beneficije. OAuth 2.0 je predstavio token koji se sam obnavlja kako bi se dobio novi pristupni token bez potrebe za ljudsku interakciju. Prethodno dobiveni pristupni token može biti povučen ili označen kao nevažeći u bilo koje vrijeme. Klijent može zahtijevati novi pristupni token za svaku vezu prema serveru s resursima, ali trebao bi koristiti postojeću valjanu akreditaciju. [50]

[50] ibidem, str. 6.-7.

10. SecurID Mehanizam

SecurID mehanizam pruža dvo-faktorsku korisničku autentikaciju. Ovaj mehanizam nudi jedino autentikaciju, i nema nikakvog utjecaja na protokol enkodiranje i nije dizajniran za pružanje bilo kakve zaštite podataka. SecurID mehanizam je dobar izbor za situacije kada klijent, koji se ponaša kao korisnik, nije siguran. U tom slučaju jednokratna šifra će dati klijentu jednu priliku da napravi nešto štetno. Ovaj mehanizam jedino pruža autentikaciju.

Postoje tri entiteta u ovom autentikacijskom mehanizmu:

- Korisnik, koji posjeduje SecurID token,
- Aplikacijski server, na kojeg se korisnik želi povezati,
- Autentikacijski server, koji je sposoban autenticirati korisnika.

Mehanizam se zasniva na korištenju zajedničke tajne i osobnog identifikacijskog broja (PIN) kojeg znaju korisnik i autentikacijski server. Zajednička tajna je spremljena u token kojeg korisnik posjeduje i na autentikacijski server. Zbog toga se se i zove dvo-faktorska autentikacija jer je potreban fizički pristup tokenu i PIN kako bi se obavila autentikacija. [51]

10.1. Autentikacijski proces

Klijent prvo generira akreditaciju koristeći lokalne informacije (tajni ključ, trenutno vrijeme i PIN/šifra). Nakon toga ako protokol to dopušta klijent šalje akreditaciju serveru kao „početni“ odgovor. Inače klijent šalje poruku sa zahtjevom za autentikaciju te nakon što server odgovori na zahtjev, klijent šalje akreditaciju. Ukoliko server ne traži novi PIN, sadržaj poruke će imati autorizacijski identitet. Ukoliko je to polje prazno onda ono automatski dobiva zadanu vrijednost koja ga deklarira kao običnog korisnika. Ovo polje se najčešće koristi kada ga koriste sistemski administratori ili proxy serveri kojima treba drugačiji identitet od onog zadanog. Ovo polje ne smije biti duže od 255 okteta, mora završavati s NUL oktetom i mora se sastojati od UTF-8 znakova. Poruka također sadrži autentikacijski identitet i numeričku šifru. Autentikacijski identitet je identitet čija brojčana

[51] Nystrom M., *The SecurID(r) SASL Mechanism*, 2000., str. 1.-2., Dostupno na: RFC-Editor, (pristupljeno 24. kolovoza 2018.).

šifra će biti korištena, ukoliko je ovo polje prazno pretpostavlja se da je identitet poslan preko drugih sredstva, npr. preko nekih temeljnih protokola koji se koriste. Ovo polje također ne smije biti duže od 255 okteta, mora završavati s NUL oktetom i mora se sastojati od UTF-8 znakova. S druge strane numerička šifra je jednokratna šifra koja će se koristiti za dobivanje pristupa. Ovo polje ne smije biti kraće od četiri okteta i ne smije biti veće od 32 okteta, mora završavati s NUL oktetom i mora se sastojati od UTF-8 znakova. Numerička šifra se najčešće sastoji od 4-8 brojki. Kada server dobije sve navedene podatke, on verificira akreditaciju koristeći svoje vlastite informacije. Ukoliko je verifikacija uspješna server šalje povratnu poruku indicirajući uspješnu autentikaciju. Inače je verifikacije neuspješna ili server treba dodatnu akreditacijiu od klijenta kako bi autenticirao korisnika. Ukoliko server traži dodatne podatke klijent mora poslati novi PIN jer stari je postao nevažeći jer su svi PINovi jednokratki u SecurID mehanizmu. Klijent će ponovno generirati sve prethodno navedene podatke i poslati ih serveru. [52]

10.2. Sigurnosni aspekti

Ovaj mehanizam jedino pruža zaštitu protiv pasivnih napada prisluškivanja. Ne pruža zaštitu privatne sesije, serverske autentikacije ili zaštitu od aktivnih napada. Napadi gdje se napadač ponaša kao aplikacijski server kako bi došao do ispravne numeričke šifre su dosta česti kada se koristi ovaj mehanizam. Kako bi se zaštitilo od ovakvih napada klijent bi trebao biti siguran da je server propisno autenticiran. Kada god se šalje korisnikov PIN, korisnička autentikacija bi se trebala odvijati na serverskoj autenticiranoj vezi sa zaštićenom privatnosti. Serverska implementacija mora zaštiti protiv replay napada zbog toga što bi napadač inače mogao dobiti pristup tako što će se autenticirati kao korisnik jer je potrebne informacije saznao prisluškivanjem. Jedan od načina da se spriječe ovakvi napadi je da se zabrane višestruke istovremene autentikacijske sesije. To znači da jednom kada pravi korisnik pokrene autentikaciju, napadač će biti blokiran dok god prvi

[52] ibidem, str. 3.- 4.

autentikacijski proces nije završen. U ovom pristupu je potrebno koristi pauziranje kako bi se spriječio DoS napad (*eng. Denial of service attack*).[53]

[53] ibidem, str. 8.

11. One Time Password SASL Mehanizam

One Time Password ili OTP je koristan autentikacijski mehanizam za situacije kada postoji limitirano povjerenje u klijenta ili server. OTP mehanizam se koristi kao zamjena za SKEY SASL mehanizam koji se koristio u starijoj verziji SASLa. OTP je dobar izbor u scenarijima kada se radi s nepouzdanim klijentom, kao što je to kiosk. Jednokratka šifra će u tom slučaju dopustiti klijentu jednu mogućnost da se ponaša kao korisnik, isto kao prethodno opisani SecurID. OTP je također dobar izbor kada je riječ o situacijama gdje su interaktivne prijave serveru dozvoljene, jer OTP autentikacijska baza podataka je jedino ranjiva dictionary napadima, to jest pokušajima pogađanja šifre/PINa. Važno je napomenuti da za svako korištenje OTP mehanizma radi promjenu u autentifikacijskoj bazi podataka gdje se ažuriraju korisnički podaci. OTP i OTP-Ext (OTP s proširenim odgovorima) nude puno mogućnosti. Međutim da bi autentikacija bila uspješna, klijent i server moraju imati isti kompatibilni komplet opcija. Zbog toga treba poštovati sljedeća pravila:

- Obavezno je korištenje sintakse za proširene odgovore;
- Server mora podržavati četiri OTP proširena odgovora: „hex“, „word“, „init-hex“ i „init-word“. Server mora podržavati „word“ i „init-word“ odgovore zbog standardnog rječnika i trebao bi podržavati alternativne rječnike. Server ne smije tražiti dodatno korištenje OTP ekstenzija ili opcija;
- Klijent treba podržavati prikaz OTP izazova korisniku i unos OTP u više-riječni format. Klijent može podržavati direktan unos pass phrase (sekvenca riječi ili teksta koja se koristi za kontrolu pristupa, slična je šifri samo što je uobičajeno duža);
- Klijent mora naznačiti kada autentikacija nije uspješna zbog premale sekvence brojeva. Trebao bi ponuditi korisniku opciju za resetiranje sekvence s „init-hex“ ili „init-word“.[54]

[54] Newman C., 1998., *The One-Time-Password SASL Mechanism*, str. 1, Dostupno na: RFC-Editor, (pristupljeno 25. kolovoza 2018.).

11.1. OTP autentikacijski mehanizam

OTP mehanizam ne pruža nikakav sigurnosni sloj. Klijent počinje autenticiranje slanjem poruke serveru koja sadrži autorizacijski identitet i autentikacijski identitet. Kada je autorizacijski identitet prazan, onda dobiva unaprijed zadani identitet. Ovo se koristi od strane sistemskih administratora ili proxy servera za prijavu s drugačijim korisničkim identitetom. Ovo polje može biti dugačko do 255 okteta i mora završavati s NUL oktetom. Preferira se korištenje US-ASCII znakova, ali dopušteni su i UTF-8 znakovi za internacionalna imena. Korištenje drugih znakova osim US-ASCII i UTF-8 znakova je zabranjeno.

Autentikacijski identitet je identitet čiji će pass phrase biti korišten. Polje može biti dugačko do 255 okteta. Isto kao za autorizacijski identitet koriste se US-ASCII znakovi, ali dopušteni su i UTF-8 znakovi dok korištenje drugih znakova je strogo zabranjeno. Kada server primi ova dva podatka, on šalje poruku koja sadrži OTP izazov i OTP-Ext odgovor. Ukoliko klijent vidi nepoznato ime hash algoritma, on neće moći obraditi pass phrase od korisnika. U ovoj situaciji klijent može zatražiti format sa šest riječi, zatražiti prekid sekvence kao što je to specificirano u SASL profilu protokola koji se koristi i pokušati s drugim SASL mehanizmom ili zatvoriti vezu i odbiti autentikaciju. Server je ograničen na jedan OTP hash algoritam po korisniku.

Kada sve prođe uspješno, klijent generira proširenu odgovor u „hex“, „word“, „init-hex“ ili „init-word“ formatu. Klijent nije dužan završiti odgovor s razmakom ili novim redom i ne bi trebao koristiti bespotrebne razmake. Server mora tolerirati unos različite dužine ali autentikacije može biti neuspješna ukoliko dužina klijentskog ulaza prelazi razumnu veličinu.[55]

11.2. Sigurnosni aspekti

Ovaj mehanizam ne pruža privatnost sesije, serversku autentikaciju ili zaštitu od aktivnih napada. Mehanizam je podložan pasivnim dictionary napadima. Njih se može spriječiti s

[55] ibidem, str. 2.

biranjem dobrog pass phrasea. Serverska autentikacijska baza podataka koja koristi OTP ne smije biti u čistom tekstu. Serverska implementacija mora imati zaštitu protiv race napada. To su napadi kada napadač prisluškuje veći dio jednokratne šifre te pogađa ostatak šifre. Nakon čega se utrkuje (zbog toga su ovakvi napadi i dobili ime race attack) s pravim korisnikom kako bi se prvi autenticirao. Ovakav napad se može spriječiti s zabranom višestruke istovremene autentikacijske sesije isto kao kod SecurID mehanizma.[56]

[56] ibidem, str. 4.-5.

12. SCRAM

SCRAM ili Salted Challenge Response Authentication Mechanism je mehanizam čija svrha je rješavanje problema kod protokola kao što je to TLS gdje izazov-odgovor mehanizam nije dovoljno napredan za današnje standarde. SCRAM također nije samo mehanizam nego i obitelj mehanizama koji rade po izazov-odgovor principu. Kada se mehanizmi iz ove obitelji koriste u kombinaciji s TLS, ili sličnim sigurnosnim slojevima na istoj razini, poboljšavaju *status quo* aplikacijskih protokola za autentikaciju i pružaju prikladan izbor za obavezne mehanizme koji se trebaju implementirati za buduće standarde aplikacijskih protokola. Ova obitelj mehanizama ne pruža pregovoranje sigurnosnog sloja. Oni su namijenjeni da se koriste s vanjskim sigurnosnim slojem, kao što su to slojevi pruženi od TLS i SSH, s opcionalnim povezivanjem kanala s vanjskim sigurnim slojem. SCRAM pruža sljedeće prednosti:[57]

- Autentikacijske informacije koje su spremljene u autentifikacijskoj bazi podataka nisu dovoljne kako bi se oponašao klijent. Informacije su „salted“ (nasumična sekvenca okteta koja se kombinira sa šifrom koja je spremljena u autentifikacijskoj bazi podataka) tako da se spriječe dictionary napadi ukoliko je baza podataka ukradena;
- Server ne može oponašati klijenta kada komunicira s drugim serverima. Iznimka je serversko autorizirani proxy;
- Mehanizam dopušta korištenje serverske autorizacija proxya bez potrebe da proxy ima super-user prava back-end servera;
- Zajednička autentikacija je podržana, ali jedino klijent dobiva ime (server nema ime);
- Kada se koristi u sklopu SASL mehanizama, SCRAM je sposoban prenositi autorizacijske identitete od klijenta do servera.

[57] Newman C. et al., *Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms*, 2010., str. 4., Dostupno na: RFC-Editor, (pristupljeno 27. kolovoza 2018.).

12.1. SCRAM Algoritam

SCRAM ne sprječava klijenta da pošalje prvu poruku sa zahtjevom za SASL autentikacijom definiranom od strane aplikacijskog protokola ili slanje zadnje serverske poruke s dodatnim podacima o rezultatu SASL autentikacijske razmjene kojeg također definira aplikacijski protokol. SCRAM klijent posjeduje korisničko ime i šifru. On šalje korisničko ime serveru koji zatim pregledava odgovarajuće autentikacijske informacije. Nakon toga server šalje klijentu „salt“ i broj iteracija. Nakon dobivenog izračuna pomoću tih informacija, klijent šalje odgovor serveru kao dokaz autentikacije. Međuvremenu server je izračunao rezultat izračuna i kada klijent pošalje svoj rezultat on ih uspoređuje. Ukoliko je klijent poslao dobar rezultat, klijentska autentikacija je uspješna. Također klijent autenticira server pomoću vrijednosti kojih je dobio od servera. [58]

12.2. Sigurnosni aspekti

Ukoliko se autentikacijska razmjena izvršava bez jakog sigurnosnog sloja (kao što je to TLS s dobrom zaštitom privatnosti podataka), onda pasivni napadi, kao što je to prisluškivanje, mogu doći do dovoljno informacija kako bi napravili offline dictionary ili brute-force napad kako bi došli do korisničke šifre. Vrijeme potrebno za ovaj napad ovisi o kriptografskoj hash funkciji koja se koristi, jakosti šifre i broj iteracija (važno kod izračuna rezultata „izazova“) koje je server odredio. Jaki eksterni sigurnosni sloj s jakom enkripcijom je dovoljan za sprječavanje ovakvih napada.

Ukoliko eksterni sigurnosni sloj koji štiti SCRAM razmjenu koristi anonimnu razmjenu ključeva, tada se može koristiti povezivanje kanala koja je jedna od mogućnosti koje nudi SCRAM mehanizam. Tako bi se mogao detektirati napadač man-in-the-middle napad na sigurnosni sloj što može prouzročiti neuspjelu autentikaciju. Međutim prilikom ovog napada napadač može dobiti dovoljno informacija za brute-force ili offline dictionary napad. Zbog toga SCRAM dozvoljava povećanje broja iteracija kroz vrijeme. Broj iteracija

[58] ibidem, str. 7.-8.

se ne može povećati jednom kada je korisnik autenticiran. U tom slučaju treba resetirati korisnikovu šifru kako bi se povećao broj iteracija.

Ako su autentikacijske informacije ukradene iz autentikacijske baze podataka tada offline dictionary ili brute-force napad mogu doći do korisnikove šifre. Korištenjem „salt“ ovakav napad se oslabljuje zbog toga što je potrebno napraviti napad na svaku šifru. Najbolje rješenje za ovakav napad je korištenje mehanizama koji direktno brane od istih, na primjer SRP autentikacija i sustav razmjene ključeva.

Ukoliko napadač dođe do autentikacijskih informacija iz autentikacijskog repozitorija i prisluškuje autentifikacijsku razmjenu ili oponaša server, napadač dobiva mogućnost da oponaša tog korisnika prema svim serverima sa SCRAM pristupom koji koriste istu hash funkciju, šifru, broj iteracija i „salt“. Zbog ovog razloga je važno koristiti nasumično generirane „salt“ vrijednosti.

SCRAM ne pregovara koju hash funkciju treba koristiti. Pregovaranje hash funkcije se ostavlja SASL mehanizmu zaduženom za pregovaranje. Važno je da klijent može sortirati lokalno dostupnu listu mehanizama po preferenciji kako bi mogao odabrati prikladni mehanizam za korištenje s liste mehanizama koju server reklamira. Kako će se lista sortirati nije specificirano zbog toga što je to lokalni problem, to jest odluka klijenta što mu je važnije. Kod sortiranja se trebaju uzeti u obzir objektivni i subjektivni dojmovi koje su prednosti kriptografskog mehanizma, na primjer ukoliko se može birati između SCRAMa s naprednijim SHA-256 mehanizmom, onda će se vjerojatno od odabrati umjesto običnog SCRAM s SHA1.

Treba napomenuti da kako bi se zaštitili pregovori SASL mehanizma, aplikacija uobičajeno treba napraviti listu serverskih mehanizama dva puta. Jednom prije i jednom nakon autentikacije. Kod drugog puta se koriste sigurnosni slojevi. Ukoliko postoji razlika između lista, autentikacija nije sigurna i treba prekinuti vezu.[59]

[59] ibidem., str. 20.-22.

13. Implementacija TOTP

TOTP ili Time Based One Time Password je ekstenzija OTP algoritma kod kojeg OTP algoritam generira kratkoročni OTP koji generalno traje 30 sekundi. Ovakva implementacija se koristi za pojačavanje faktora sigurnosti. Ovakva implementacija ima široku upotrebu pa se koristi za mrežne aplikacije, pristup virtualnim privatnim mrežama i Wi-Fi prijavu u transakcijske web aplikacije. U nastavku je prikazan postupak implementacije TOTP algoritma u JavaScript jeziku.

13.1. Node.js, JSOTP i Browserify

Kako bi se implementirao TOTP algoritam potrebno je koristiti određene pakete to jest node module. Modul JSOTP pruža sve potrebne funkcije za optimalno korištenje TOTP algoritma. Za implementaciju korištene su sljedeće JSOTP funkcije:

- jsotp.TOTP(secret),
- jsotp.TOTP.now(),
- jsotp.TOTP.verify(totp).

Funkcija jsotp.TOTP(secret) prima base32 enkodiran string te nakon toga generira TOTP instancu. U implementaciji koja je korištena u ovom radu korisnik može sam odrediti string koji će biti korišten. U slučaju da to ne želi, jsotp modul ima funkciju jsotp.Base32.random_gen(veličina). Ta funkcija generira nasumičan base32 enkodirani string. Ukoliko se ne specificira veličina, zadana vrijednost je 16. Funkcija jsotp.TOTP.now() vraća jednokratnu šifru s trenutnim vremenom. Upravo je to jednokratna šifra koja se koristi kod autentifikacije. Zbog toga što se koristi trenutno vrijeme ona nakon određenog vremenskog perioda će postati nevažeća te će se automatski generirati nova jednokratna šifra s vremenom u tom momentu. Funkcija jsotp.TOTP.verify(totp) kao argument prima jednokratnu šifru te provjerava da li je ona važeća.

Drugi modul koji je korišten je Browserify. On nema nikakav utjecaj na samo izvršavanje TOTP algoritma. Browserify je modul koji omogućava korištenje require() funkcije u kodu za internet preglednik. Postoje alternativna rješenja za korištenje instaliranih modula, ali Browserify je jedno od jednostavnijih i pouzdanijih rješenja zbog toga što nakon učini dostupne node module on se više ne koristi te se nastavlja normalna implementacija.

Od ostalih alata pri implementaciji TOTP korištena je jQuery biblioteka radi pojednostavljenog selektiranja i manipuliranja HTML dokumentom. Također je korišten Bootstrap framework radi boljeg vizualnog prikaza, ali on nije obavezan za ovu implementaciju.

13.2. Google Authenticator i Google API Chart

Jedan od ključnih dijelova TOTP implementacije je mobilna aplikacija Google Authenticator. Ona omogućava skeniranje QR koda ili upisivanje base32 enkodiranog stringa koji se koristi tokom implementacije. Kada se Google Authenticator poveže, on će svakih 30 sekundi generirati novu jednokratnu šifru. Ovakav način je općeprihvaćen pa se on može vidjeti kod korištenja aplikacije kao što je to Steam. Korisnik mora instalirati Steam aplikaciju na svoj mobitel kako bi mogao aktivirati svoj korisnički račun i koristiti sve mogućnosti koje Steam pruža. Pri svakom pokušaju prijave, nakon što korisnik upiše točno korisničko ime i šifru, dobije prozor gdje se od korisnika očekuje da upiše jednokratnu šifru koju generira aplikacija na njegovom mobitelu koja je povezana s njegovim korisničkim računom. Ukoliko je jednokratna šifra netočna ili je prekasno upisana, prijava neće biti moguća i autentikacija je neuspješna. Google Authenticator je slična aplikacija koja pruža generiranje jednokratnih šifri za OTP autentikaciju. Google API Chart je korišten za dinamično generiranje QR koda kako korisnik ove implementacije ne bi morao ručno upisivati kod kako bi povezao Google Authenticator.

13.3. Koraci implementacije

Prije početka kodiranja potrebno je instalirati JSOTP modul. Za to je potrebno imati Node.js. Pri izradi ove implementacije korištena je verzija 8.11.3, ovisno o tome kada se pokušava replicirati ova implementacija možda će biti potrebno prilagoditi neke naredbe ili koristiti druge module jer moduli koji su korišteni za ovu implementaciju možda neće biti kompatibilni sa novom verzijom Node.js. Prije svega je potrebno instalirati JSOTP modul. Kod korištenja Node modela potrebno je raditi s command promptom ili PowerShellom. Svi korišteni moduli se moraju nalaziti unutar mape s ostatkom koda. Nakon što se navigira do mape gdje se želi instalirati modul potrebno je unijeti sljedeću komandu u PowerShell.

```
C:\Users\rkarl\Desktop\TOTP>npm install jsotp
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\rkarl\Desktop\TOTP\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\rkarl\Desktop\TOTP\package.json'
npm WARN TOTP No description
npm WARN TOTP No repository field.
npm WARN TOTP No README data
npm WARN TOTP No license field.

+ jsotp@1.0.3
added 2 packages in 0.305s
```

Kod 4 - Instaliranje JSOTP modula

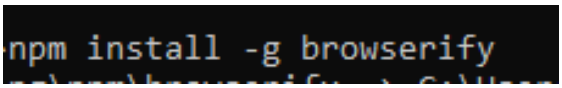
Nakon naredbe „npm install jsotp“ u mapi se nalazi mapa node_modules gdje se nalazi JSOTP modul. Sada su na raspolaganju sve prethodno navedene funkcije JSOTP modula. U nastavku je prikazan kod koji se nalazi unutar JavaScript datoteke.

```
let jsotp = require('jsotp');

$('#qrImg').attr('src',
'https://chart.googleapis.com/chart?chs=200x200&cht=qr&chl=200x200&chld=M|0&ch
t=qr&chl=otpauth://totp/user@host.com%3Fsecret%3D' + $('#secret').val());
let secret = $('#secret').val();
let totp = jsotp.TOTP(secret);
let passcode = totp.now();
$('#otp').text(passcode);
timeout();

function timeout() {
  setInterval(function () {
      if(totp.verify(passcode) == false){
        console.log(passcode);
        passcode = totp.now();
        console.log(passcode);
        $('#otp').text(passcode);
      }

  }, 100);
};

$(function () {

  $('#secret').keyup(function () {
    $('#qrImg').attr('src',
'https://chart.googleapis.com/chart?chs=200x200&cht=qr&chl=200x200&chld=M|0&ch
t=qr&chl=otpauth://totp/user@host.com%3Fsecret%3D' + $('#secret').val());
    let secret = $('#secret').val();
    let totp = jsotp.TOTP(secret);
    let test = totp.now();
    $('#otp').text(test);
  });

});
```

Kod 5 - OTP JavaScript implementacija

Prije nego što počne bilo kakvo kodiranje, potrebno je uključiti JSOTP modul s require funkcijom. Nakon toga dolazi generiranje QR koda koji selektira HTML element koji ima ID qrImg gdje u src atribut dodaje Google API Chart link kojem se na kraju linka dodaje vrijednost base32 enkodiranog stringa kako bi se generirao točan QR kod. Nakon toga je napravljena varijabla koja u sebe sprema vrijedno polja u kojem se nalazi base32 enkodirani string. Varijabla totp u sebe prima totp instancu koja je dobivena pomoću jsotp.TOTP funkcije. Varijabla passcode poprima vrijednost jednokratne šifre. Koju se pomoću jQuery funkcije ispisuje. Nakon toga dolazi timeout funkcija. Njezin zadatak je da promijeni jednokratnu šifru kada ona više nije valjana. U tu svrhu je korištena funkcija SetInterval koja omogućava izvršavanje koda u vremenskim intervalima koje se definira na kraju u milisekundama. Funkcija timeout provjerava svakih 100 milisekundi da li je trenutna jednokratna šifra ispravna pomoću if uvjeta. Kada jednokratna šifra više nije ispravna, varijabla passcode dobiva novu vrijednost jednokratne šifre koja se zatim pomoću jQuery naredbe mijenja na stranici. Dvije naredbe console.log nemaju utjecaj na implementaciju, već su dodane kako bi se u konzoli mogla usporediti sadašnja jednokratna šifra sa nevažećom šifrom. Na kraju se nalazi skraćeni oblik jQuery funkcije $(document).ready(). Ona se u ovom slučaju koristi da se osigura da taj dio koda neće biti pokrenut prije nego što se taj dio stranice ne učita. U njoj se se nalazi funkcija keyup koja se izvršava pri svakom mijenjanju polja u kojem se nalazi base64 enkodirani string. To je učinjeno kako ne bi bilo potrebno osvježavati stranicu kako bi se vidjele promjene u QR kodu i jednokratnoj šifri.

Nakon što je napisan JavaScript kod, sada ga je potrebno prilagoditi kako bi ga mogao koristiti internet preglednik. Ovo ne bilo potrebno raditi ukoliko se ne koriste node moduli. Potrebno je instalirati Browserify s slijedećom naredbom:



Kod 6 - Instaliranje Browserify modula

Nakon toga je potrebno selektirati JavaScript datoteku u kojoj je napisan prethodni kod kako bi ga prilagodili s naredbom:

```
browserify javascript.js -o otp.js
```

Kod 7- Korištenje Browserify

Osim što je potrebno selektirati datoteku s kodom, potrebno je napisati ime datoteke u koju se želi spremiti generirani kod. To treba biti JavaScript datoteka. Novu datoteku je potrebno dodati u HTML datoteku kako bi funkcioniralo. Rezultat je moguće vidjeti na sljedećoj slici.

Slika 3 - QR kod i OTP šifra

Prvo polje predstavlja base32 enkodirani string iz kojeg je generiran QR kod. OTP šifra, to jest jednokratna šifra u tom momentu za gore navedeni string je bio 362 841.

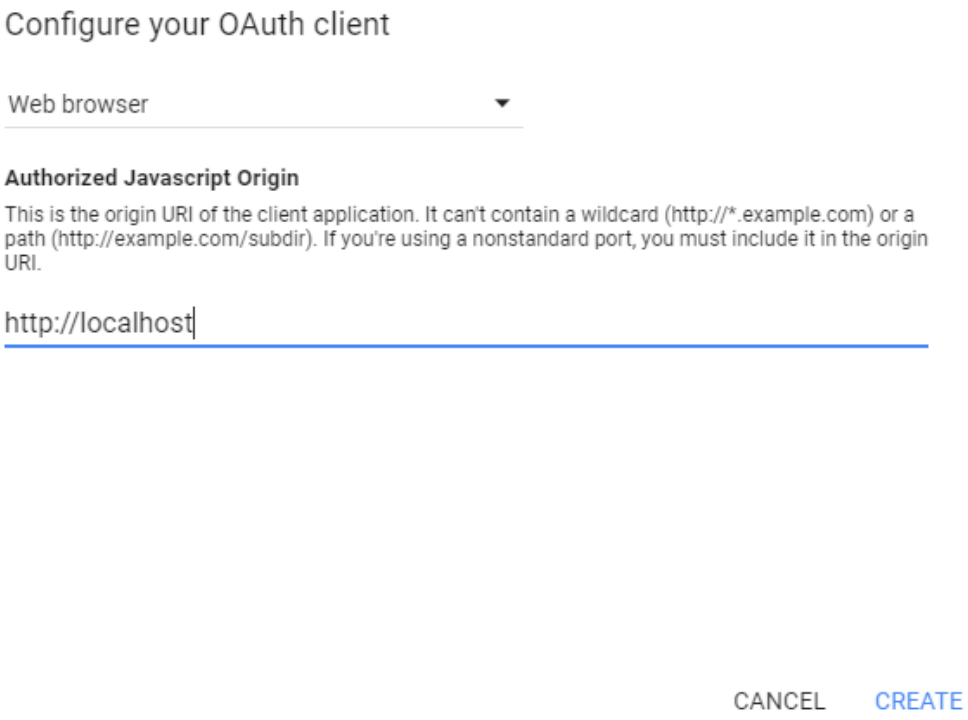
Slika 4 - Google Authenticator generirani OTP

Na svojoj mobilnoj aplikaciji korisnik vidi istu jednokratnu šifru koju je potrebno unijeti kod OTP autentikacije.

14. OAuth implementacija

Postoji više načina na koje se može implementirati autentikacija pomoću drugih korisničkih računa kao što su to Gmail, Twitter, GitHub, Facebook i drugi. U nastavku će se objasniti implementacija OAuth autentikacije uz pomoć Gmail računa. Ukoliko se želi napraviti OAuth autentikacija s mogućnosti odabira između više mogućnosti, onda je u tom slučaju najbolje koristiti Firebase alate.

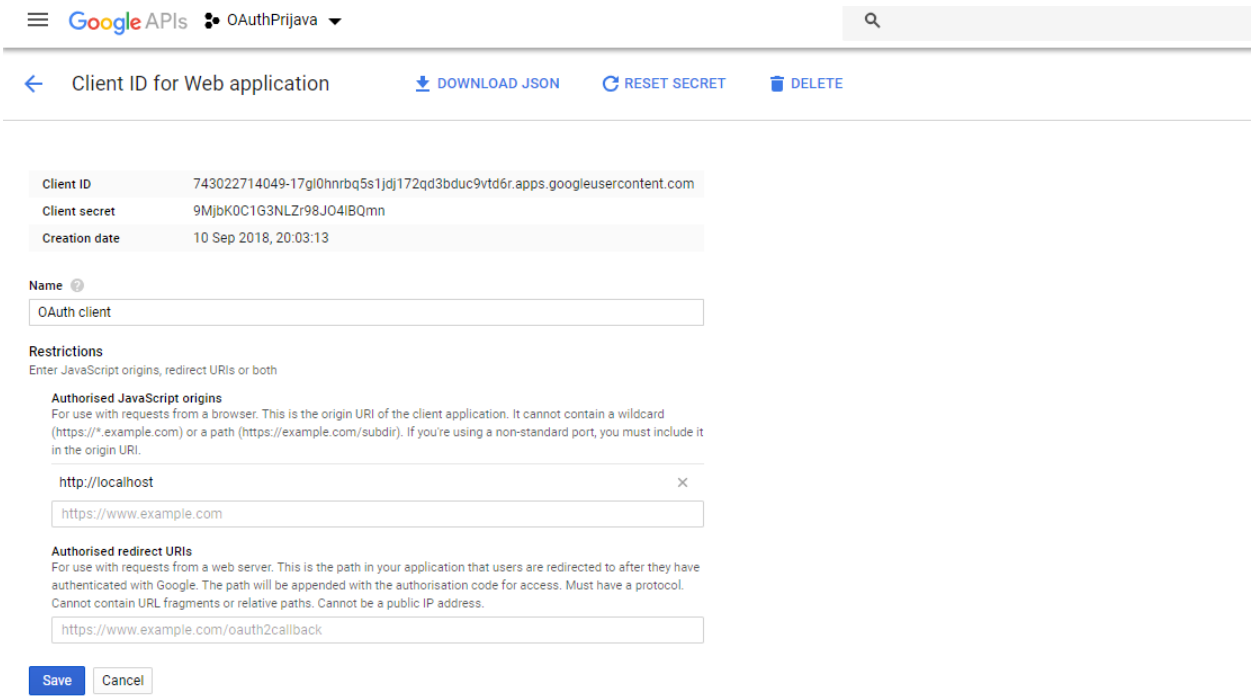
Prvo što je potrebno napraviti je izraditi novi projekt u Google Cloud Platform Console kako bi se dobio Google API ključ bez kojeg nije moguće napraviti implementaciju. Pri izradi novog projekta bitno je odabrati Web browser platformu i u polje Authorized Javascript Origin treba unijeti http://localhost kao što je prikazano na sljedećoj slici.



Slika 5 - Podešavanje Google API

Localhost znači da se web stranica ne nalazi na pravom serveru, već lokalnom serveru koji je napravljen za potrebe implementacije. Za izradu lokalnog servera, u ovoj implementaciji korištena je XAMPP aplikacija.

Nakon što je projekt uspješno napravljen potrebno je otići u Credentials i uzeti ID klijenta koji je potreban za implementaciju, što je prikazano na sljedećoj slici:



Slika 6 - ID klijenta

Sada se može započeti implementacija. U header dijelu HTML datoteke potrebno je dodati sljedeći kod:

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.m
in.css">
<meta name="google-signin-client_id" content="743022714049-
17gl0hnrbq5s1jdj172qd3bduc9vtd6r.apps.googleusercontent.com">
<script src="https://apis.google.com/js/platform.js" async
defer></script>
```

Kod 8 - Povezivanje Google API s projektom

Prvi dio je namijenjen za Bootstrap framework. Drugi dio je najbitniji gdje se u atribut content dodaje ID klijenta. Treći dio je obavezni dio koji Google API zahtijeva. Kako bi se dobio karakteristični Google dugme treba dodati sljedeći kod u HTML datoteku:

```
<div class="g-signin2" data-onsuccess="onSignIn"></div>
<div class="data">
    <p>Detalji profila</p>
    <img src="" alt="" id="pic" class="img-circle" width="100px"
height="100px">
    <p>Email</p>
    <p><span id="email" class="alert alert-danger"></span></p>
    <button onclick="signOut()" type="button" class="btn btn-
danger">Odjava</button>
</div>
```

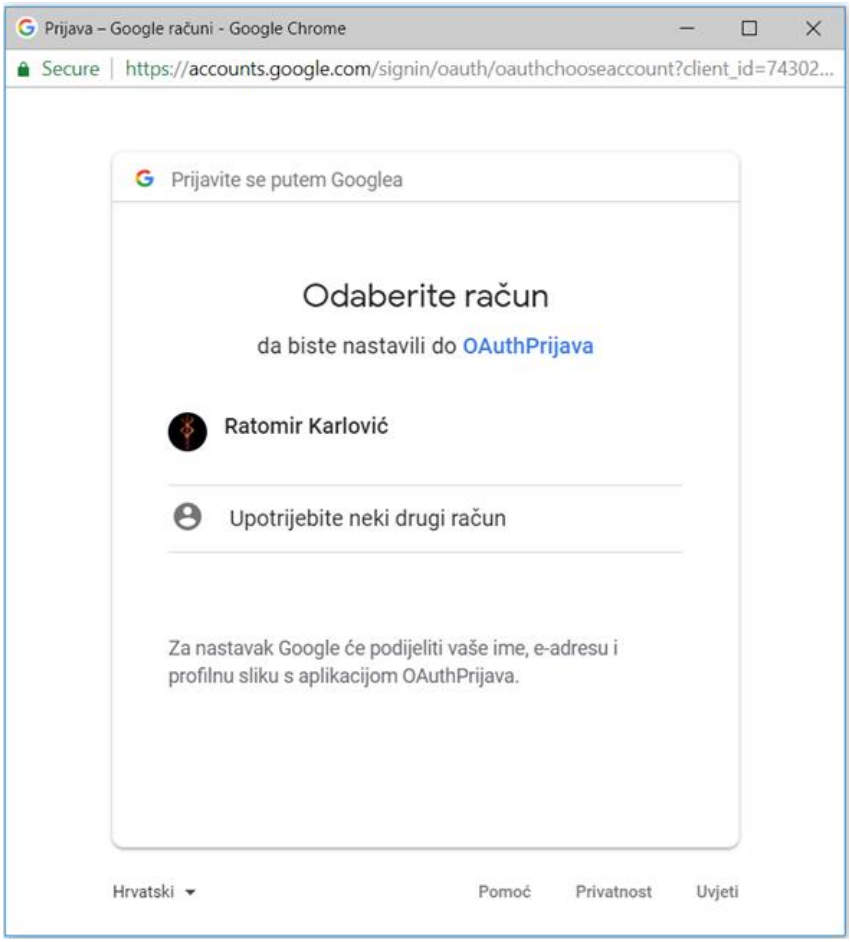
Kod 9 - Prijava i odjava unutar HTML

Dugme koje će biti zaduženo za prijavu potrebno je atribut data-onsuccess="onSignIn" kao što je definirano u Google API dokumentaciji. Ostatak koda se pokazuje nakon uspješne prijave korisnika gdje će pokazati sliku korisnika i njegovu elektroničku adresu. U sljedećem kodu su prikazane funkcionalnosti napisane u JavaScript jeziku.

```
function onSignIn(googleUser){
    var profile = googleUser.getBasicProfile();
    $(".g-signin2").css("display","none");
    $(".data").css("display","block");
    $("#pic").attr("src", profile.getImageUrl());
    $("#email").text(profile.getEmail());
}

function signOut() {
    var auth2 = gapi.auth2.getAuthInstance();
    auth2.signOut().then(function () {
      alert("Uspješno ste se odjavili!")
      $(".g-signin2").css("display","block");
      $(".data").css("display","none");
    });
  }
```

Kod 10 - JavaScript funkcije za prijavu i odjavu

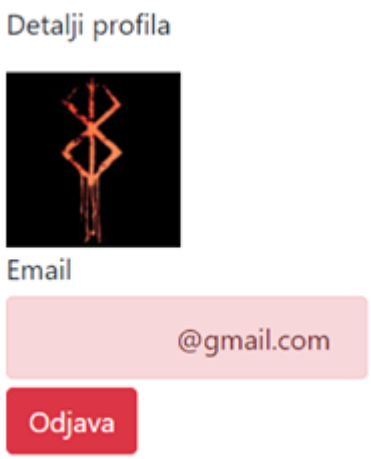
Funkcija onSignIn dobiva atribut googleUser čije se informacije zatim spremaju u varijablu profile pomoću googleUser.getBasicProfile() naredbe koja dolazi s Google APIem. Nakon toga dolazi CSS s kojim se sakriva dugme za prijavu i pokazuje slika i elektronička adresa korisnika koji se autenticirao. Pomoću funkcija getImageUrl() i getEmail() dobivaju se podaci za sliku profila i elektroničke adrese. Funkcija signOout koristi Google API naredbu var auth2 = gapi.auth2.getAuthInstance() kako bi se napravila odjava koja odjavljuje korisnika samo s trenutne stranice, ali korisnik će ostati prijavljen u ostalim Google servisima. Kada korisnik pritisne na gumb za prijavu, dobit će novi prozor gdje može odabrati s kojim korisničkim računom se želi prijaviti ukoliko ih ima više. U primjeru na slici postoji samo jedan zapamćeni korisnički račun.



Slika 7 - Prijava korisnika

Nakon odabira željenog korisničkog računa, korisnik će možda morati upisati šifru ovisno o tome na koji način je spremljena prijava.

Ukoliko je prijava uspješna, prozor se automatski zatvara te se na stranici mogu vidjeti korisnički podaci, u ovom slučaju njegova slika i elektronička adresa kao što je to prikazano na slijedećoj slici.



Slika 8 - Rezultat prijave pomoću OAuth metode

Nakon što korisnik pritisne na dugme za odjavu, korisnik se odjavljuje s trenutne stranice, ali sve ostale prijave preko Google računa su i dalje valjane.

15. SCRAM-SHA1 implementacija

Za implementaciju SASL SCRAM-SHA1 mehanizma korišten je Node.js koji je omogućio korištenje modula crypto, Crypto-JS i net. Modul net se koristi za TCP komunikaciju servera i klijenta. Modul crypto se koristi za dobivanje nasumičnog niza znakova koji se koristi kao salt. Crypto-JS omogućava korištenje SHA256 algoritma s kojim će se zaštiti podaci.

U nastavku je prikazan kod koji se koristi za server.

Kod 11 - Server

```
const net = require('net');
const HmacSHA256 = require('crypto-js/hmac-sha256');
const crypto = require('crypto');



var genRandomString = function(length){
    return crypto.randomBytes(Math.ceil(length/2))
            .toString('hex')
            .slice(0,length);
};

const nonce = genRandomString(8);
const password = '12345678';
const serverPassword = 'server123';

function salting(password, nonce, iteration) {
    let Salting = password;
    for (let index = 0; index < iteration; index++) {
        Salting = HmacSHA256(Salting, nonce);
    };
    return (Salting.toString());
}
```

```
const server = net.createServer()

server.on("connection", (socket) => {

    const dataTimeOut = (msg) => setTimeout(() => {
        socket.write(JSON.stringify(msg));
    }, 2000);

    socket.on("error", (err) =>{
    console.log("Policy server socket error: ");
    console.log(err.stack);
    });

    socket.setEncoding("utf8");
    socket.on("data", (data) => {
        const msg = JSON.parse(data);
        const GSHeader = Buffer.from(msg.GS, 'base64').toString();
        if (GSHeader != null && msg.order == 1) {
            let noncePure = Buffer.from(msg.r, 'base64').toString();
            let nonceMerge = noncePure.concat(nonce);
            let nonceEncoded = Buffer.from(nonceMerge).toString('base64');
            msg.r = nonceEncoded;
            let iteration = (Math.floor(Math.random() * 20) + 1);
            msg.iteration = iteration;
            msg.order = 2;
            dataTimeOut(msg);

        }
        if(msg.order == 3){
            let ClientProof = Buffer.from(msg.ClientProof, 'base64').toString();
            let proof = ClientProof;
            let iteration = msg.iteration;
            let commonNonce = Buffer.from(msg.r, 'base64').toString();
            let testProof = salting(password, commonNonce, iteration);

            console.log('Server SHA256 password:  ' + testProof);
            console.log('Klijent SHA256 password: ' + proof);
```

```
            setTimeout(() => {
            if(testProof == proof){
                msg.status = 'success';
                let commonNonce = Buffer.from(msg.r, 'base64').toString();
                let iteration = msg.iteration;
                let serverProofPure = salting(serverPassword, commonNonce, iteration);
                let serverProof = Buffer.from(serverProofPure).toString('base64');
                msg.ServerProof = serverProof;
                console.log('Autentikacija klijenta uspješna!!!')
                setTimeout(() => {
                console.log('Server Proof Encoded: '+ serverProof);
                console.log('Server Proof: '+ serverProofPure);
            }, 1500);
            }else{
                msg.status = 'fail';
                console.log('Autentikacija klijenta neuspješna!!!')
            }
            msg.order = 4;
            dataTimeOut(msg);
            }, 3500);
        }


    })


})

server.listen(1337, '127.0.0.1');
```

Za generiranje nasumičnog niza podataka koristi se funkcija getRandomString() koja prima željenu dužinu niza. Ona se koristi za generiranje nonce varijable. Nonce označava broj koji se koristi samo jednom. On se koristi kod generiranja Salta koji će se koristiti kod generiranja Salted passworda za server i klijent. Za generiranje broja iteracija koristi se funkcija (Math.floor(Math.random() * 20) + 1). S njom se dobiva nasumičan broj od 1-20. Maksimalni broj se može povećati, ali treba napomenuti kako s većim brojem iteracija, autentikacija je sigurnija, ali i sporija. Za ovaj primjer je korišten mali broj iteracija jer je

svrha demonstrirati kako mehanizam funkcionira. U praktičnoj primjeni, broj iteracija bi trebao biti dovoljno velik da se postigne što bolja zaštita, ali dovoljno malen da autentikacija ostane brza kako ne bi utjecao na iskustvo korisnika. Funkcija dataTimeOut se ne koristi u praksi, ali u ovom primjeru njezina svrha je da uspori ispisivanje rezultata zbog lakšeg čitanja. Za stvaranje TCP servera koristi se net.createServer() nakon kojeg dolaze predefinirane funkcije kao što je to server.on("connection", (socket) => { }) unutar kojeg se pišu serverske naredbe koje se žele izvršiti. Nakon toga dolazi funkcija koja ukoliko kod radi kako treba neće nikad biti korištena jer njezina svrha da ispiše grešku koja se dogodila tokom pokretanja koda. Funkcija socket.setEncoding("utf8") se koristi za postavljanje enkodiranja server jer inače bi se svaka varijable trebala posebno pretvarati. Zatim dolazi najvažniji dio koda, a to je socket.on("data", (data) =>{ }) koji čeka da server primi podatke nakon čega izvršava funkcije koje se nalaze unutar njega. On se izvršava svaki put kada server prime podatke pa je zbog toga potrebno definirati uvjete kada se određeni dio koda izvršavati, u suprotnom svaki put kad server dobije podatke on će izvršiti sav kod unutar ove funkcije što će najvjerojatnije završiti s greškom. Prije nego što se postavljeni uvjeti za izvršavanje koda, postavljena je varijabla msg u koju se spremaju dobiveni podaci, prije toga je potrebno iskoristiti JSON.parse() kako bi se kasnije moglo pristupati podacima. Ovo je potrebno napraviti svaki put kada server primi podatke pa je to razlog zašto se nalazi izvan uvjeta. Određene varijable kao što su nonce, ClientProof i slične, moraju biti enkodirane uz pomoć base64. Za tu svrhu se koristi funkcija Buffer.from(varijabla).toString('base64') kako bi se podaci enkodirali, a za dekodiranje Buffer.from(base64EnkodiranaVarijabla, 'base64').toString(). Prvi uvjet kod servera se izvršava kada klijent pošalje podatke jer je SCRAM mehanizam kod kojeg autentikacija započinje kada klijent prvi pošalje poruku serveru. GSHeader mora biti uključen pa se on provjerava kod prvog uvjeta, te msg.order koji označava redoslijed slanja poruka između klijenta i servera. On je ključan u prepoznavanju kada se određeni dijelovi koda trebaju izvršiti. Kod prve razmjene, server dodaje svoj nonce na klijentov uz pomoć funkcije concat() i definira broj iteracija koji će se koristiti pri ovoj autentikaciji. Server mijenja msg.order kako bi klijent znao na kojem se koraku autentikacije nalazi. Idući put nakon što server primi podatke od klijenta, on uspoređuje klijentski dokaz (ClientProof) sa svojim dokazom kako bi odredio da li se klijent uspješno autenticirao. Ukoliko je autentikacija

uspješna, server šalje poruku klijentu da je autentikacija uspješna, ali mora poslati serverski dokaz kako bi se server autenticirao kod klijenta. Dokaz se dobiva uz pomoć funkcije salting(). Ona prima šifru klijenta ili servera, Salt to jest zajednički nonce klijenta i servera te broj iteracija kako bi funkcija znala koliko puta treba iskoristiti SHA256 algoritam. Nakon što klijent pošalje svoju zadnju poruku smatra se da je autentikacija klijenta uspješna.

Kod 12 pokazuje kod koji je korišten za demonstraciju klijentske strane u SCRAM-SHA256 mehanizmu.

Kod 12 - Klijent

```
const net = require('net');
const HmacSHA256 = require('crypto-js/hmac-sha256');
const crypto = require('crypto');


var genRandomString = function(length){
    return crypto.randomBytes(Math.ceil(length/2))
            .toString('hex')
            .slice(0,length);
};

var salting = function(password, nonce, iteration) {
    let Salting = password;
    for (let index = 0; index < iteration; index++) {
        Salting = HmacSHA256(Salting, nonce);
    };

    return (Salting.toString());
}
```

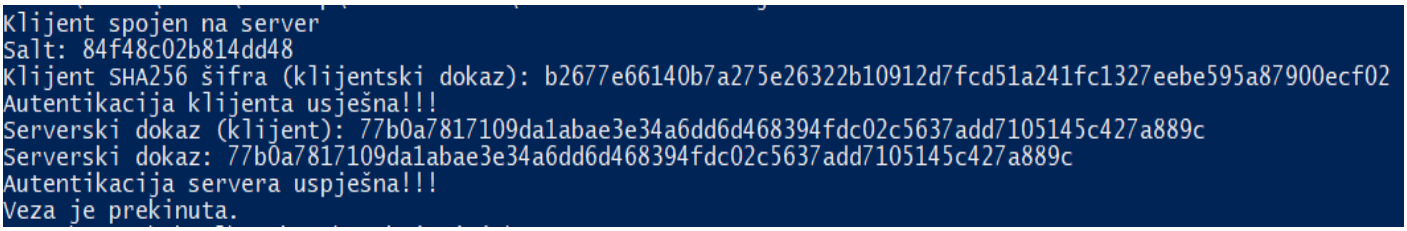
```
const noncePure = genRandomString(8);
const nonce = Buffer.from(noncePure).toString('base64');
const clientPassword = '12345678';
const serverPassword = 'server123';
const GSPure = 'n';
let GSEncoded = Buffer.from(GSPure).toString('base64');


let msg = {
    GS: GSEncoded,
    order: 1,
    n: 'user',
    r: nonce,
    iteration: null,
    ClientProof: null,
    ServerProof: null,
    status: null
}

var client = new net.Socket();
client.connect(1337, '127.0.0.1', function() {
    console.log('Klijent spojen na server');
    client.write(JSON.stringify(msg));
});

client.on('data', function(data) {
    let msg = JSON.parse(data);
    if(msg.order == 2){
        let iteration = msg.iteration;
        let commonNonceEncoded = msg.r;
        let commonNonce = Buffer.from(commonNonceEncoded, 'base64').toString();
        console.log('Salt: ' + commonNonce);
        let clientProofPure = salting(clientPassword, commonNonce, iteration);
        console.log('Klijent SHA256 password: ' + clientProofPure);
        let clientProof = Buffer.from(clientProofPure).toString('base64');
        msg.ClientProof = clientProof;
        msg.order = 3;
        client.write(JSON.stringify(msg));
    }
```

```
if (msg.order == 4) {
        if(msg.status == 'success'){
            console.log('Autentikacija klijenta usješna!!!');
            let serverProofEncoded = msg.ServerProof;
            let serverProof = Buffer.from(serverProofEncoded, 'base64').toString();
            console.log('Server Proof: '+ serverProof);
            let commonNonce = Buffer.from(msg.r, 'base64').toString();
            let iteration = msg.iteration;
            let serverProofClient = salting(serverPassword, commonNonce, iteration);
            if (serverProof == serverProofClient){
                console.log('Autentikacija servera uspješna!!!');
            }else{
                console.log('Autentikacija servera neuspješna!!!');
            }
        }else{
            console.log('Autentikacija neusješna!!!');
        }

        client.destroy();
    }

});

client.on('close', function() {
    console.log('Connection closed');
});
```

Kako bi se napravio klijent potrebno je napraviti novi socket uz pomoć naredbe new net.Socket(); i pridružiti je određenoj varijabli pomoću koje će se pozivati naredbe. Nakon toga je potrebno spojiti klijenta na server. To se radi uz pomoć client.connect naredbe kojoj se daje port i adresa servera. Pri uspješnom spajanju klijenta na server ispisuje se poruka koja to povrđuje i šalje se prva poruka serveru koja je prije toga obrađena funkcijom JSON.stringify(). Nakon toga kao što je to slučaj kod servera, potrebno je imati funkciju koja se izvršava kod primanja podataka sa servera. Kada server pošalje svoj prvi odgovor sa Saltom koji će se koristiti, klijent koristi funkciju salting() kako bi dobio klijentski dokaz koji šalje serveru kako bi se autenticirao. Kada server pošalje svoj zadnji

odgovor, klijent provjerava status. Ukoliko je status jednak „success“ to znači da je autentikacija uspješna, nakon čega klijent izračunava serverski dokaz i uspoređuje ga s dokazom kojeg je server poslao u svojoj posljednjoj poruci. Ukoliko su dva dokaza ista, autentikacija servera je uspješna i ispisuje se poruka koja to potvrđuje, u suprotnom autentikacija servera nije uspješna i ispisuje se prikladna poruka. Nakon toga se izvršava naredba client.destroy() koja prekida vezu sa serverom. Na sljedećoj slici je prikazano kako to izgleda unutar PowerShell konzole.

Slika 9 - Prikaz ispisa konzole za klijenta

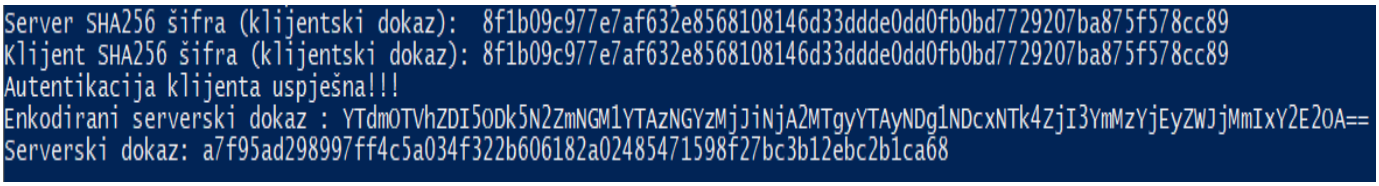


Kada se klijent spoji na server ispisuje se poruka „Klijent spojen na server“. Nakon toga se odvijaju slijedeće radnje:

1. Prikazuje se koji se Salt koristi pri generiranju klijentskog dokaza to jest šifre
2. Ispisuje se klijentski dokaz koji se šalje serveru
3. Klijent prima zadnju poruku od servera
4. Ispisuje se rezultat autentikacije
5. Ukoliko je je autentikacija uspješna, ispisuje se serverski dokaz koji je izračunao klijenti i dokaz koji je poslao server
6. Dva dokaza se uspoređuju te se ispisuje rezultat usporedbe
7. Prekida se veza servera i klijenta

Prekid veze servera i klijenta označava kraj SCRAM-SHA256 autentikacije. Na slici 10 je prikazana serverska strana autentikacije.

Slika 10 - Prikaz ispisa konzole za server



Nakon što klijent pošalje svoj dokaz, server izračunava klijentski dokaz. Oba dokaza se ispisuju te se nakon toga izvršava usporedba. Ukoliko je ona uspješna, server šalje enkodirani serverski dokaz klijentu. Server ispisuje serverski dokaz i enkodirani serverski dokaz prije nego što pošalje enkodirani dokaz klijentu.

15. Zaključak

Iz svega navedenog može se zaključiti kako ne postoji najbolja kriptografska metoda autentikacije i autorizacije. Ovisno o potrebama sustava i mogućim napadima na njega, treba odabrati odgovarajuću metodu koja će zadovoljiti sve kriterije. Međutim, iako se odabere najbolja metoda ona sama neće biti dovoljna, nego će uz korištenje metode trebati koristiti različite protokole, kao što je to TLS, kako bi se osigurali svi aspekti sigurnosnog sustava, uspješno obranili od napada i podaci postalo neupotrebljivi čak i ako je napad bio uspješan. Zbog toga kriptografske metode autentikacije i autorizacije moraju držati korak s razvojem tehnologije inače će pružati premalu zaštitu svojim korisnicima. Međutim to nije jedini kriterij razvoja ovakvih metoda. Osim što trebaju pružati visoku razinu zaštite koju je lako implementirati u već postojeće sigurnosne sustave bez potrebe za velikom promjenom sigurnosnog sustava. Metode bi trebale biti modularne isto kao i sigurnosni sustavi u kojima će se koristiti. Korisnik bi uvijek trebao imati izbor kod budućih metoda za autentikaciju i autorizaciju između manje sigurnih metoda koje su lakše za implementaciju i metoda koje pružaju veću sigurnost, ali zbog toga će njihov proces implementacije biti duži, ali ne nužno i kompliciraniji. Buduće metode bi trebale nastojati ostvariti što veću sigurnost korisnika kroz jednostavnije i sigurnije implementiranje autentikacije. Od takvih metoda se također očekuje da imaju mogućnost rada s ostalim metodama kao što je to slučaj s OAuth i OTP metodom. Primjer takve primjene je aplikacija Steam. Ona koristi OAuth kao autorizacijsku metodu, dok je glavna uloga OTP autentikacija korisnika. Ovakva upotreba metoda se ne treba primjenjivati u svim slučajevima jer kao što je objašnjeno u prethodnim poglavljima ovog rada, postoji mnogo različitih metoda koje imaju svoje prednosti i mane. Ovisno o tome što je potrebno za određeni projekt, prikladna metoda treba biti odabrana. Ukoliko se koriste osjetljivi podaci kao što su to brojevi kreditnih kartica i slične, razina sigurnosti koju sustav treba pružiti raste. Trenutni mehanizmi pružaju veliki izbor između razina sigurnosti, ali njihova je mana što i dalje većina metoda nema zaštitu integriteta podataka i povjerljivost podataka zbog čega je potrebno koristiti druge protokole. Budući mehanizmi bi trebali riješiti taj problem kako bi se njihova sigurnost povećala i postali dostupniji svima.

Literatura

Dokumentacija

1. Melnikov A. i K. Zeilenga, *Simple Authentication and Security Layer SASL)*, 2006., Dostupno na: RFC-Editor, (pristupljeno 21. kolovoza 2018.).
2. [Zeilenga K., The PLAIN Simple Authentication and Security Layer (SASL) Mechanism, 2006., Dostupno na: RFC-Editor, (pristupljeno 22. kolovoza 2018.).
3. Klensin J., Catoe R. i P. Krumviede, IMAP/POP AUTHorize Extension for Simple Challenge/Response, 1997., Dostupno na: RFC-Editor, (pristupljeno 22. kolovoza 2018.).
4. Mills W., Showalter T. i H. Tschofenig, A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth, 2015., Dostupno na: RFC-Editor, (pristupljeno 24. kolovoza 2018.).
5. Nystrom M., The SecurID(r) SASL Mechanism, 2000., str. Dostupno na: RFC-Editor, (pristupljeno 24. kolovoza 2018.).
6. Newman C., 1998., The One-Time-Password SASL Mechanism, Dostupno na: RFC-Editor, (pristupljeno 25. kolovoza 2018.).
7. Newman C. et al., Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms, 2010., Dostupno na: RFC-Editor, (pristupljeno 27. kolovoza 2018.).

Online članci

1. OAuth: https://oauth.net/, (pristupljeno 27. kolovoza 2018.).
2. Twillio: https://www.twilio.com/docs/authy/api/one-time-passwords, (pristupljeno 27. kolovoza 2018.).
3. Firebase Authentication: https://firebase.google.com/docs/auth/, (pristupljeno 27. kolovoza 2018.).
4. Developers Google: https://developers.google.com/identity/sign-in/web/sign-in, (pristupljeno 27. kolovoza 2018.).
5. JQuery Documentation: https://api.jquery.com/, (pristupljeno 27. kolovoza 2018.).

Popis slika

Popis kodova

Sažetak

Uz sve prednosti koje dolaze s globalnim povezivanjem, sigurnost informacija je postala problem. Zbog toga postoje različite kriptografske metode koje pružaju korisnicima određenu razinu sigurnosti bez da utječu na njihovo korisničko iskustvo. SASL mehanizmi su jedni od tih sigurnosnih mjera koje štite korisnike i njihove informacije. Postoje različiti mehanizmi s različitim razinama sigurnosti koje pružaju. Jedni od najpoznatijih mehanizama su OTP i OAuth mehanizam koji su široko rasprostranjeni i pružaju najbolju sigurnost svojim korisnicima. Budući mehanizmi trebali bi svakako obratiti pažnju na propuste koje sadašnji mehanizmi imaju, kao što je loša zaštićenost integriteta podataka i povjerljivosti podataka. Također, budući mehanizmi će nastojati implementirati složenije metode autentikacije, kao što je to biometrijska autentikacija. S razvojem tehnologija, sigurnost će biti izložena novim napadima, ali to je također prilika za razvoj boljih i bržih metoda za autentikaciju i autorizaciju.

Ključne riječi: SASL, SASL mehanizmi, OTP, OAuth, autentikacija, autorizacija, kriptografija

Summary

With all the good aspects that come from global connection, there is an issue of information security. That is the reason why there are many different cryptographic methods that provide users with a certain level of security without affecting their user experience. SASL mechanisms are one of those security measures that protect users and their information. There are a lot of different mechanisms with different security levels. The most popular mechanisms are OTP and OAuth mechanisms that are used wide and provide their users with the best security. Future mechanisms should pay attention to flaws of current mechanism that have bad security for data integrity and data confidentiality. Also, future mechanisms should try to implement more complex authentication methods, like biometric authentication. With growth of technology, security will get exposed to new attacks but that is also an opportunity to develop better and faster methods for authentication and authorization.

Keywords: SASL, SASL mechanisms, OTP, OAuth, authentication, authorization, cryptography