RAPORT PROJEKTU 2

Projekt: SKJ – port knocking v1.0

Autor: Robert Grochowski

Indeks: s17124

OPIS ROZWIĄZANIA

Projekt składa się z dwóch (głównych) klas – Server i Client, które odpowiednio reprezentują serwer oraz klienta.

Klasa Server:

Parametry startowe zostały zaimplementowane zgodnie z wymaganiami dołączonymi w skrypcie t.j. oznaczają poprawną kolejność sekwencji nasłuchiwanych portów UDP. Serwer nasłuchuje na podanych wcześniej portach, następnie umieszcza klienta w mapie „kandydatów" aby każdy wątek mógł umieścić w niej swoje pozwolenie do poprawnej autoryzacji (zmienna *candidates*). Kluczem mapy jest *SocketAddress* (zawierający IP i port klienta) a wartością tablica *Boolean*. Tablica *boolean* jest tablicą pozwoleń. Tzn. każdy port ma swoje id (np. id 0 – pierwszy argument programu) co odpowiada indeksowi w tablicy. Jeśli wartość pod tym indeksem jest true, to znaczy że ten sekwencja w tym porcie się zgadza oraz ten port udzielił swojej zgody. Analogicznie w przypadku wartości *false*. Ponadto, w metodzie *run()* znajduje się nieskończona pętla, która nadzoruje zawartość mapy. Jeśli cała tablica *boolean* ma wartości *true*, to znaczy że klient otrzymuje autoryzację. Poza tym, ustalany jest limit czasu (Timeout) w przypadku gdy klient nie wyśle nic na konkretny port (któryś z portów w ogóle nie dostanie wiadomości). Kiedy czas minie, cały element z mapy jest usuwany, a klient nie dostaje autoryzacji.

Przykład wyjścia:
Powodzenie:

```
Starting the server
Start listening on 3001 UDP port
Start listening on 3000 UDP port
Start listening on 3003 UDP port
Start listening on 3004 UDP port
Start listening on 3002 UDP port
RECEIVED MESSAGE [AUTH_REQ:0] from [/127.0.0.1:56724] on port [3002]
RECEIVED MESSAGE [AUTH_REQ:1] from [/127.0.0.1:56724] on port [3004]
RECEIVED MESSAGE [AUTH_REQ:2] from [/127.0.0.1:56724] on port [3003]
RECEIVED MESSAGE [AUTH_REQ:4] from [/127.0.0.1:56724] on port [3001]
RECEIVED MESSAGE [AUTH_REQ:3] from [/127.0.0.1:56724] on port [3000]
UDP Port [3003] has approved the client's seq?:true
UDP Port [3000] has approved the client's seq?:true
UDP Port [3001] has approved the client's seq?:true
UDP Port [3004] has approved the client's seq?:true
UDP Port [3002] has approved the client's seq?:true
Granting the access for [/127.0.0.1:56724]
Starting TCP socket on port: 55418
```

```
TCP port has been sent by UDP packet
Client [/127.0.0.1] has established TCP connection
Sending authorization key
Client received key successfully
Closing TCP connection
```

Niepowodzenie (w tym przypadku zła sekwencja pukania):

```
Starting the server
Start listening on 3002 UDP port
Start listening on 3000 UDP port
Start listening on 3003 UDP port
Start listening on 3001 UDP port
Start listening on 3004 UDP port
RECEIVED MESSAGE [AUTH_REQ:1] from [/127.0.0.1:52617] on port [3004]
RECEIVED MESSAGE [AUTH_REQ:0] from [/127.0.0.1:52617] on port [3001]
RECEIVED MESSAGE [AUTH_REQ:2] from [/127.0.0.1:52617] on port [3003]
RECEIVED MESSAGE [AUTH_REQ:4] from [/127.0.0.1:52617] on port [3002]
RECEIVED MESSAGE [AUTH_REQ:3] from [/127.0.0.1:52617] on port [3000]
UDP Port [3000] has approved the client's seq?:true
UDP Port [3002] has approved the client's seq?:false
UDP Port [3003] has approved the client's seq?:true
UDP Port [3004] has approved the client's seq?:true
UDP Port [3001] has approved the client's seq?:false
Client [/127.0.0.1:52617] is timeout..
```

Klasa Client:

Parametry startowe zostały zaimplementowanie zgodnie z wymaganiami projektu t.j. pierwszy argument jest adresem IP serwera, a kolejne wyznaczają kolejność „pukania” na porty UDP serwera. Klient po uruchomieniu od razu wysyła pakiety UDP z wiadomością: „AUTH_REQ:X” gdzie X jest numerem sekwencji portu (jest to zwarte w wiadomości, ponieważ uwzględniam to, że pakiety nie muszą przyjść kolejno po sobie).

Upłynięcie limitu czasu (timeout) odpowiedzi od serwera jest ustalany przez wartość stałej TIMEOUT. Po przekroczeniu zadanego czasu i nie uzyskaniu odpowiedzi klient kończy działanie zwracając stosowny komunikat. W przypadku powodzenia, wypisuje uzyskany klucz na konsoli. Przykładowe działanie klienta (podczas gdy serwer jest uruchomiony):

Powodzenie – uzyskanie klucza od serwera:

```
Start knocking
Message [AUTH_REQ:0] has been sent on UDP port [3002]
Message [AUTH_REQ:1] has been sent on UDP port [3004]
Message [AUTH_REQ:2] has been sent on UDP port [3003]
Message [AUTH_REQ:3] has been sent on UDP port [3000]
Message [AUTH_REQ:4] has been sent on UDP port [3001]
Knocking finished - waiting for UDP message
I have received message with port! msg:[PORT:55418]
Trying to establish TCP connection with given port: [55418]
Connection established!
Incoming message: WELCOME
Requesting auth key
Received auth key! :[G3G5EGH26166SHH5525]
Connection closed
```

Niepowodzenie – zła sekwencja lub niezapukanie na odpowiedni port

```
Start knocking
Message [AUTH_REQ:0] has been sent on UDP port [3002]
Message [AUTH_REQ:1] has been sent on UDP port [3004]
Message [AUTH_REQ:2] has been sent on UDP port [3003]
Message [AUTH_REQ:3] has been sent on UDP port [3000]
Message [AUTH_REQ:4] has been sent on UDP port [3005]
Knocking finished - waiting for UDP message
Did not received back message from server!
(probably bad UDP sequence or invalid ports..)
```

Zabrakło czasu na

Jedyne czego nie zdążyłem zrobić to timeout (w bardzo skrajnym przypadku) gdy klient nie połączy się z serwerem przez TCP. Aktualnie posiadam pętle while, która czeka bez końca na połącznie klienta.

Podsumowanie

Projekt był bardzo ciekawy, zainspirował mnie do zagłębienia wiedzy dot. Autoryzacji w sieci. Starałem się gdzie mogłem umieszczać komentarze w mniej jasnych miejscach oraz kładłem nacisk na czytelność kodu aby jak najbardziej ułatwić Panu jego analizę. Ponadto bardzo ściśle trzymałem się wymogów zawartych w skrypcie (w tym zaimplementowanie obsługi wielu klientów na raz) aby projekt był jak najbardziej solidny. Mam nadzieję, że wykonałem wszystko poprawnie.

-