HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

Opis GSS standarda

CCERT-PUBDOC-2004-02-61

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.
Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, **www.lss.hr**- laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Sve više novih aplikacija su tzv. distribuirane aplikacije koje se ne izvode na jednom računalu već koriste nekoliko računala spojenih u mrežu i na svakom od njih se izvodi samo jedan dio aplikacije.
Dijelovi aplikacije koji se nalaze na različitim računalima moraju stalno izmjenjivati podatke, a kako je mrežna komunikacija osjetljiva na napade neovlaštenih korisnika, svaka distribuirana aplikacija je ujedno potencijalna sigurnosna rupa u sustavu. Zbog toga je u sve takve aplikacije potrebno ugraditi posebnu zaštitu svih podataka koji se šalju preko mreže.
Zaštita mrežne komunikacije podrazumijeva autentikaciju korisnika te zaštitu podataka od neovlaštenog pristupa ili mijenjanja. Postoje mnogi protokoli i softverske realizacije za implementaciju takve zaštite, ali su često dosta kompliciranih za implementaciju, a gotovi kod je teško prenosiv između različitih računalnih platformi.
GSS (engl. *Generic Security Standard*) je standard koji definira zaštitu komunikacije između aplikacija na mreži. GSS je sloj između mrežne aplikacije i sigurnosnog mehanizma za zaštitu podataka. Zadaća GSS-a je da omogući određenu apstrakciju u programiranju. Njegovom primjenom programer može ugraditi zaštitu podataka u kod bez znanja o konkretnom sigurnosnom mehanizmu koji će se primjenjivati za zaštitu podataka. GSS se ne brine o načinu prijenosa podataka između aplikacija niti o njihovoj interpretaciji, a isto tako se ne brine niti o fizičkom mehanizmu zaštite podataka. Prijenos podataka je zadaća same aplikacije, a fizička zaštita podataka tj. protokoli za enkripciju i potpisivanje podataka, te parametri za autentikaciju su zadaća sigurnosnog mehanizma koji se primjenjuje.
GSS ima posebnu biblioteku funkcija napisanu u programskom jeziku C koju je moguće upotrebljavati u korisničkim aplikacijama. Biblioteka funkcija koja implementira GSS se obično naziva GSS-API (engl. *GSS Application Programming Interface*).
GSS trenutno ima podršku samo za Kerberos v5 sigurnosni mehanizam no u budućnosti se planira podrška za razne mehanizme, poput RSA sustava. Uskoro bi, uz biblioteku u jeziku C, trebala biti završena i biblioteka za programski jezik Javu.

2. Kerberos v5

Kerberos je sigurnosni mehanizam razvijen na MIT-u, 80-ih godina. Trenutno najnovija verzija programa je 5 i ta verzija se danas smatra Kerberos standardom. Kako je rad GSS-a zasnovan na Kerberos-u, u ovom poglavlju će biti ukratko opisan princip rada tog sustava.
Kerberos omogućuje autentikaciju korisnika te enkripciju i digitalno potpisivanje komunikacije između korisnika i nekog mrežnog servisa.
Najveća prednost Kerberos sustava je što omogućuje korisniku pristup svim mrežnim servisima pomoću jedinstvene zaporke koju je dovoljno upisati samo jednom.
Kerberos je mehanizam na kojem se zasniva rad Microsoft Active Directory-a. Za rad u Windows domeni dovoljno je prijaviti se za rad na desktop računalu kako bi se dobio slobodan pristup svim dijeljenim resursima u lokalnoj Windows računalnoj mreži.
Kerberos ima sustav autentikacije korisnika koji se zasniva na posebnom autentikacijskom poslužitelju (na Windows domeni to je uloga domenskog poslužitelja). Razmjena podataka između svakog računala i autentikacijskog poslužitelja je enkriptirana pomoću ključa izvedenog iz korisničke zaporke. Takav ključ se zove poslužiteljski ključ i poznat je samo dotičnom računalu i autentikacijskom poslužitelju. Svi podaci u Kerberos sustavu su enkriptirani pomoću DES simetričnog enkripcijskog algoritma.
Za razmjenu podataka između dva računala jedno računalo uvijek inicira komunikaciju i u nastavku dokumenta će se zvati klijent, a drugo odgovara na taj zahtjev i zvati će se poslužitelj. Podaci se između dva računala šalju šifrirani pomoću posebnog ključa sjednice. Ključ sjednice je slučajna vrijednost koju generira autentikacijski poslužitelj i šalje je svakom računalu enkriptiranu pomoću njegovog poslužiteljskog ključa. Za svaku komunikaciju se generira novi ključ sjednice, a postoji i vremensko ograničenje nakon kojeg taj ključ više ne vrijedi.
Prije komunikacije klijent mora poslužitelju poslati svoju identifikacijsku oznaku. Oznaka se sastoji od dva dijela.
Prvi dio je ključ sjednice enkriptiran pomoću poslužiteljskog ključa drugog računala. Klijent dobije tako enkriptiran ključ sjednice od autentikacijskog poslužitelja, a kako ne zna poslužiteljski ključ drugog računala ne može ga ni pročitati ni mijenjati već samo poslati drugom računalu.

Drugi dio identifikacijske oznake je naziv klijenta enkriptiran pomoću ključa sjednice te *hash* sažetak tog naziva. Poslužitelj može dekriptirati ključ sjednice koji mu je poslao klijent i tim ključem dekriptirati i naziv klijenta. Ako je poruka zaista došla od klijenta onda će dekriptirani naziv odgovarati primljenom *hash* sažetku. Radi bolje sigurnosti u identifikacijsku je oznaku uključeno i vrijeme kada je poruka poslana, tako da poslužitelj može provjeriti valjanost primljenog ključa sjednice.
Kerberos sustav je najranjiviji na napad primjenom sile, što pogotovo dolazi do izražaja ako korisnici koriste kratke i jednostavne zaporke. Kerberos sustav ne pruža zaštitu za svu komunikaciju između dva računala, već samo za poruke onih programa koji su posebno modificirani da koriste Kerberos.

3. Opis GSS standarda

GSS biblioteka funkcija omogućuje implementaciju zaštite mrežnog prometa između dvije aplikacije. Pomoću GSS biblioteke programer može implementirati takvu zaštitu, a da uopće ne zna koji sigurnosni mehanizam će se koristiti za fizičku zaštitu podataka. GSS je neovisan o platformi na kojoj se izvodi i o korištenom komunikacijskom protokolu.
GSS biblioteka trenutno podržava samo rad s Kerberos sigurnosnim mehanizmom, ali u budućnosti se planira podrška za razne druge sigurnosne sustave kao što je RSA algoritam za asimetrično šifriranje podataka. Isto tako trenutno postoji samo C implementacija biblioteke funkcija, ali u budućnosti se očekuje izvedba biblioteke u jeziku Java.
GSS biblioteka omogućuje autentikaciju korisnika te tako stvara sigurnosno okružje u kojem dvije aplikacije mogu komunicirati. Osim toga, GSS podržava i enkripciju podataka te generiranje *hash* sažetaka za provjeru autentičnosti podataka.
GSS biblioteka funkcija ne pruža fizički mehanizam zaštite podataka i ne daje certifikate za autentikaciju korisnika, što je zadaća sigurnosnog mehanizma poput Kerberos-a.
GSS biblioteka također ne omogućuje način prijenosa paketa podataka i ne analizira sadržaj paketa, što je zadaća aplikacije koja koristi GSS. Navedena aplikacija mora sama oslobađati memoriju koja je rezervirana korištenjem GSS funkcija.
GSS sučelje se automatski brine za izbor određenog sigurnosnog mehanizma za zaštitu podataka, ali korisnik može posebno navesti specifičan sigurnosni sustav koji želi koristiti. Iako takva mogućnost postoji, njeno korištenje nije preporučljivo jer se na taj način onemogućava prenosivost koda na platforme gdje konkretni sigurnosni mehanizam nije podržan.

3.1. Tijek komunikacije

Komunikacija između pošiljatelja i primatelja poruka preko GSS standarda ima slijedeće faze:

1. Obje strane u komunikaciji moraju dobiti potrebne sigurnosne certifikate.
2. Pošiljatelj šalje zahtjev za uspostavljanjem sigurnosnog okružja između dvije aplikacije i primatelj ga prihvaća.
3. Pošiljatelj koristi ugrađene funkcije za enkripciju i digitalno potpisivanje poruka i zatim šalje tako obrađene poruke.
4. Primatelj dekriptira i provjerava potpis poruka.
5. Ako je potrebno, primatelj šalje svoju identifikacijsku oznaku što je znak primatelju da je poruka uspješno poslana.
6. Obje strane uništavaju sigurnosno okružje i sve preostale GSS podatke.

3.2. Sigurnosni certifikati

Certifikati (engl. *Security Credentials*) su posebne podatkovne strukture koje potvrđuju identitet nekog računala u komunikaciji preko GSS standarda. Certifikate ne generira GSS mehanizam već ih generiraju sigurnosni mehanizmi u nižem komunikacijskom sloju. U podatkovnoj strukturi nekog certifikata svaki sigurnosni mehanizam generira vlastitu identifikacijsku oznaku. Za potvrdu identiteta je dovoljno da sustav prepozna jednu od brojnih sigurnosnih oznaka unutar certifikata pa će sigurnosni certifikat generiran na jednom sustavu biti prepoznat na svakom drugom sustavu koji ima instaliran neki podskup sigurnosnih mehanizama sa prvog sustava.
Aplikacija može dobiti sigurnosni certifikat pomoću funkcije `gss_acquire_cred()`, a moguće je koristiti i inicijalno postavljeni certifikat. Način korištenja inicijalno postavljenog certifikata će biti detaljnije opisan u slijedećem poglavlju.

Svaki certifikat ima određeno vremensko ograničenje, nakon čega se više ne može koristiti. Trajanje certifikata se može navesti kao parametar u funkciji `gss_acquire_cred()`.

3.3. Uspostavljanje sigurnosnog okružja

Sigurnosno okružje je par GSS-API podatkovnih struktura u kojima aplikacije spremaju identifikacijske podatke o računalu s kojim komuniciraju. Sigurnosno okružje se uspostavlja tako da aplikacije razmijene sigurnosne certifikate opisane u prethodnom poglavlju. Između dvije aplikacije može postojati i više od jednog sigurnosnog okružja.

Sigurnosno okružje se uspostavlja tako da ga jedna aplikacija (klijent) zahtijeva od druge (poslužitelj), a druga aplikacija ga mora prihvatiti.

Klijent zahtijeva sigurnosno okružje pozivanjem funkcije `gss_init_sec_context()`, a poslužitelj ga prihvaća pozivanjem funkcije `gss_accept_sec_context()`. Ovisno o potrebnoj sigurnosnoj razini, poslužitelj može zahtijevati više od jedne razmjene certifikata, zbog čega je obje funkcije potrebno pozivati u petlji. U svakom prolazu kroz petlju dvije aplikacije izmijene sigurnosne certifikate, a poslužitelj odlučuje kada je broj razmjena dovoljan i o tome obavještava klijentsku aplikaciju.

Funkcija `gss_init_sec_context()` kao argument zahtijeva sigurnosni certifikat dobiven na način opisan u prošlom poglavlju, a osim toga od važnijih argumenata zahtijeva ulaznu poruku dobivenu kao odgovor od poslužitelja u prethodnoj iteraciji te pokazivač na strukturu u koju će biti pohranjena izlazna poruka za slanje poslužitelju. U prvom prolazu kroz petlju se argument koji sadrži odgovor poslužitelja iz prethodnog poziva postavlja na `GSS_C_NO_BUFFER`.

Ako aplikacija koristi inicijalno postavljene certifikate (što je i preporučeni način korištenja) onda argument funkcije koji sadrži sigurnosni certifikat treba postaviti na vrijednost `GSS_C_NO_CREDENTIAL` i tada nije potrebno posebno generirati certifikate, kao što je to opisano u prethodnom poglavlju.

Funkcija `gss_accept_sec_context()`, koju koristi poslužitelj, ima slične argumente kao i funkcija `gss_init_sec_context()`. Funkcija također zahtijeva sigurnosni certifikat koji je ili generiran funkcijom `gss_acquire_cred()` ili postavljen na inicijalnu vrijednost.

Ova funkcija preko pokazivača vraća podatkovnu strukturu koju je potrebno poslati klijentu, a taj argument se u prvom prolazu kroz petlju postavlja na vrijednost `GSS_C_NO_BUFFER`.

Kada daljnja razmjena certifikata nije potrebna, funkcija će vratiti vrijednost `GSS_S_COMPLETE`, a inače vraća vrijednost `GSS_S_CONTINUE_NEEDED`. Osim toga, kada je izmjena certifikata gotova, funkcija će kao podatak za slanje klijentu vratiti niz duljine nula.

Funkcija `gss_init_sec_context()`, koju koristi klijentska aplikacija, će vratiti vrijednost `GSS_S_COMPLETE`, nakon što od poslužitelja primi niz duljine nula, nakon čega i klijentska aplikacija prekida svoju petlju za razmjenu certifikata.

3.4. Napredne mogućnosti za autentikaciju aplikacija

Uspostavljanje sigurnosnog okružja na način opisan u prethodnom poglavlju je najjednostavniji slučaj i koristi inicijalne postavke. Obje funkcije kao argument primaju razne zastavice pomoću kojih je moguće detaljno podesiti način uspostavljanja sigurnosnog okružja. Te mogućnosti su:

- Delegiranje certifikata – omogućava da se poslužitelj ponaša kao *proxy* poslužitelj. Kada poslužitelj od klijenta primi zahtjev za uspostavljanjem okružja on ga prosljeđuje nekom drugom računalu. Za korištenje te opcije potrebno je podesiti zastavicu `GSS_C_DELEG_FLAG`.
- Obostrana autentikacija – ovisno o namjeni softvera može se zahtijevati obostrana autentikacija klijentskih i poslužiteljskih aplikacija ili je moguće zahtijevati samo predstavljanje klijenta poslužitelju. Obostrana autentikacija se postiže postavljanjem zastavice `GSS_C_MUTUAL_FLAG`. Neki sigurnosni mehanizmi uvijek rade obostranu autentikaciju, neovisno o tome je li ona zatražena ili nije. Klijentska aplikacija mora sama prekinuti daljnju komunikaciju ako nije moguće ostvariti obostranu autentikaciju, GSS to ne radi automatski.
- Provjera redoslijeda i ponavljanja podataka – GSS može automatski provjeriti da li paketi za autentikaciju aplikacija stižu u pravilnom redoslijedu te da li dolazi do dupliciranja poslanih paketa. Provjera redoslijeda se postiže pomoću zastavice `GSS_C_SEQUENCE_FLAG`, a provjera ponavljanja paketa pomoću zastavice `GSS_C_REPLAY_FLAG`.

- Anonimna autentikacija – klijent se može predstaviti poslužitelju bez da poslužitelj zna pravi identitet klijenta. Klijent u tom slučaju šalje certifikat kojim dokazuje svoje pravo pristupa resursima na poslužitelju, ali ne šalje podatke o svom identitetu. Zastavica koju je potrebno podesiti za korištenje te opcije je: `GSS_C_ANON_FLAG`.
- Uvoz i izvoz certifikata – koriste se u više-procesnim aplikacijama, npr. kada poslužitelj u jednom procesu sluša dolazne konekcije klijenata, a zatim otvara novi proces za svakog pojedinog klijenta u kojem se odvija sva daljnja komunikacija. Za izvoz i uvoz certifikata postoje dvije posebne funkcije. Proces koji izvozi certifikat poziva funkciju `gss_export_sec_context()`. Ta funkcija najprije isključuje sigurnosno okružje u tom procesu i generira podatkovnu strukturu koju je moguće poslati drugom procesu. Sigurnosno okružje se mora isključiti jer ono može istovremeno postojati samo u jednom procesu. Proces koji uvozi certifikat poziva funkciju `gss_import_sec_context()`, koja otvara sigurnosno sučelje u novom procesu.

3.5. Informacije o sigurnosnom okružju

Pomoću funkcije `gss_inquire_context()` aplikacija može saznati informacije o već kreiranim sigurnosnim okružjima. Funkcija vraća ime klijenta, ime poslužitelja, sigurnosni mehanizam na kojem se bazira sigurnosno okružje, preostalo vrijeme u sekundama u kojem je okružje važeće te razne zastavice koje su korištene kod uspostavljanja okružja. Pomoću funkcije je također moguće saznati da li je okružje potpuno uspostavljeno te da li je aplikacija koja je pozvala funkciju klijent ili poslužitelj.

3.6. Zaštita podataka

Nakon što je uspostavljeno sigurnosno okružje na način opisan u poglavljima 3.2 – 3.4, moguće je jednostavno slati podatke između dvije aplikacije. Aplikacije su obavile autentikaciju i to pruža određeni nivo zaštite podataka koji se šalju. Međutim, GSS biblioteka ima još dva mehanizma koji omogućuju dodatni nivo zaštite i to su: kriptiranje i potpisivanje poruka.

Za potpisivanje poruka se koristi MIC (engl. *Mechanism Integrity Code*), što je poseban *hash* niz koji je različit za svaku poruku i primatelj poruke može provjeriti da li je poruka stigla s odgovarajućim MIC nizom. MIC se može generirati pomoću funkcije `gss_get_mic()`, a primatelj može provjeriti ispravnost poruke pomoću funkcije `gss_verify_mic()`. Funkcija `gss_get_mic()` generira MIC niz zasebno od poruke i aplikacije moraju razmijeniti oba podatka. Aplikacija koja prima poruke mora također sama raspoznati koja od primljenih struktura je poruka, a koja je pripadni MIC niz.

Enkripcija podataka se provodi pomoću funkcije `gss_wrap()`, a dekripcija pomoću funkcije `gss_unwrap()`. Ta funkcija, osim enkripcije podataka, može automatski dodati i MIC niz za provjeru autentičnosti poruke. Funkcija kao jedan od argumenata prima zastavicu koja određuje da li će funkcija provoditi samo enkripciju ili će dodati i MIC niz.

Funkcije `gss_get_mic()` i `gss_wrap()` obje mogu kreirati MIC, a razlika je u tome što prva funkcija kreira MIC odvojen od izvorne poruke, a druga funkcija ga zajedno s porukom pohranjuje u istu strukturu te na kraju enkriptira.

Enkripcija poruke povećava njenu veličinu, a to povećanje ovisi o korištenom sigurnosnom mehanizmu i različito je u slučaju da je s porukom uključen i MIC. Problem je što poruka s tim povećanjem može prekoračiti maksimalnu veličinu paketa koja je dozvoljena u korištenom transportnom protokolu. Maksimalnu veličinu poruke koja će nakon enkripcije biti dovoljno mala moguće je saznati pomoću funkcije `gss_wrap_size_limit()`.

Funkcije `gss_verify_mic()` i `gss_unwrap()` vraćaju vrijednost `GSS_S_COMPLETE` ako je poruka uspješno provjerena, odnosno `GSS_S_BAD_SIG` ako je MIC kod neispravan.

3.7. Potvrda prijema

GSS nema poseban mehanizam za potvrdu prijema poruka, ali ga je lako implementirati tako da primatelj poruke pošalje njen MIC natrag pošiljatelju. Ako je poruka primljena zajedno s MIC nizom, primatelj ga može jednostavno vratiti natrag, a ako nije onda ga mora generirati sam.

3.8. Oslobađanje memorije

Nakon što je završen transfer podataka aplikacije moraju osloboditi svu memoriju koja je bila rezervirana za GSS funkcije.

Memorija rezervirana za pohranu podataka sigurnosnog okružja se oslobađa pomoću funkcije `gss_delete_sec_context()`. Funkcija `gss_release_cred()` oslobađa svu memoriju rezerviranu za sigurnosne certifikate ako nije korišten inicijalni certifikat. Za oslobađanje ostale rezervirane memorije služe funkcije: `gss_release_buffer()`, `gss_release_name()` i `gss_release_oid_set()`.

4. Instalacija softvera

GSS C biblioteka funkcija je trenutno dostupna samo za UNIX/Linux operacijski sustav i zadnja trenutno dostupna stabilna verzija biblioteke je 0.0.10.
Biblioteka je besplatna (dostupna je pod GNU licencom) i može se skinuti sa adresa:
ftp://alpha.gnu.org/gnu/gss/ i http://josefsson.org/gss/releases/.
Biblioteka dolazi u izvornom kodu i prije instalacije ju je potrebno prevesti u binarni kod. Sam postupak prevođenja i instalacije je isti kao i kod većine Linux paketa. Za prevođenje koda prvo je potrebno podesiti parametre koji su specifični za operacijski sustav na tom računalu. To je moguće napraviti pomoću naredbe *./configure*. Samo prevođenje se radi pomoću naredbe *make*, a instalacija paketa pomoću naredbe *make install*.
Konfiguraciju i prevođenje paketa može napraviti svaki korisnik na sustavu, ali za instalaciju su potrebne administratorske ovlasti.

5. Primjer korištenja GSS standarda

U ovom primjeru su prikazana dva kratka odsječka koda koji koriste GSS funkcije. Prvi odsječak koda prikazuje realizaciju klijentske aplikacije, a drugi prikazuje realizaciju poslužitelja. Odsječci koda nisu gotove aplikacije i nije ih moguće jednostavno prevesti u izvršni kod. Ispis koda gotove aplikacije bi bio predugačak i nepraktičan za potrebe ovog dokumenta.
Sva potrebna objašnjenja uz ispis koda su napisana kao komentari uz dio koda na koji se odnose.

5.1. Klijentska aplikacija

```
int call_server(char *host, u_short port, gss_OID oid, char *service_name, OM_uint32
        deleg_flag, char *msg, int use_file) {
    gss_ctx_id_t context;
    gss_buffer_desc in_buf, out_buf, context_token;
    int s, state;
    OM_uint32 ret_flags;
    OM_uint32 maj_stat, min_stat;
    gss_name_t src_name, targ_name;
    gss_buffer_desc sname, tname;
    OM_uint32 lifetime;
    gss_OID mechanism, name_type;
    int is_local;
    OM_uint32 context_flags;
    int is_open;
    gss_qop_t qop_state;
    gss_OID_set mech_names;
    gss_buffer_desc oid_name;
    int i;
    int conf_req_flag = 0;
    int req_output_size = 1012;
    OM_uint32 max_input_size = 0;
    char *mechStr;

    //Otvaranje sigurnosnog okružja
    if (client_establish_context(s, service_name, deleg_flag, oid, &context,
            &ret_flags) < 0) {
        (void) close(s);
        return -1;
    }

    //Informacije o sigurnosnom okružju
    maj_stat = gss_inquire_context(&min_stat, context, &src_name, &targ_name,
        &lifetime, &mechanism, &context_flags, &is_local, &is_open);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("inquiring context", maj_stat, min_stat);
        return -1;
    }
```

```
if (maj_stat == GSS_S_CONTEXT_EXPIRED) {
     printf(" context expired\n");
     display_status("Context is expired", maj_stat, min_stat);
     return -1;
}
// Testiranje maksimalne veličine poruke
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
     GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
} else fprintf (stderr, "Iznos koji vraća gss_wrap_size_limit\n");
conf_req_flag = 1;
maj_stat = gss_wrap_size_limit(&min_stat, context, conf_req_flag,
     GSS_C_QOP_DEFAULT, req_output_size, &max_input_size);
if (maj_stat != GSS_S_COMPLETE) {
    display_status("wrap_size_limit call", maj_stat, min_stat);
} else fprintf (stderr, "Iznos koji vraća gss_wrap_size_limit\n");
maj_stat = gss_display_name(&min_stat, src_name, &sname, &name_type);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("displaying source name", maj_stat, min_stat);
     return -1;
}
maj_stat = gss_display_name(&min_stat, targ_name, &tname, (gss_OID *) NULL);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("displaying target name", maj_stat, min_stat);
     return -1;
}

(void) gss_release_name(&min_stat, &src_name);
(void) gss_release_name(&min_stat, &targ_name);
(void) gss_release_buffer(&min_stat, &sname);
(void) gss_release_buffer(&min_stat, &tname);
maj_stat = gss_oid_to_str(&min_stat, name_type, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("converting oid->string", maj_stat, min_stat);
     return -1;
}

(void) gss_release_buffer(&min_stat, &oid_name);
//Ispitivanje podržanih imena
maj_stat = gss_inquire_names_for_mech(&min_stat, mechanism, &mech_names);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("inquiring mech names", maj_stat, min_stat);
     return -1;
}
maj_stat = gss_oid_to_str(&min_stat, mechanism, &oid_name);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("converting oid->string", maj_stat, min_stat);
     return -1;
}
mechStr = (char *)__gss_oid_to_mech(mechanism);

(void) gss_release_buffer(&min_stat, &oid_name);
for (i=0; i < mech_names->count; i++) {
     maj_stat = gss_oid_to_str(&min_stat, &mech_names->elements[i], &oid_name);
     if (maj_stat != GSS_S_COMPLETE) {
          display_status("converting oid->string", maj_stat, min_stat);
          return -1;
     }
     (void) gss_release_buffer(&min_stat, &oid_name);
}
(void) gss_release_oid_set(&min_stat, &mech_names);
if (use_file) {
    read_file(msg, &in_buf);
} else {
     in_buf.value = msg;
     in_buf.length = strlen(msg) + 1;
}
if (ret_flags & GSS_C_CONF_FLAG) state = 1;
else state = 0;
maj_stat = gss_wrap(&min_stat, context, 1, GSS_C_QOP_DEFAULT, &in_buf,
     &state, &out_buf);
if (maj_stat != GSS_S_COMPLETE) {
     display_status("wrapping message", maj_stat, min_stat);
     (void) close(s);
```

```
            (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
            return -1;
        } else if (! state) {
            fprintf(stderr, "Warning! Message not encrypted.\n");
        }
        //Salje posluzitelju
        if (send_token(s, &out_buf) < 0) {
            (void) close(s);
            (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
            return -1;
        }
        (void) gss_release_buffer(&min_stat, &out_buf);
        //Čita potpis u buffer
        if (recv_token(s, &out_buf) < 0) {
            (void) close(s);
            (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
            return -1;
        }
        //Provjera potpisa
        maj_stat = gss_verify_mic(&min_stat, context, &in_buf, &out_buf, &qop_state);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("verifying signature", maj_stat, min_stat);
            (void) close(s);
            (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
            return -1;
        }
        (void) gss_release_buffer(&min_stat, &out_buf);
        if (use_file) free(in_buf.value);
        printf("Signature verified.\n");
        //Brisanje sigurnosnog okružja
        maj_stat = gss_delete_sec_context(&min_stat, &context, &out_buf);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("deleting context", maj_stat, min_stat);
            (void) close(s);
            (void) gss_delete_sec_context(&min_stat, &context, GSS_C_NO_BUFFER);
            return -1;
        }
        (void) gss_release_buffer(&min_stat, &out_buf);
        (void) close(s);
        return 0;
}

int client_establish_context(int s, char *service_name, OM_uint32 deleg_flag,
            gss_OID oid, gss_ctx_id_t *gss_context, OM_uint32 *ret_flags) {
        gss_buffer_desc send_tok, recv_tok, *token_ptr;
        gss_name_t target_name;
        OM_uint32 maj_stat, min_stat;

        send_tok.value = service_name;
        send_tok.length = strlen(service_name) + 1;
        maj_stat = gss_import_name(&min_stat, &send_tok,
            (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &target_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("parsing name", maj_stat, min_stat);
            return -1;
        }

        token_ptr = GSS_C_NO_BUFFER;
        *gss_context = GSS_C_NO_CONTEXT;
        do {
            maj_stat = gss_init_sec_context(&min_stat, GSS_C_NO_CREDENTIAL, gss_context,
                target_name, oid, GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG | deleg_flag,
                0, NULL, token_ptr, NULL, &send_tok, ret_flags, NULL);
            if (gss_context == NULL) {
                printf("Cannot create context\n");
                return GSS_S_NO_CONTEXT;
            }
            if (token_ptr != GSS_C_NO_BUFFER)
                (void)gss_release_buffer(&min_stat, &recv_tok);
            if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
                display_status("initializing context", maj_stat, min_stat);
                (void) gss_release_name(&min_stat, &target_name);
                return -1;
            }
            if (send_tok.length != 0) {
```

```
            if (send_token(s, &send_tok) < 0) {
                (void) gss_release_buffer(&min_stat, &send_tok);
                (void) gss_release_name(&min_stat, &target_name);
                return -1;
            }
        }
        (void) gss_release_buffer(&min_stat, &send_tok);
        if (maj_stat == GSS_S_CONTINUE_NEEDED) {
            fprintf(stdout, "continue needed...");
            if (recv_token(s, &recv_tok) < 0) {
                (void) gss_release_name(&min_stat, &target_name);
                return -1;
            }
            token_ptr = &recv_tok;
        }
        printf("\n");
    } while (maj_stat == GSS_S_CONTINUE_NEEDED);
    (void) gss_release_name(&min_stat, &target_name);
    return 0;
}
```

5.2. Poslužiteljska aplikacija

```
int server_acquire_creds(char *service_name, gss_OID mechOid,
        gss_cred_id_t *server_creds) {
    gss_buffer_desc name_buf;
    gss_name_t server_name;
    OM_uint32 maj_stat, min_stat;
    gss_OID_set_desc mechOidSet;
    gss_OID_set desiredMechs = GSS_C_NULL_OID_SET;
    if (mechOid != GSS_C_NULL_OID) {
        desiredMechs = &mechOidSet;
        mechOidSet.count = 1;
        mechOidSet.elements = mechOid;
    } else desiredMechs = GSS_C_NULL_OID_SET;
    name_buf.value = service_name;
    name_buf.length = strlen((const char *)name_buf.value) + 1;
    maj_stat = gss_import_name(&min_stat, &name_buf,
        (gss_OID) GSS_C_NT_HOSTBASED_SERVICE, &server_name);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("importing name", maj_stat, min_stat);
        if (mechOid != GSS_C_NO_OID) gss_release_oid(&min_stat, &mechOid);
        return -1;
    }
    maj_stat = gss_acquire_cred(&min_stat, server_name, 0, desiredMechs,
        GSS_C_ACCEPT, server_creds, NULL, NULL);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("acquiring credentials", maj_stat, min_stat);
        return -1;
    }
    (void) gss_release_name(&min_stat, &server_name);
    return 0;
}
int sign_server(int s, gss_cred_id_t server_creds) {
    gss_buffer_desc client_name, xmit_buf, msg_buf;
    gss_ctx_id_t context;
    OM_uint32 maj_stat, min_stat;
    int i, conf_state, ret_flags;
    char *cp;
    //Uspostavljanje sigurnosnog konteksta
    if (server_establish_context(s, server_creds, &context, &client_name,
        (OM_uint32 *)&ret_flags) < 0) return(-1);
    (void) gss_release_buffer(&min_stat, &client_name);
    for (i=0; i < 3; i++)
        if (test_import_export_context(&context)) return -1;
    //Primanje enkriptirane poruke
    if (recv_token(s, &xmit_buf) < 0) return(-1);
    if (verbose && log) {
        fprintf(log, "Wrapped message token:\n");
        print_token(&xmit_buf);
    }
    maj_stat = gss_unwrap(&min_stat, context, &xmit_buf, &msg_buf, &conf_state,
        (gss_qop_t *) NULL);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("unwrapping message", maj_stat, min_stat);
```

```
            return(-1);
        } else if (! conf_state) {
            fprintf(stderr, "Warning! Message not encrypted.\n");
        }
        (void) gss_release_buffer(&min_stat, &xmit_buf);
        fprintf(log, "Received message: ");
        cp = (char *)msg_buf.value;
        if (isprint(cp[0]) && isprint(cp[1])) fprintf(log, "\"%s\"\n", cp);
        else {
            printf("\n");
            print_token(&msg_buf);
        }
        //Kreira potpis poruke
        maj_stat = gss_get_mic(&min_stat, context, GSS_C_QOP_DEFAULT,
            &msg_buf, &xmit_buf);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("signing message", maj_stat, min_stat);
            return(-1);
        }
        (void) gss_release_buffer(&min_stat, &msg_buf);
        //Salje potvrdu primitka poruke
        if (send_token(s, &xmit_buf) < 0) return(-1);
        (void) gss_release_buffer(&min_stat, &xmit_buf);
        //Briše sigurnosno okružje
        maj_stat = gss_delete_sec_context(&min_stat, &context, NULL);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("deleting context", maj_stat, min_stat);
            return(-1);
        }
        fflush(log);
        return(0);
}
int server_establish_context(int s, gss_cred_id_t server_creds,
            gss_ctx_id_t *context, gss_buffer_t client_name, OM_uint32 *ret_flags) {
        gss_buffer_desc send_tok, recv_tok;
        gss_name_t client;
        gss_OID doid;
        OM_uint32 maj_stat, min_stat;
        gss_buffer_desc oid_name;
        char *mechStr;
        *context = GSS_C_NO_CONTEXT;
        do {
            if (recv_token(s, &recv_tok) < 0) return -1;
            if (verbose && log) {
                fprintf(log, "Received token (size=%d): \n", recv_tok.length);
                print_token(&recv_tok);
            }
            maj_stat = gss_accept_sec_context(&min_stat, context, server_creds,
                &recv_tok, GSS_C_NO_CHANNEL_BINDINGS, &client, &doid, &send_tok,
                ret_flags, NULL, NULL);
            if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
                display_status("accepting context", maj_stat, min_stat);
                (void) gss_release_buffer(&min_stat, &recv_tok);
                return -1;
            }
            (void) gss_release_buffer(&min_stat, &recv_tok);
            if (send_tok.length != 0) {
                if (verbose && log) {
                    print_token(&send_tok);
                }
                if (send_token(s, &send_tok) < 0) {
                    fprintf(log, "failure sending token\n");
                    return -1;
                }
                (void) gss_release_buffer(&min_stat, &send_tok);
            }
            if (verbose && log) {
                if (maj_stat==GSS_S_CONTINUE_NEEDED) fprintf(log, "continue needed\n");
                else fprintf(log, "\n");
                fflush(log);
            }
        } while (maj_stat == GSS_S_CONTINUE_NEEDED);
        //Prikaz zastavica
        display_ctx_flags(*ret_flags);
        if (verbose && log) {
```

```
        maj_stat = gss_oid_to_str(&min_stat, doid, &oid_name);
        if (maj_stat != GSS_S_COMPLETE) {
            display_status("converting oid->string", maj_stat, min_stat);
            return -1;
        }
        mechStr = (char *)__gss_oid_to_mech(doid);

        (void) gss_release_buffer(&min_stat, &oid_name);
    }
    maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
    if (maj_stat != GSS_S_COMPLETE) {
        display_status("displaying name", maj_stat, min_stat);
        return -1;
    }
    return 0;
}
```

6. Nedostaci GSS standarda

Mnogi programeri smatraju kako GSS C biblioteka funkcija pruža nezgrapno i neintuitivno programersko sučelje. Dvije glavne zamjerke koje se tu pojavljuju su:

- Sve GSS funkcije daju informaciju o uspješnom izvršavanju odnosno nastaloj pogrešci preko dvije 32-bitne cjelobrojne vrijednosti. Prva vrijednost je sama povratna vrijednost funkcije, a druga se vraća preko pokazivača koji je uvijek prvi argument funkcije. Takvo rješenje je nezgrapno jer se prvi parametar funkcije mora navesti čak i kada povratna vrijednost nije bitna (funkcija ne dopušta NULL pokazivač na mjestu prvog argumenta). Osim toga, logično je da prvi argument funkcije bude najvažniji parametar, a ne neki koji se najčešće ni ne koristi. Puno bolje rješenje bi bilo da funkcija vraća pokazivač na posebnu strukturu u kojoj bi bile sadržane sve poruke o pogreškama.
- GSS ima lošu kontrolu nad memorijom. GSS ima svoje funkcije za oslobađanje podataka unutar posebnih struktura kao što je `gss_buffer_desc`, ali korisnik je sam odgovoran za oslobađanje memorije koje zauzimaju te iste strukture, tj. nema funkcije koja oslobađa memoriju na koju pokazuje `gss_buffer_t` pokazivač. Takav način upravljanja memorijom je kompliciran i česte su programerske pogreške koje mogu dovesti do nestabilnosti aplikacije i do sigurnosnih propusta. Najbolje rješenje bi bilo da ugrađene GSS funkcije potpuno upravljaju sa svojim strukturama podataka, tj. da same rezerviraju i oslobađaju potrebnu memoriju.

7. Zaključak

Kao što je opisano u prethodnim poglavljima, GSS standard ima razne nedostatke i vjerojatno nije najjednostavnije postojeće sučelje za implementaciju zaštite podataka u mrežnim aplikacijama. Sve GSS funkcije ne moraju biti podržane u svim izvedbama i verzijama biblioteke i to je vjerojatno još jedan razlog slabe popularnosti GSS biblioteke.
Usprkos nedostacima, GSS biblioteka funkcija znatno olakšava rad s Kerberos sustavom, a vjerojatno će postati puno popularnija u praksi kad će postojati podrška za dodatne sigurnosne mehanizme. Isto tako, popularnost GSS-a bi mogla porasti kada se pojavi biblioteka funkcija za jezik Javu.

8. Reference

RFC dokument sa opisom GSS standarda: ftp://ftp.isi.edu/in-notes/rfc2743.txt
C implementacija GSS biblioteke: ftp://ftp.isi.edu/in-notes/rfc2744.txt
Sun priručnik za upotrebu GSS sučelja: http://docs.sun.com/db/doc/816-1331
GNU priručnik za upotrebu GSS sučelja: http://www.gnu.org/software/gss/manual/index.html