Vysoké učení technické v Brně

Fakulta informačních technologií

Síťové aplikace a správa sítí

2015/16

Projekt – Klient SIP

Michal Melichar (xmelic17)

Brno, 22.11.2015

Obsah

1. Úvod 3
2. Analýza 4
   2.1. Zadání 4
   2.2. Protokol SIP 4
   2.3. Protokol UDP 4
   2.4. Hašovací funkce MD5 4
3. Popis implementace 5
   3.1. Globální proměnné 5
      3.1.1. Struktura *Tzprava* 5
      3.1.2. Struktura *Tbuf_zprava* 5
      3.1.3. Struktura *Tmd5* 5
      3.1.4. Struktura *Todeslana_zprava* 6
   3.2. Hlavní tělo programu 6
   3.3. Funkce definované v programu 6
      3.3.1. void vytvor_zpravu(Tzprava *matice, Tbuf_zprava *buffer, Tmd5 *md5_zprava, Todeslana_zprava *message, int rozhodovani) 6
      3.3.2. void ziskej_udaje(char hledany_vyraz[], Tzprava *matice, Tbuf_zprava *buffer, bool nonce) 7
      3.3.3. void printfERR(int error) 7
      3.3.4. void vytvor_md5(Tzprava *matice, Tmd5 *md5_zprava) 8
      3.3.5. void generuj_hodnoty(Tzprava *matice, Todeslana_zprava *message, int i) 8
      3.3.6. void cislo_odpovedi(Tbuf_zprava *buffer) 8
      3.3.7. void vypis_odpovedi(Tzprava *matice, int typ_tisku, Tbuf_zprava *buffer, Todeslana_zprava *message) 9
      3.3.8. void zpracuj_signal(int cislo_signalu) 9
      3.3.9. void nastav_vychozi_hodnoty(Tzprava *matice, Todeslana_zprava *message) 9
      3.3.10. bool ziskat_udaje(char hled_vyraz[], char vstup[], char vystup_tmp[]) 10
      3.3.11. int zpracuj_soubor(char *soubor, Tzprava *matice) 10
      3.3.12. void uvolni_pamet() 10
      3.3.13. void zpracuj_port(char ip_adresa[], char port[]) 10
      3.3.14. void zjisti_moji_ip(char *moje_ip) 10
4. Použité zdroje 11

1. Úvod

V rámci projektu jsem implementoval SIP klienta, který se připojí k serveru pomocí autentizačního mechanismu MD5 digest. Klient udržuje spojení se serverem. Klient dále odesílá zprávy, které jsou umístěné v souboru. V případě požadavku se klient korektně odhlásí od serveru a ukončí spojení.

2. Analýza

2.1. Zadání

Výsledný program podporující verzi protokolu SIP 2.0 napsaný v jazyce C, se dokáže připojit k SIP serveru. Provést autentizaci pomocí protokolu MD5 digest. Pokud byl zadaný parametr –m a soubor obsahující zprávy, tak provede jejich odeslání. Klient udržuje spojení se serverem do té doby, dokud neobdrží signál SIGTERM, SIGQUIT nebo SIGINT. Po obdržení některého z toho signálu dojde ke korektnímu odhlášení od serveru a následně ukončení programu. Klient podporuje transportní protokol UPD a využívá pro síťovou komunikaci BSD sokety, pro autentizaci využívá knihovnu OpenSSL.

2.2. Protokol SIP

SIP neboli Session Initiation Protocol je internetový protokol určený pro přenos přenos signalizace v internetové telefonii. Můžeme sem zařadit multimediální konference, hovory přes IP síť, sdílení multimediálních dat. Klasicky používá UDP port 5060, ale může fungovat i nad TCP/5060. Koncepce protokolu je podobná jako u HTTP (Hypertext Transfer Protocol), WWW (World Wide Web) nebo SMTP (Simple Mail Transfer Protocol).

2.3. Protokol UDP

UDP neboli User Datagram Protocol je jeden z internetových protokolů. UDP protokol nám nezaručuje, že se přenášený datagram po cestě neztratí, nezmění se pořadí doručených datagramů nebo zda nebude některý datagram doručen vícekrát. Jeho výhoda spočívá v jednoduchosti oprati protokolu TCP. Je výhodné jej použít v aplikacích, které pracují na systému otázka – odpověď. Hlavní přínos je především pro jednodušší implementaci serverů, kde nevadí, když dojde ke ztrátě některých datagramů.

2.4. Hašovací funkce MD5

MD5 (Message-Digest algorithm) je hašovací funkce, která vytváří z libovolných vstupních dat výstupy pevné délky. Jeho hlavní vlastností je, že malá změna na vstupu vede k velké změně na výstupu (k velkému rozdílu výsledného řetězce). Z MD5 výstupu nelze zpět spočítat vstupní hodnoty. Princip fungování v SIP autentizaci je vygenerování MD5 posloupnosti na straně klienta, přenos po síti a porovnání s vygenerovanou MD5 posloupností na straně serveru. Heslo se tak nikdy neodesílá v otevřené formě.

3. Popis implementace

3.1. Globální proměnné

Program využívá globální proměnné, které jsou využity k řízení chování programu při určitých situacích a ty nastavují jednotlivé funkce. Dále jsou přítomny 4 struktury, které obsahuji proměnné využívání ke zpracování údajů získaných buď se souborů, nebo při zpracování odpovědí přijatých od serveru.

3.1.1. Struktura *Tzprava*

Struktura obsahující informace potřebné k sestavení registrační zprávy a její odeslání na server. Do sktruktury jsou ukládané informace získané ze souboru zadaného s parametrem *–p*. Definice struktury:

```
typedef struct {
	char user[MAX_BUFFER_3];	=> Název uživatele
	char password[MAX_BUFFER_3];	=> Heslo
	char tag[MAX_BUFFER];		=> Náhodně generovaný Tag
	char call_id[MAX_BUFFER];	=> Náhodně generované Call-ID
	char branch[MAX_BUFFER_2];	=> Náhodně generovaný branch
	char realm[MAX_BUFFER_2];	=> Hodnota realm získaná od serveru při registraci
	char nonce[MAX_BUFFER_2];	=> Hodnota nonce získaná od serveru při registraci
	char *moje_ip;			=> Ukazatel na mou IP adresu
	unsigned int muj_port;		=> Číslo portu, ze kterého odesíláme zprávy
	char muj_port_char[6];		=> Číslo portu převedené na řetězec
	unsigned int port_server;	=> Číslo portu, na kterém naslouchá server
	char port_server_char[6];	=> Číslo portu převedené na řetězec
	char cilova_ip[MAX_BUFFER_3];	=> IP adresa serveru
	bool odpoved;			=> Bool hodnota, nastavena v programu automaticky
	unsigned int cseq;		=> Hodnota obsahující číslo požadavku na server
	unsigned long int expires;	=> Doba expirace registrace na serveru
	char metoda[9];			=> Uložení názvu metody v odesílané zprávě
}Tzprava;
```

3.1.2. Struktura *Tbuf_zprava*

Struktura pro uložení informací získaných během běhu programu.

```
typedef struct {
	char zaslana_zprava[MAX_BUFFER_ZPRAVA];	=> Buffer, pro uložení sestavené zprávy
	char prijata_zprava[MAX_BUFFER_ZPRAVA];	=> Buffer, pro uložení odpovědi serveru
	int typ_odpovedi;			=> Číselná hodnota odpovědi serveru
	char nazev_odpovedi[MAX_BUFFER_2];	=> Název odpovědi serveru na požadavek
}Tbuf_zprava;
```

3.1.3. Struktura *Tmd5*

Struktura obsahující průběžné hodnoty vzniklé přu generování MD5 posloupnosti a výsledné MD5 autorizační zprávy odeslané na server.

```
typedef struct{
	char ha_1[33]; => První část MD5 posloupnosti (user:realm:password)
	char ha_2[33]; => Druhá část MD5 posloupnosti metoda:sip:user@server)
	char md5[33];  => MD5(user:realm:password):nonce:MD5(metoda:sip:user@server)
}Tmd5;
```

3.1.4. Struktura *Todeslana_zprava*

Struktura obsahuje data sloužící k odeslání zprávy (MESSAGE) na server.

```
typedef struct {
	char zprava[MAX_BUFFER_ZPRAVA]; => Text zprávy odeslané příjemci
	char adresa[MAX_BUFFER_3];      => Adresa, na kterou odesíláme zprávu
	unsigned int adresa_port;       => Číslo portu, na který odesíláme zprávu
	char adresa_port_char[6];       => Číslo portu převedené na řetězec
	char tag[MAX_BUFFER];           => Nová hodnota Tagu
	char call_id[MAX_BUFFER];       => Nová hodnota Call-ID
	char branch[MAX_BUFFER_2];      => Nová hodnota branch
}Todeslana_zprava;
```

3.2. Hlavní tělo programu

V hlavní části programu se nejprve provede zpracování zadaných parametrů programů. Především získání názvů souborů u parametrů *–m* a *–p*. Dále je zde řešeno chybné zadaní parametrů, zpracování souboru s údaji o uživateli. Následně dojde k vytvoření a nastavení socketu a přiřazení konkrétního čísla portu, ze kterého budeme odesílat požadavky a zprávy. Program odesílá požadavky ze socketu 32866. Pokud se nepovede odeslání socketu, program vypíše chybu a skončí.

Pokud je adresa serveru na localhostu, poté se nastaví i adresa klienta na localhost. V opačném případě je nutné zjistit adresu klienta přes funkci *getnameinfo()*. Před samotným zahájením komunikace se serverem je nutné ještě vygenerovat hodnoty Tag, Call-ID a branch, následně ještě nastavit zpracování signálů SIGTERM, SIGQUIT a SIGINT. Poté už nekonečný cyklus, který se ukončí právě zachycením daných signálů. V případě požadavku na ukončení programu dojde nejdříve ke správnému odhlášení na serveru. Během cyklu *While()* se provádí odeslání požadavků na registraci, udržovaní spojení nebo odeslání zprávy.

Udržování spojení se serverem se provádí pomocí definice signálu alaram. Kde v případě vypršení časového limitu dojde k odeslání žádosti na prodloužení registrace na serveru.

V případě, že klient nedostane odpověď na požadavek nebo potvrzení doručení zprávy, bere to jako, že se datagram poškodil nebo ztratil po cestě a provede odeslání datagramu znovu. Celkem program může odeslat stejnou zprávu 3x, poté vypíše chybu a daný datagram dále ignoruje.

3.3. Funkce definované v programu

3.3.1. void vytvor_zpravu(Tzprava *matice, Tbuf_zprava *buffer, Tmd5 *md5_zprava, Todeslana_zprava *message, int rozhodovani)

Funkce slouží k vytvoření registrační zprávy nebo zprávy MESSAGE, kterou budeme odesílat na server. V rámci jejich parametrů se předávají 3 struktury, které jsou globálně definované. Proměnná *rozhodovani* slouží k určení o jaký typ zprávy se bude jednat.

Hodnoty proměnné *rozhodovani*:	**Význam**
0	Slouží k sestavení zprávy **REGISTER** bez řádku **Authorization**
1	Slouží k sestavení zprávy **REGISTER** s údaji obsaženými v řádku **Authorization** a vygenerovanou **MD5** posloupností
2	Slouží k sestavení zprávy **MESSAGE**

3.3.2. void ziskej_udaje(char hledany_vyraz[], Tzprava *matice, Tbuf_zprava *buffer, bool nonce)

Funkce slouží především k vyhledání údajů *nonce* a *realm* v odpovědi serveru. V rámci funkce se dynamicky alokuje pole, které má délku od umístění hledaného výrazu ve zprávě až po konec zprávy. Poté dojde ke zpracování hledané hodnoty, jenž se nachází mezi znaky “. Na závěr dojde k přidání nulového znaku na konec nalezeného řetězce. Poté se nastaví vyhledaná hodnota do patřičné proměnné ve struktuře Tzprava.

Proměnná *nonce* slouží k určení zda se hledá hodnota *nonce* a nebo *realm*.

Hodnota proměnné nonce	Význam
bool	Hledaná hodnota je typu *nonce*
true	Hlednaá hodnota je typu *realm*

3.3.3. void printfERR(int error)

Funkce slouží k chybovému výpisu na standartní chybový výstup. Parametrem funkce je výčtový typ *enum*. Funkce dále pracuje s polem, která obsahuje seznam chybových hlášení. Index pole odpovídá výčtovému typu *enum*.

Definice chyb, které mohou nastat.

E_PARAMETR	"ERROR: Spatne zadane parametry programu."
E_MEMORY	"ERROR: Doslo k chybe pri alokaci pameti."
E_OPEN_FILE	"ERROR: Soubor se nepodarilo otevrit."
E_CLOSE_FILE	"ERROR: Soubor se nepodarilo koretkne uzavrit."
E_READ_FILE	"ERROR: Doslo k chybe pri zpracovani souboru."
E_MAX_BUFFER	"ERROR: Doslo k vnitrni chybe programu."
E_ZPRAVA	"ERROR: Pripojeni k SIP serveru se nezdarilo."
E_ODPOVED_SERVER	"ERROR: Doslo k chybe na strane serveru. Program nenalezl odpoved serveru."
E_TYP_ODPOVED	"ERROR: Server odeslal spatny typ odpovedi."
E_BIND	"ERROR: Doslo k chybe pri vazbe socketu na lokalni port."
E_METOD	"ERROR: SIP server neprijal vas pozadavek."
E_SERVER	"ERROR: V souboru profiles.txt je spatne zadan server."
E_USER	"ERROR: V souboru profiles.txt je spatne zadane jmeno uzivatele."
E_PASSWORD	"ERROR: V souboru profiles.txt je spatne zadane heslo

E_PORT	uzivatele." "ERROR: Spatne cislo portu nebo je port spatne zadan."

3.3.4. void vytvor_md5(Tzprava *matice, Tmd5 *md5_zprava)

Funkce slouží k vygenerování MD5 posloupnosti podle zadaných vlastností. Vygenerovaná hodnota se využívá k autorizaci na SIP serveru. Funkci se předávají dva parametry. První parametr je struktura obsahující všechny potřebné údaje ke správnému vygenerování MD5 posloupnosti. Druhý parametr je struktura obsahující vygenerovanou MD5 strukturu a její pomocné mezi kroky.

MD5 se generuje podle následujícího klíče:

- ✓ Nejprve je potřeba spočítat hodnotu hashovací funkce řetězce HA1: **user:realm:password**
- ✓ Druhým krokem je spočítat hodnotu hashovací funkce řetězce HA2: **metoda:sip:user@server**
- ✓ Poledním krokem je vypočet hodnoty hashovací funkce pro řetězec: **HA1:nonce:HA2**

Celkově tedy: **MD5(user:realm:password):nonce:MD5(metoda:sip:user@server)**

V rámci funkce je také potřeba řešit převod MD5 posloupnosti v 16ckové soustavě na řetězec. Toho docílíme pomocí funkce *snprintf()*.

3.3.5. void generuj_hodnoty(Tzprava *matice, Todeslana_zprava *message, int i)

Funkce slouží k vygenerování náhodných hodnot *Tag, CALL-ID, branch*. Toho docílíme pomocí funkce *rand()*. Výsledná náhodná hodnota je řetězec, na který jsme převedli číslo pomocí funkce *snprintf().* Proměnná *i* slouží k určení pro jaký účel potřebujeme generovat hodnoty.

Hodnota proměnné *i*	Význam
1	Pro generování hodnot do zpráv **REGISTER**
else	Pro generování hodnot do zpráv **MESSAGE**

U hodnoty branch je nutné ještě přidat pevnou posloupnost (z9hG4bK) na začátek výsledné náhodné hodnoty. Hodnoty Tag, Call-ID nic takového nepotřebují.

3.3.6. void cislo_odpovedi(Tbuf_zprava *buffer)

Funkce slouží k získání hodnoty odpovědi serveru a názvu této odpovědi. Ve funkci je definovaný řetězec, který předchází námi hledané hodnotě. Pomocí funkce *strstr()* najdeme pozici, kde začíná v přijaté zprávě naše zpracovávaná hodnota. Poté pomocí pointrové arimetriky se posuneme na začátek naší hodnoty a provedeme zpracování hodnoty odpovědi i název této odpovědi. Obě nalezené hodnoty uložíme do struktury *buffer*.

3.3.7. void vypis_odpovedi(Tzprava *matice, int typ_tisku, Tbuf_zprava *buffer, Todeslana_zprava *message)

Funkce slouží pro správný výpis na standartní výstup. Parametry funkce jsou tři struktury, které obsahují informace potřebné k výpisu. Proměnná *typ_tisku* slouží k určení, které informace budeme vypisovat, tzn. o jaký druh výpisu se bude jednat.

Hodnota proměnné *typ_tisku*	Význam
1	Jedná se o výpis požadavku na registraci na server REGISTER *<sekvenční_číslo> <název_požadavku > <zdrojová_sip_adresa> <cílová_sip_adresa>*
2	Jedná se o výpis odpovědi serveru na náš požadavek *<sekvenční_číslo> < číslo_a_název_odpovědi> <zdrojová_sip_adresa> <cílová_sip_adresa>*
3	Jedná se o výpis v případě zprávz MESSAGE *<sekvenční_číslo> <název_požadavku > <zdrojová_sip_adresa> <cílová_sip_adresa> <zpráva>*
4	Jedná se o výpis požadavku na odhlášení od serveru UNREGISTER *<sekvenční_číslo> <název_požadavku > <zdrojová_sip_adresa> <cílová_sip_adresa>*

Při výpisu je také nutné rozhodnou zda součástí výpisu bude také číslo portu. To se vypisuje pouze v případě, že je jiné než standartní port 5060.

3.3.8. void zpracuj_signal(int cislo_signalu)

Funkce slouží obsluze odchytnutého signálu. Celkem zpracováváme 3 signály (SIGTERM, SIGQUIT, SIGINT, SIGALRM). Signál SIGALRM slouží k obsluze doby registrace na serveru. Zbylé signály slouží ke správnému ukončení programu a odhlášení klienta od serveru. Proměnnou *číslo_signalu* předáváme signál, který vyvolal přerušení provádění programu.

3.3.9. void nastav_vychozi_hodnoty(Tzprava *matice, Todeslana_zprava *message)

Funkce provádí úvodní inicializaci některých proměnných. Proměnné předávají dvě struktury, u kterých budeme provádět nastavení výchozí hodnotu určitých proměnných. Provádí se zde také převod číselných hodnot na řetězce a uložení do proměnných. Dále se zde provádí kontrola, zda nebylo zadané nesmyslné číslo portu serveru.

Konkrétně zde nastavuje tyto proměnné: tag, call_id, branch, odpověď, cseq, metoda, muj_port, muj_port_char, port_server_char.

3.3.10. bool ziskat_udaje(char hled_vyraz[], char vstup[], char vystup_tmp[])

Funkce slouží k vyhledání řetězce ve větším poli. V našem případě se jedná o jeden přečtený řádek ze souboru. Funkce nalezne klíčové slovo. Poté se pomocí pointrové aritmetiky posune za klíčové slovo a získá jeho hodnotu. Funkce nalezenou hodnotu uloží do předaného pole. Funkce má návratovou hodnotu bool , která určuje, zda proběhlo vyhledání hodnoty správně nebo zda nedošlo během zpracování k chybě.

3.3.11. int zpracuj_soubor(char *soubor, Tzprava *matice)

Funkce provede zpracování souboru zadaného parametrem *–p*. Provede vyhledání odpovídajících sekvencí a uloží je do vhodných proměnných ve struktuře. V případě chybějícího údaje *expires* nastaví jeho hodnotu na 3600. Po dokončení zpracování souboru, provede jeho uzavření.

3.3.12. void uvolni_pamet()

Funkce má na starost uvolnění dynamicky alokované paměti a nastavení uvolněných ukazatelů na hodnotu NULL.

3.3.13. void zpracuj_port(char ip_adresa[], char port[])

Funkce má na starosti rozdělení adresy a portu. Následně uloží obě hodnoty do správných proměnných. Původně totiž adresa obsahuje zároveň i port. To je předané v parametru *ip_adresa*. Funkce provede zpracování tohoto pole, jeho vynulovaní a uložení ve správním formátu. Parametr *port* slouží k uložení hodnoty portu. Pokud nebylo nalezeno číslo portu v adrese, tak se nastaví výchozí port 5060.

3.3.14. void zjisti_moji_ip(char *moje_ip)

Funkce slouží k zjištění IP adresy, na které běží klient. Původní funkce je převzata z manuálových stránek funkce *getnameinfo()*. Byla však pro naše potřeby upravena, zredukována a doplněna.

Funkce vyhledá IPv4 rozhraní a z toho získá IP adresu rozhraní, na kterém běží klient. Rozhraní localhostu, IPv6 ignoruje.

4. Použité zdroje

RFC: 768 http://www.ietf.org/rfc/rfc768.txt

RFC:3261 (http://tools.ietf.org/html/rfc3261)

RFC:3428 (http://www.tools.ietf.org/html/rfc3428)

MD5 autentizace protokolu SIP (http://www.kapejod.org/en/2013/02/reversing-sip-digest-authentication-in-javascript-node-js)

Odpovědi SIP http://www.3cx.com/global/cz/voip-sip-webrtc/sip-responses/