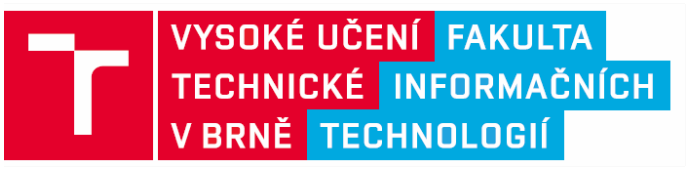
Vysoké učení technické v Brně

Fakulta informačních technologií



ISA - Síťové aplikace a správa sítí

Klient POP3 s podporou TLS

xkanda00 - Rastislav Kanda

Obsah

1 Úvod do problematiky

Elektronická pošta *skr. Email* je základným pilierom medzi službami prístupnými na Internete. Svojim významom sa v praktickom živote dostala na úroveň klasickej pošty, no v prípade flexibility ju bezo sporu prekonáva.

O všetko sa stará sieť Emailových serverov, z čoho je každý z nich počítačový ekvivalent klasického roznášača pošty. Tu sa dostávame k pojmu schránka. Každý užívateľ emailového serveru, má vytvorený účet, teda má založenú schránku na emailovom serveri. Na to aby mohol do schránky pristupovať slúži emailový klient. Klient zabezpečuje všetko potrebné aby si užívateľ mohol pohodlne prezerať správy. Klient môže byť v rôznych formách, či už aplikácii pre príkazový riadok, aplikácia s grafickým rozhraním alebo webová aplikácia s grafickým rozhraním vo webovom prehliadači.

Jeden z protokolov ktorý dnes zabezpečuje emailovú komunikáciu je `POP3`, čo je skratkou z anglického *Post Office Protocol, 3. verzia*[1]. POP3 server je možno prirovnať k reálnej pošte ktorá udržiava emailovú správu do doby kým si ju človek nevyzdvihne *(nevymaže)*. POP3 je dnes najviac preferovaný poštový protokol, hlavne vďaka tomu že je možný urobiť tú istú prácu ako ostatné, ale s najmejším počtom zlyhaní. POP3 protokol funguje na báze sťahovania, teda človek odošle pomocou TCP/IP požiadavok, aby mu server preposlal emaily, ktoré sa nachádzajú v jeho schránke a následne ich možno odstrániť. POP3 je vhodný pre užívateľov ktorý nemajú stály prístup k internetu, resp. majú ho časovo obmedzený. Na druhej strane existuje tu nemožnosť filtrovania správ a teda zo servera môžu byť stiahnuté aj nevyžiadané správy, *spam*.

2 Zadanie projektu

Zadaním projektu bolo vytvoriť POP3 klienta ktorý bude podporovať zabezpečenú komunikáciu pomocou TLS/SSL šifrovania.
Klient na štandardný vstup dostáva vo forme parametrov:

- parameter *<server>* - adresu servera vo forme IPv4 alebo IPv6 adresy
- parameter *-p <port>* - číslo portu na ktorom má naviazať komunikáciu so serverom
- parameter *-T/-S* - prepínač, ktorý zabezpečuje naviazanie šifrovaného spojenia - bližšie špecifikovaný v sekcii 3.3 na strane 5
- parameter *-c<cert_file>* - umiestnenie súbora *certfile*, ktorý sa použije na overenie platnosti certifikátu prijatého zo serveru
- parameter *-C <cert_dir>* - cestu k priečinku *certdir* v ktorom sa vyhľadávajú certifikáty na overenie platnosti certifikátu prijatého zo serveru
- parameter *-d-* prepínač, ktorý zašle správu serveru na zmazanie správ
- paramater *-n* - prepínač, ktorý zabezpečí že klient bude pracovať iba s novými správami - kontrola novosti správ bližšie špecifikovaná v sekcii 3.4.4 na strane 6
- parameter *-a <auth_file>* - cestu k súboru s autentizačnými údajmi
- parameter *-o <out_dir>* - cestu k priečinku kde sa budú ukladať správy stiahnuté zo serveru

[1] (Wikipedia 2017 *Post Office Protocol — Wikipedia, The Free Encyclopedia*, `http://sk.wikipedia.org/w/index.php?title=Post\%20Office\%20Protocol&oldid=6439846`, [Online; accessed 17-November-2017])

3 Implementácia

3.1 Spracovanie argumentov

Na spracovávanie argumentov slúži trieda *parser{}* v ktorej sú naimplementované všetky metódy týkajúce sa overenie argumentov a rozhranie cez ktoré je možno sa na jednotlivé položky dotazovať.
Argumenty sú načítavané zo štandardného vstupu a ukladané do dočasného poľa argumentov aby mohli byť spracované jeden po druhom a nevynechal sa žiaden.
O všetko spracovanie sa stará metóda `void parse_arg()`, ktorá je volaná v konštruktore triedy parser.

Základom celej kontroly argumentov je stavový automat, ktorý skontroluje po poradí každý argument a podľa obsahu argumentu naplní základné dátové štruktúry.

Popis metód triedy parser:

- `void getAuthData()` - metóda, ktorá načíta dáta zo súboru a zapíše prihlasovacie údaje do predpripravených triednych premenných
- `long returnPort()` - metóda vráti číslo portu, ktoré bolo zadané argumentom *-p*
- `string returnServer()` - metóda vráti adresu servera
- `string getUsername()` - pomocou metódy sa možno dotázať na prihlasovacie meno uložené zo súboru
- `string getPassword()` - pomocou metódy sa možno dotázať na prihlasovacie heslo uložené zo súboru
- `string getOutdir()` - metóda vráti zadanú cestu k priečinku pre ukladanie správ
- `string getCertdir()` - metóda vráti cestu k priečinku, kde sú uložené certifikačné súbory
- `string getCertfile()` - metóda vráti cestu k súboru s certifikátom
- `bool secureStart()` - metóda vráti hodnotu typu *boolean*, ktorá slúži ako prepínač šifrovanej komunikácie
- `bool do_delete()` - metóda vráti hodnotu, ktorá určuje či sa odošle požiadavok na zmazanie správ
- `bool do_secure()` - metóda vráti hodnotu, ktorá určuje či sa serveru odošle požiadavok na zašifrovanie komunikácie pomocou STARTTLS
- `bool is_just_new_msgs()` - metóda vráti hodnotu, ktorá určuje či sa bude pracovať iba s novými správami
- `bool hostname_to_ip(const string &hostname)` - metóda slúžiaca na overenie platnosti adresy servera, v triede *parser{}* pracuje spoločne s funkciou *getaddrinfo()*

Pri akejkoľvek chybe argumentov sa vypíše nápoveda a stavový riadok o tom ktorý argument bol zle zadaný.

3.2 Naviazanie spojenia

Naviazanie spojenia so serverom zabezpečuje rovnomenná metóda
`mySocket::connect()/mySecuredSocket::connect()` či už v triede *mySecuredSocket* alebo *mySocket*.

V nej dochádza v prvom rade ku zisteniu adresy servera pomocou funkcie `getaddrinfo()` kde pri chybe, pretože je predpoklad že uživateľ zadal názov serveru a nie ip adresu, je volaná metóda `hostname_to_ip()`, ktorá zistí či zadaný server je preložiteľný na ip adresu a následne opätovne zavolá funkciu `getaddrinfo()`, ktorá nám vráti informácie o danom serveri.

Pokiaľ po druhej kontrole funkciou nastane chyba, adresa je pravdepodobne zlá a nie je možné pripojiť sa na zadanú adresu. Teda program sa ukončí.

Najpodstatnejšia informácia v tejto fáze je pre nás typ ip adresy *(ai_family)*, pretože podľa typu sa rozhodne aký socket sa má vytvoriť, či typu IPv4 alebo IPv6. Po úspešnom vytvorení socketu je nainicializovaná štruktúra socketu správnymi údajmi a následné pripojenie na server funkciou `::connect()` *(z knižnice sys/socket.h)*

V prípade šifrovaného spojenia prebehne inicializácia šifrovania spojenia.

3.3 Šifrované spojenie TLS/STLS

Ak si užívateľ vyžiada zašifrovanie spojenia hneď v úvode, prepínačom *-T* je po nešifrovanom pripojení inicializovaná štruktúra pre spojenie so serverom šifrovane, na porte 995 pre šifrovanú komunikáciu[2].

Všetka komunikácia prebieha výhradne cez funkcie `SSL_read()/SSL_write()`.

Ak užívateľ zadá argument *-S* je spojenie naviazané nešifrovaným spôsobom, na porte 110 pre nešifrovanú komunikáciu. Ihneď po uvítacej správe je serveru zaslaný príkaz `STLS` - pred príkazmi pre prihlásenie - kedy server začne očakávať od klienta naviazanie šifrovaného spojenia na tom istom porte. Všetka ďalšia komunikácia prebieha šifrovane pomocou funkcii `SSL_read()/SSL_write()`.

Ďalej je potreba nainicializovať metódu spojenia, v mojom prípade TLSv1.2 a takisto aj kontext šifrovaného spojenia pomocou funkcie `SSL_CTX_new()`.

Z tohoto kontextu je následne vytváraná štruktúra SSL ktorá je nositeľom všetkých informácii pre šifrované spojenie.

V ďalšom kroku je potreba overiť certifikáty serveru a porovnať ich s certifikátmi, ktoré uživateľ zadal ako cestu k súboru argumentom. Toto zabezpečuje funkcia `SSL_CTX_load_verify_locations()`, kde parametrom môže byť buď cesta k súboru typu *.pem* kde je uložený certifikát alebo cesta k priečinku kde sú uložené súbory s certifikátmi ktoré sa majú použiť na overenie.

Ak nie je zadaný ani jeden z argumentov *-c / -C* tak prebehne autentifikácia pomocou funkcie `SSL_CTX_set_default_verify_paths()`.

Následne je previazaný socket ktorý sme si vytvorili na začiatku s SSL štruktúrou pomocou `SSL_set_fd()` a pomocou funkcie `SSL_connect()` je naviazané šifrované spojenie.

Testovanie autentizácie certifikátu prebieha pomocou funkcie `SSL_get_verify_result()`.

[2](Chris Newman 1999 *Using TLS with IMAP, POP3 and ACAP*, RFC 2595, `http://www.rfc-editor.org/rfc/rfc2595.txt`, RFC Editor, `http://www.rfc-editor.org/rfc/rfc2595.txt`)

3.4 Komunikácia

3.4.1 Prihlásenie na server

Po nadviazaní potrebného spojenia, prebieha prihlásenie pomocou kombinácie príkazov `USER username` a `PASS password`, kde autentizačné údaje sú načítavané zo súboru z ktorého trieda `parser` vyextrahovala potrebné informácie.

3.4.2 Komunikácia so serverom

Komunikácia s POP3 serverom funguje na báze zasielania správ a prijímania odpovedí[3]. Najprv je potreba zistiť koľko správ sa na serveri nachádza. Klient odošle príkaz `STAT` a čaká na odpoveď v ktorej sa nachádza počet správ na serveri. Následne v cykle stiahne každú správu zo servera, príkazom `RETR <int>` kde int je číslo správy na serveri a správu uloží do zadaného priečinka pre ukladanie správ.

Ak je zvolený parameter, ktorý určuje prácu iba s novými správami (*-n*) tak sa pri každej správe skontroluje novosť správy, (sekcia 3.4.4 na strane č.6) pred uložením. Ak je zvolený parameter *-d* tak pre každú správu ktorá je uložená sa odošle príkaz `DELE <int>`, kde int je číslo správy na serveri, čo zaobstará vymazanie správy zo serveru (ale iba pri úspešnom ukončení spojenia príkazom QUIT). Správy ktoré už existujú, pri zapnutom prepínači *-n*, sa neukladajú, takže sa ani nemažú zo servera.

3.4.3 Formát ukladania správ

Každá správa ktorá prichádza z nejakého externého servera, tzn. niekto ju odoslal, obsahuje v hlavičke unikátny identifikátor `Message-ID:<>`[4]. Pri ukladaní správy sa vytvára súbor ktorý ma v názve práve tento unikátny identifikátor.
Formát: `Msg ID [some_unique_characters@email_server.suffix]`, kde v hranatých zátvorkách je práve tento unikátny identifikátor (hranaté zátvorky nie sú súčasťou názvu súboru).
V nepravdepodobnom prípade, ktorý ale môže nastať, sa na koniec názvu pridá vlastný identifikátor `-<int>` kde int je celé číslo. Napríklad ak správa neobsahuje Message-Id (uvítacia správa z vlastného serveru), v tom prípade správy môžu vyzerať ako: `Msg ID, Msg ID-1, Msg ID-2-1`.
V prípade že sťahujeme opätovne správy, bez prepínača *-n*, správy sa môžu ukladať opätovne ale je im priradený vlastný identifikátor za pomlčkou.
Napríklad: `Msg ID 04EC5270-D1B4-48DC-91D6-EA50AE31D461@gmail.com-1`.

3.4.4 Kontrola novosti správ

Pri kontrole novosti správ sa využíva práve kontrola `Message-Id`, ak sa v danom priečinku už nachádza správa s rovnakým Message-Id ako má práve sťahovaná správa, tak sa správa preskočí a pokračuje sa na ďalšiu správu.

[3](John G. Myers a Marshall T. Rose
1996 *Post Office Protocol - Version 3*, STD 53, `http://www.rfc-editor.org/rfc/rfc1939.txt`, RFC Editor, `http://www.rfc-editor.org/rfc/rfc1939.txt`)

[4](Peter W. Resnick
2008 *Internet Message Format*, RFC 5322, `http://www.rfc-editor.org/rfc/rfc5322.txt`, RFC Editor, `http://www.rfc-editor.org/rfc/rfc5322.txt`)

3.4.5 Ošetrenie javu dot-stuffed (*duplikovanie bodiek*)

Každý POP3 server prevádza kontrolu na znak `crlf.crlf`. Pritom, preventívne, prevádza kontrolu ak sa na začiatku riadku nachádza bodka, tak ju zduplikuje.[5] Toto je potrebné ošetriť v prípade klienta aby nedochádzalo k nekonzistentnej veľkosti stiahnutej správy. Teda klient taktiež kontroluje obsah správy riadok po riadku a vykonáva kontrolu či správa neobsahuje postupnosť znakov `crlf..` v tomto prípade došlo v najväčšom prípade k zdvojeniu znaku bodka a je potrebné prebytočný znak odstrániť. Na odstraňovanie znakov je použitý regulárny výraz a funkcia `regex_replace()`.

3.4.6 Odhlásenie zo serveru

Po ukončení všetkých operácii je potrebné pre sfinalizovanie všetkých operácií úspešné odhlásenie. To je prevedené pomocou príkazu `QUIT`.

Aplikácia po úspešnom odhlásení vypíše počet všetkých správ s ktorými operovala. Pri ukončení programu je vypísaná hláška ukončenia programu. Pri úspešnom ukončení je to počet stiahnutých správ, inak je vypísaná príslušná chybová hláška. Pri chybe argumentov je vypísaná nápoveda.

4 Nevýhody a nedostatky implementácie

- Je potreba stiahnuť celú správu do prepisovateľného bufferu, aby bolo možné zistiť či sa daná správa už nachádza v zadanom priečinku.
- Ak je klientovi zadaný priečinok ktorý neexistuje, ale klient nestiahne žiadnu správu zo serveru, priečinok bude aj tak vytvorený.

[5](J. Klensin 2008 *Simple Mail Transfer Protocol*, RFC 5321, `http://www.rfc-editor.org/rfc/rfc5321.txt`, RFC Editor, `http://www.rfc-editor.org/rfc/rfc5321.txt`)

Odkazy

Klensin, J.

2008 *Simple Mail Transfer Protocol*, RFC 5321, http://www.rfc-editor.org/rfc/rfc5321.txt, RFC Editor, http://www.rfc-editor.org/rfc/rfc5321.txt.

Myers, John G. a Marshall T. Rose

1996 *Post Office Protocol - Version 3*, STD 53, http://www.rfc-editor.org/rfc/rfc1939.txt, RFC Editor, http://www.rfc-editor.org/rfc/rfc1939.txt.

Newman, Chris

1999 *Using TLS with IMAP, POP3 and ACAP*, RFC 2595, http://www.rfc-editor.org/rfc/rfc2595.txt, RFC Editor, http://www.rfc-editor.org/rfc/rfc2595.txt.

Resnick, Peter W.

2008 *Internet Message Format*, RFC 5322, http://www.rfc-editor.org/rfc/rfc5322.txt, RFC Editor, http://www.rfc-editor.org/rfc/rfc5322.txt.

Wikipedia

2017 *Post Office Protocol — Wikipedia, The Free Encyclopedia*, http://sk.wikipedia.org/w/index.php?title=Post\%20Office\%20Protocol&oldid=6439846, [Online; accessed 17-November-2017].