Vysoké učení technické
v Brně
Fakulta informačních
technologií

Síťové aplikace a správa sítí

Discord Bot

November 17, 2020

Michal Koval(xkoval17)

Contents

1 Zadanie

Vytvořte program isabot, který bude působit jako bot na komunikační službě Discord. Bot se připojí na Discord server na kanál "#isa-bot" a bude reagovat na všechny zprávy zaslané ostatními uživateli. Bot bude fungovat jako echo pro všechny zprávy, které zachytí. V případě, že bot na daném kanále zachytí jakoukoli zprávu jiného uživatele (tedy jinou než svou vlastní) a zároveň, která není jiného bota (uživatelské jméno neobsahuje podřetězec "bot"), odešle tuto zprávu zpátky na kanál a to ve formátu "echo: <username> - <message>" (kde <username> představuje uživatelské jméno uživatele, který odeslal původní zprávu).

2 Úvod do problematiky

Na vypracovanie tohto projektu je dôležité mať istý teoretický základ. Treba ovládať zasielnie správ na aplikačnej vrstve internetu - HTTP. Treba poznať TLS šifrovanie, pretože Discord ho používa a nedá sa s ním komunikovať, bez toho aby sme správy správne zašifrovali a rozšifrovali. Ďalej je potrebné sa zoznámiť s REST API. Discord REST Api zasiela vo svojich odpovediach na požiadavky v dáta v JSON formát. Takže je potrebné vedieť s týmto formátom pracovať.

2.1 HTTP

HTTP je protokol definujúci požiadavky a odpovede medzi klientmi a servermi. HTTP klient (označovaný ako user agent), ako webový prehliadač zvyčajne začne požiadavku nadviazaním TCP spojenia na určenom porte vzdialeného stroja (štandardne port 80). HTTP server počúvajúci na danom porte čaká, kým klient pošle reťazec s požiadavkou ako "GET / HTTP/1.1" (ktorý žiada o zaslanie štartovacej stránky webservera) nasledovaný sériou hlavičiek opisujúcich detaily požiadavky a nasledovaných telesom ľubovoľných údajov. Niektoré hlavičky sú nepovinné, zatiaľ čo verzia HTTP/1.1 niektoré vyžaduje (ako názov stroja). Po prijatí požiadavky server pošle reťazec s odpoveďou ako "200 OK" nasledovanou hlavičkami spolu so samotnou správou, ktorej telo tvorí obsah požadovaného súboru, chybové hlásenie alebo iná informácia.

Vývoj HTTP koordinovalo World Wide Web Consortium a pracovné skupiny Internet Engineering Task Force, čím vytvorili sadu dokumentov RFC, predovšetkým RFC 2616 definujúci HTTP/1.1, dnes používanú verziu HTTP

2.1.1 HTTP Dotaz

HTTP ponúka niekoľko metód, každú s iným účelom.

- GET Zďaleka najbežnejší typ žiadosti. Žiada o zdroj uvedením jeho URL
- POST Podobne ako GET, okrem toho, že je pridané telo správy zvyčajne obsahujúce dvojice kľúč-hodnota z HTML formulára a taktiež na upload súborov
- PUT Používa sa na úpravu, resp. editáciu údajov špecifikovaného objektu, podobne ako POST, pričom POST vytvára nový objekt (nové dáta) a PUT upravuje tieto dáta
- DELETE Zriedka implementované. Zmazanie zdroja
- HEAD Podobné GET, okrem toho, že sa nepožaduje telo správy, iba hlavičky. Používa sa na získavanie metainformácií o dokumente
- TRACE Odošle kópiu obdržanej požiadavky späť odosielateľovi, takže klient môže zistiť, čo na požiadavke menia alebo pridávajú servery, ktorými táto prechádza

- OPTIONS Vracia HTTP metódy, ktoré daný webserver podporuje. Je možné použiť na otestovanie funkcionality servera
- CONNECT Zriedka implementované, na použitie s proxy serverom, ktorý sa môže zmeniť na SSL tunel

Formát požiadavky je:

- 1 riadok s metodou požiadavky
- hlavičky, každá na osobitnom riadku
- prázdny riadok
- nepovinné telo požiadavky

Znaky koncov riadku musia byť <CR><LF>.
Príklad za formátu GET požiadavky z programu:

```
GET /api/v8/guilds/777464830239834152/channels HTTP/1.1
Host: discord.com
Accept: application/json
Connection: keep-alive
User-Agent: DiscordBot
Authorization: Bot NzY3NDc1MDM5NTMxOTU4Mjgy.X4yc1A.JOYK
```

2.1.2 HTTP Odpoveď

Odpoveď má podobný formát ako požiadavka:

- 1 riadok zobrazujúci status kód a príslušnú správu
- hlavičky, každá na osobitnom riadku
- prázdny riadok
- nepovinné telo požiadavky

Od HTTP/1.0 a neskôr, je prvým riadom odpovede status kód ("404") a textová fráza vysvetľujúca daný kôd ("Not Found"). Na základe status kódu klient primárne spracuváva odpoveď a sekundárne pomocou hlavičiek. Pomocou prvého čísla kódu delíme status kódy do 5 všeobecných tried:

- 1XX Informačné
- 2XX Úspech / OK
- 3XX Presmerovanie
- 4XX Chyba u klienta
- 5XX Chyba na strane serveru

2.2 TLS - Transport Layer Security

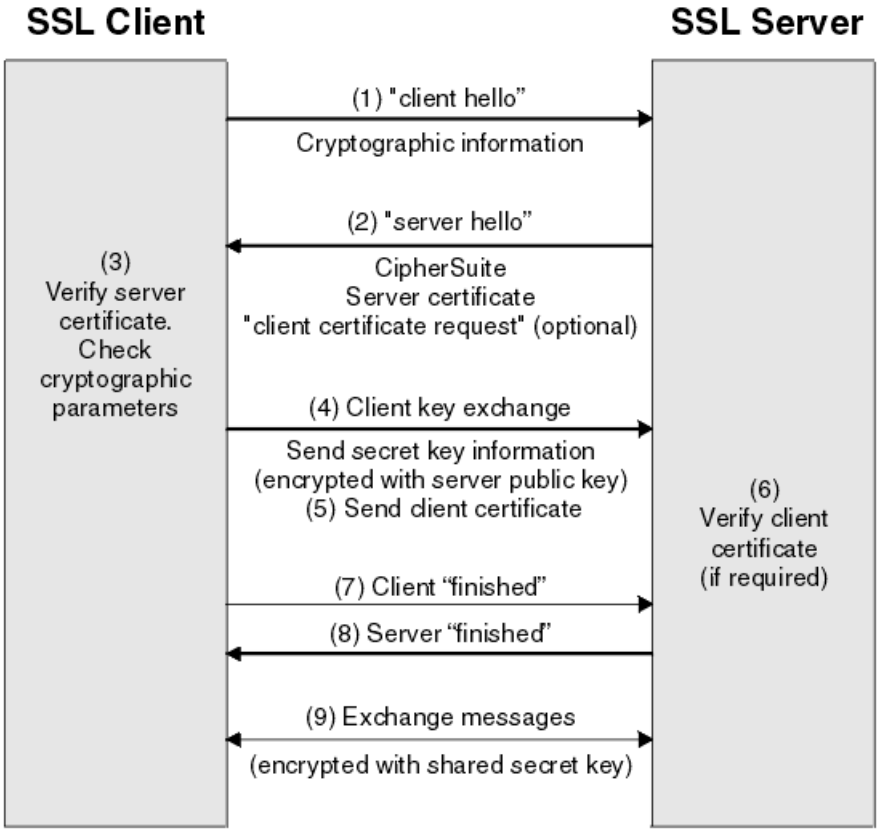
Protokol TLS umožňuje aplikáciám komunikovať cez sieť spôsobom akým bol navrhnutý, aby predchádzal odpočúvaniu, manipulácii, falšovaniu správ. TLS poskytuje koncobodovú autentifikáciu a súkromie v komunikácii cez internet používaním kryptografie. Typicky je autorizovaný len server (to znamená, že jeho identita je zaručená) zatiaľ čo klient ostáva neautorizovaný. To znamená, že koncový užívateľ, či už jednotlivec alebo aplikácia, si môže byť istý s kým komunikuje.

Ďalšia úroveň zabezpečenia, v ktorej obidve strany "konverzácie" si môžu byť isté s kým komunikujú, je známa ako obojstranná autorizácia. Obojstranná autorizácia vyžaduje infraštruktúru verejného kľúča.

2.2.1 Handshake

TLS klient a server dohodnú spojenie používaním procedúry podania rúk (handshake). Počas podania rúk sa klient a server dohodnú na rôznych parametroch používaných na vytvorenie bezpečnosti spojenia. Priebeh podania rúk:

- 1. Klient sa pripojí na server s povoleným TLS spojením a začne žiadať o bezpečné spojenie
- 2. Server vyberie zo zoznamu najsilnejšiu šifru a transformačnú funkciu, ktoré tiež podporuje a oznámi klientovi svoje rozhodnutie
- 3. Server pošle späť svoju identifikáciu z digitálneho certifikátu. Certifikát obyčajne obsahuje meno servera, dôveryhodnú certifikačnú autoritu a verejný šifrovací kľúč servera
- 4. Klient môže kontaktovať server, ktorý vydal certifikát (certifikačnú autoritu) a overiť si ešte predtým, ako začne proces, že certifikát je platný
- 5. V objednávke musí server vygenerovať kryptografický kľúč pre bezpečné spojenie. Klient zašifruje náhodné číslo pomocou verejného kľúča servera a pošle výsledok serveru. Dešifrovať ho dokáže iba server pomocou svojho súkromného kľúča. To zabezpečí, že kľúč ostane pre tretiu stranu utajený, lebo iba klient a server majú prístup k týmto dátam



2.3 API - Application programming interface

Rozhranie pre programovanie aplikácií je zbierka funkcií alebo tried, ktoré určujú akým spôsobom sa majú funkcie knižníc volať zo zdrojového kódu programu. API funkcie sú programové celky, ktoré programátor volá namiesto vlastného naprogramovania.

2.3.1 REST API

Representational state transfer je softwarový architektonický štýl, ktorý definuje pravidlá a obmedzenia pre tvorbu webových služieb. Služby, ktoré dodržiavajú tento štýl sa nazývajú RESTful webové služby. Tieto webové služby umožňujú klientskym systémom prístup k a manipuláciu webových prostriedkov danej služby pomocou uniformných a preddefinovaných bezstavových operácií. REST môže byť použítý pomocou akýchkoľvek protokolov, pri webových službách je využitý prevažne s HTTP. Pri zaslaní požiadavky na URI webového prostriedku, s ktorým chceme pracovať, server zašle odpoveď, v ktorej tele može dátový obsah formátovaný v HTML, JSON, XML atď.

2.4 JSON

JavaScript Object Notation je dátový formát nezávislý na počítačovej platforme, určený pre prenos dát, ktoré môžu byť organizované v poliach alebo agregované v objektoch. JSON dokáže poňať pole hodnôt (indexované aj neindexované), objekty (pole dvojíc index:hodnota) a jednotlivé hodnoty, ktoré môžu byť reťazce, čísla (celé, desatiné), pravdivostné hodnoty a prázdnu hodnotu (null). Príklad dát formátovaných v JSON:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

3 Návrh a implementácia

3.1 Hlavný modul

Hlavný modul implementovaný v `main.cpp` a `main.hpp` volá nosné fukcie ostatných modulov. Je v ňom napísaný základná logika aplikácie na vyššej abstrakčnej úrovni. Program sa začne spracovaním argumentov a pokračuje ďalej pripravením sieťových prvkov pre šifrovanú komunikáciu s Discordom. Následne z Discordu získame ID guildy v, ktorej je bot, a ID kanálu #isa-bot. Následne nasleduje nekonečná smyčka požiadaviek o nové správy a ich obsluha. Hlavná obsluhová slučka:

```
while (true) {
    // Get new messages since last message
    new_msgs = get_channel_messages(ssl_ctx, args.token, channel_info.channel_ID, last_ID,
                                    time);
    time = new_msgs.reset_time;
    if (new_msgs.msg_list.size() > 0){
        last_ID = new_msgs.msg_list.back().msg_ID;
        for (Msg_info x : new_msgs.msg_list) {
            msgs_to_send.push(x);
        }
    } else {
        if (msgs_to_send.size() == 0){
            // Sleep for second until next check of new messages
            // Required for handling of rate limiting and smooth user experience
            sleep(1);
            continue;
        }
    }
    // Reply to every new non bot message
    while (true) {
        if (msgs_to_send.size() == 0){
            break;
        }
        Msg_info x = msgs_to_send.front();
        msg_ret = send_msg(ssl_ctx, x.content, x.username, args.token,
                           channel_info.channel_ID, time);
        if (msg_ret.first != OK){
            break;
        }
        time = msg_ret.second;
        // Print to stdout if verbose is toggled
        if (args.verbose){
            cout << "isa-bot - " << x.username << " : " << x.content << endl;
        }
        msgs_to_send.pop();}}
```

3.2 Argparse

Cieľom moduku je spracovanie programových parametrov. Je implementovaný v `argparser.cpp` a `argparser.hpp`.

Hlavnou funkcia je `Args handle_args(int argc, char *argv[])`. Táto funkcia postupne skontroluje všetky programové parametre. Ak nájde nevyžiadaný parameter alebo viac krát zadaný ten istý parameter vypíše chybovú správu na štandardný chybový výstup a vráti chybu do mainu, ktorý ukončí program. V prípade zadania parametra na výpis pomocnej správy zavolá funkciu na jej výpis `void print_help()`.

3.3 Network

Tento modul implementuje tu časť programu, ktorá ma čokoľvek do činenia so sieťami a ktorá robí túto aplikáciu sieťovou aplikáciu. Implementovaný v súboroch `network.cpp` a `network.hpp`.

`SSL_CTX* create_CTX()` vytvorí a vráti štruktúru `SSL_CTX`, ktorá reprezentuje sieťový kontext aplikácie pre šifrovanú komunikáciu pomocou OpenSSL. Táto štruktúra sa má vytvárať iba raz za dobu života programu.

`SSL_conn setup_connection(SSL_CTX* ctx)` vytvorí nové zabezpečené spojenie s Discord serverom. Spojenie vytvára pomocou využia `BIO` z `openssl/bio.h`. Jedná sa o vyššiu vtupno výstupu abstrakciu. Vďaka tomu netreba manuálne vytvárať socket a spojenie, ktoré už BIO vytvorí samo.

`send_using_ssl(const char* message, BIO* bio_ssl, int reset_time)` zašle správu `message`, pomocou už vytvoreného TSL spojenia `bio_ssl`, ak je aktuálny čas aspoň `reset_time` v sekundách od Unixovej epochy. Toto čakanie je nutné z dôvodu rate-limitu, daným Discord serverom. To znamená, že môžme zasielať len niekoľko správ za niejakú časú dobu. Z rovnakého dôvodu nie je môžné aplikáciu spraviť tak, aby odpovede bota boli zdanlivo instantné, problém sa prejavu najmä pri príchode mnohých správ do kanálu vo veľmi krátkej časovej dobe.

3.4 Primitive json

Tento modul je nazvaný podľa jeho účelu. Slúži na veľmi jednoduché a cielené spracovanie a parsovanie JSON dát pre účely tejto aplikácie. Nie je to prenositeľný plnohodnotný JSON parser. Jadro jeho funkcionality je implementované pomocou regulárnych výrazov.

4 Spúšťanie programu

Prekladanie programu je pomocou Makefile a príkazu `make`, poprípade `gmake` na freeBSD. Spustenie je nasledujúce:

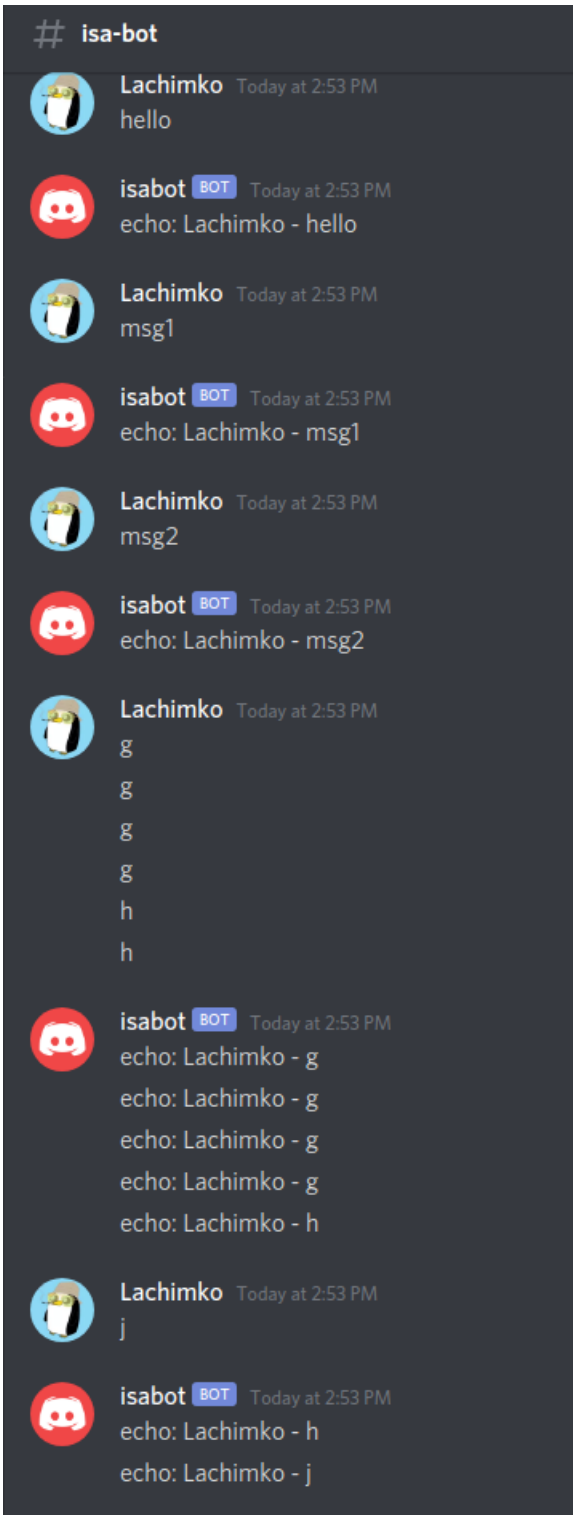
```
./isabot [-h|--help] [-v|--verbose] -t <bot_access_token>
-h|--help : vypíše nádpovedu na štandardný výstup
-v|--verbose : zapne vypisovanie správ, na ktoré bot reaguje, na štandardný
               výstup vo formáte  "<channel> - <username>: <message>".
-t <bot_access_token> : povinný argument autentizační token pre přístup
                        bota na Discord.

Poradie parametrov je ľubovoľné.
Spustenie programu bez parametrov zobrazí nápovedu.
```

Program sa ukončí vytvorením signálu SIGINT, teda stlačením kláves `CTRL+C`.

4.1 Príklad spustenia

```
$ ./isabot -v -t NzY3NDc1MDM5NTMxOTU4Mjgy.X4yc1A.JOYK0VPZBX1I1y7-0bSnlCNiYj8
isa-bot - Lachimko : hello
isa-bot - Lachimko : msg1
isa-bot - Lachimko : msg2
isa-bot - Lachimko : g
isa-bot - Lachimko : g
isa-bot - Lachimko : g
isa-bot - Lachimko : g
isa-bot - Lachimko : h
isa-bot - Lachimko : h
isa-bot - Lachimko : j
^C
Signal :2 - Terminating program
```


isa-bot
Lachimko Today at 2:53 PM
hello
isabot BOT Today at 2:53 PM
echo: Lachimko - hello
Lachimko Today at 2:53 PM
msg1
isabot BOT Today at 2:53 PM
echo: Lachimko - msg1
Lachimko Today at 2:53 PM
msg2
isabot BOT Today at 2:53 PM
echo: Lachimko - msg2
Lachimko Today at 2:53 PM
g
g
g
g
h
h
isabot BOT Today at 2:53 PM
echo: Lachimko - g
echo: Lachimko - g
echo: Lachimko - g
echo: Lachimko - g
echo: Lachimko - h
Lachimko Today at 2:53 PM
j
isabot BOT Today at 2:53 PM
echo: Lachimko - h
echo: Lachimko - j

References

[1] RFC 7235 - Hypertext Transfer Protocol (HTTP/1.1): Authentication https://tools.ietf.org/html/rfc7235

[2] RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content https://tools.ietf.org/html/rfc7231

[3] RFC 8259 - The JavaScript Object Notation (JSON) Data Interchange Format https://tools.ietf.org/html/rfc8259

[4] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1 https://tools.ietf.org/html/rfc2616

[5] Wikipedia Hypertext Transfer Protocol https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[6] Discord Developer Documentation https://discord.com/developers/docs/intro

[7] Wikipedia Transport Layer Security https://sk.wikipedia.org/wiki/Transport_Layer_Security

[8] Wikipedia Representational state transfer https://en.wikipedia.org/wiki/Representational_state_transfer