Dokumentacja SKJ3.0

Zadanie 2 port knocking

Co zostało zrealizowane:

* Struktura programu:
  + App
    - Klasa startowa, odpowiada za trzymanie zmiennych używanych przez interface i identyfikacji danej instancji aplikacji.
  + Chat
    - Klasa odpowiedzialna za obsługę czatu oraz komend.
  + Gate
    - Klasa odpowiedzialna za obsługę całej komunikacji z zewnątrz, zarówno przez UDP jak i TCP.
  + NetPath
    - (Dawniej Player) Klasa służąca jako opakowanie dla adresu sieciowego (ip-port).
  + Packet
    - (Dawniej Box) Klasa wykorzystywana jako medium do komunikacji pomiędzy klientami, jej obiekty są wysyłane i odbierane zarówno przez TCP jak i (opakowane w DatagramPacket) UDP. Jej metody statyczne natomiast służą ukryciu przed resztą aplikacji sposobu działania celem uproszenia pisania programu.
* Funkcjonalność:
  + Rozbudowany wariant wymagań z projektu 2. Program obsługuje UDP knocking z autoryzacją. Obsługa n klientów jednocześnie (ze względu na nie wykorzystanie Exec mogą pojawić się opóźnienia). Dane przesyłane stanowią komunikację z czatem. Timeout oczekiwania na połącznie po autentykacji.
  + Dodatkowa funkcjonalność czat UDP: możliwość wysłania dowolnej praktycznie wiadomości (ograniczenie wynika z maksymalnej zaimplementowanej wielkości buforu pakietu UDP) na konkretne ip-port z dowolnego klienta do dowolnego klienta (aplikacja dopuszcza wysyłanie wiadomości nawet do siebie samego).
  + Usprawnienia: program ma bardziej jednolitą i uporządkowaną strukturę (w odniesieniu do pierwszego projektu na którym bazuje), tam gdzie było to możliwe zaimplementowano HashMap celem przyśpieszenia działania programu (w tej konkretnej wersji zysk jest niewielki). Pakiet ma zdecydowanie bardziej zrozumiałą strukturę dzięki czemu powinno występować zdecydowanie mniej błędów.

Co nie zostało zrealizowane:

Program nie wykorzystuje funkcji Exec celem optymalizacji zarządzania wątkami oraz ich bezpiecznego przerywania. Ograniczona funkcjonalność dopuszcza tylko komunikacje w IPv4. Czat jest bardzo wrażliwy na błędy składni czy znaki białe przy wprowadzaniu poleceń. Timeout kolejki logowania też nie został do końca zaimplementowany (został napisany ale jest wyłączony celem oszczędzania zasobów) jego wartość była niewielka a ilość sprawianych problemów nieproporcjonalna.

Jak zainstalować:

Program jest mobilny t.j. gdziekolwiek się nie znajdzie będzie funkcjonował poprawnie

Plik konfiguracji serwera ServerCFG jest w katalogu głównym projektu. Pierwsza linia pliku zawiera informacje czy serwer ma przyjmować każdą próbę komunikacji czy też ma przeprowadzać sprawdzenie zgodności kluczy. Pozostałe linie zawierają klucze format klucza to: ”port wartość,(…),port wartość”. Każda linia kluczy to osobna niezależnie poprawna sekwencja. Server automatycznie tworzy potrzebne porty (nawet gdy akceptuje każde połącznie bez weryfikacji).

Ewentualne błędy:

Program wykorzystuje JDK 1.9 (ze względu na część FX aplikacji) ponadto w chwili obecnej obsługuję komunikacje tylko przy pomocy IPv4. Błędy jeżeli jakieś się pojawią z zasady jeżeli nie zakończyły działania programu (co zostało przewidziane w wszystkich znalezionych błędach krytycznych) mają charakter: (w konsoli) logu, oraz (w okienku czatu) informacyjny (celem wyjaśnienia np. dlaczego dana komenda się nie wykonała). Przy dużej ilości portów i klientów serwer może mieć problem z podłączeniem kolejnego (bo nie znajdzie portu a nie dlatego że nie jest go w stanie obsłużyć).

Opis protokołu:

Protokół komunikacji opiera się w całości o klasę Packet z następującą struktórą danych:

public static final int *AUTH*=0,*SNDMSG*=1,*SNDFED*=2,*DIS*=3,*CON*=4; //============================================================= CONTENT NEW  
public final int whatDoIcarry;  
public final NetPath sender, reciver;  
//============================================================= NAV  
public final boolean killGateOnExitEnter,addToExecQuene,udp;  
public final int val;// Key/Port  
public final String message;

Gdzie whatDoIcarry stanowi informację o tym co zwiera dany pakiet:

0 = autentykacja => dane logowania do serwera;

1 = wiadomość => wiadomość do danego adresata;

2 = zawartość czatu => otrzymywana przez klienta to udanym zalogowaniu;

3 = wiadomość o odłączeniu się od klienta/serwera;

4 = połączenie => adres portu na którym serwer oczekuje połączenia;

Sender, reciver są odpowiednio adresami nadawcy i odbiorcy wiadomości;

killGateOnExitEnter odpowiada za bezpieczne zamykanie połączenia TCP bez przerwania;

addToExecQuene nie zostało zaimplementowane, miało pozwalać pakietowi wykonywać lokalnie kod na osobnym wątku;

udp informuję klasę Gate o sposobie wysłania wiadomości

val zawiera dowolną chwilową potrzebą wartość

message zawiera informacje w formacie String, w tej implementacji tylko wiadomości od czatu do czatu.

Przykład inicjalizacji połączenia Aplikacji z serwerem (uproszczona wersja połączonych logów klienta i serwera):

{Nadawca: (Klient/Serwer),Odbiorca: ( Klient/ Serwer),(co przenoszę),(udp?),(val),(mess)}

=>{K(0.0.0.1-1024),S(0.0.0.2-2001),(Autentykacja),(TAK UDP),(klucz1),null}

=>{K(0.0.0.1-1024),S(0.0.0.2-2002),(Autentykacja),(TAK UDP),(klucz2),null}

=>{S(0.0.0.2-1025),K(0.0.0.1-1024),(Połączenie),(TAK UDP),(numer portu),null}

=>[K(0.0.0.1-1024),S(0.0.0.2-(numer portu)),połącznie TCP]

=>{S(0.0.0.2-(numer portu)),K(0.0.0.1-(…)),(historia czatu),(NIE UDP),0,”historia czatu”}

=>{ K(0.0.0.1-(…)) ,S(0.0.0.2-(numer portu)),(wiadmość),(NIE UDP),0,”blablabla”}

=>{ K(0.0.0.1-(…)) ,S(0.0.0.2-(numer portu)),(rozłączenie),(NIE UDP),0,null}

Hasło edycji 123