Tftp - opis MiniProjektu

Rafał Byczek

22 stycznia 2017

Zacznę więc od opowiedzenia po krótce o pomocniczych zadaniach, które pojawiły się na satori i o moich rozwiązaniach do nich.

- **G - Klient TFTP (RFC 1350)**
  Moje rozwiązanie zostało zaimplementowane w pliku **clientTFTP.py**. My tutaj podajemy hosta i nazwę pliku do ściągnięcia i w liniach od 40 do 45 następuję wstępne porozumienie z serwerem w celu ściągnięcia pliku. Główne ciało programu jest zawarte w funckji **def start(self)**, tutaj to też pytamy o plik i jak dostajemy pozwolenie to dopóki nie przyjdzie blok krótszy niż 512 bajtów albo coś się samo ze niezepsuje to trzymamy sobie zmienną informującą nas na paczkę o jakim numerze aktualnie oczekujemy, jeżeli taka też przyjdzie, co sprawdza funkcja **def check(self, out)** to jej zawartośc doklejamy do zawartości pobieranego pliku i odsyłamy serwerowi informację, że przyszło co miało przyjść, a jeżeli nas serwer oszuka i wyśle paczkę o innym numerze to my go o tym informujemy, że przyszła zła paczka i prosimy o poprawną jeszcze raz. I tak koniec końców jak plik się pobierze możemy albo poprzez metodę **def getFile(self)** dostać plik albo poprzez **def getCode(self)** wypisać kod md5 naszego pliku, który mieliśmy pobrać. (Mała modyfikacja nastąpiła tylko taka, że na potrzeby testowania Serwera z zadania H zmieniła się forma uruchamiania programu na **python2 clientTFTP.py host plik port**, wiadomo co to co.).

- **H - Serwer TFTP (RFC 1350)**
  Moje rozwiązanie zostało zaimplementowane w pliku **serverTFTP.py**, uruchamianie w postaci **python2 serverTFTP.py port folder**. No i co się tutaj dzieje w tym kodzie, no to w liniach 59-61 tworzy się socket dla servera i na ustalonym porcie zaczyna on nasłuchiwać. No i nasłuchuje czekając na klienta. Jak ktoś napisze na ten port to on to odczytuje i w liniacg 67-73 sprawdza czy przypadkiem to co się pojawiło na porcie nie jest może ramką z zapytaniem RRQ. A jeżeli się okazuje, że jest to sprawdza czy taki plik o jaki poprosił klient znajduje się w folderze i jeżeli tak to zaczyna go klientowi serwować wysyłając najpierw paczkę numer 1 na początek współpracy, a jeżeli takiego pliku nie ma to wysyłany jest klientowi stosowny komunikat. I to się dzieje w liniach 79-80 które to tworzą obiekt klasy tftpSender przekazując plik, klienta i socket i zadaniem klasy tftpSender jest dogadać się z klientem już po potwierdzeniu wysyłania pakietów. W tej klasie nazwy pól dość jasno mówią do czego służa zaś co do metod no to, **def transmit(self)** to poprostu wysyła na dany socket kolejną paczkę danych, która została podrzucona do wysłania. Metode **def ack(self, blockNumber)** sprawdza czy klient potwierdził ten blok, który powinien i jednocześnie ładuje do paczki nową część pliku jednocześnie kontrolując czy plik się jeszcze nie skończył. No i teraz kwestia funkcji **def transmitFile(self)**. Tutaj się dzieje cała reszta, nasłuchujemy co klient do nas napisał, i czy napisał to dobry klient, z którym zaczęliśmy współpracę celem wysłania pliku. Jeżeli wszystko się zgadza i klient wysłał ramkę z ACK to patrzymy czy dobrze potwierdził i jak tak to wysyłamy nową paczkę, a jak nie, to będziemy próbowac wysyłać tę samą aż do póki nie dostaniemy potwierdzenia. I tak aż do póki nie wyślemy całego pliku.

- **I - Klient TFTP (RFC 1350 + RFC 7440)**
  Jest to prawie to samo co zadanie G, z jedną różnicą, że masz też obsługiwać protokoł **RFC 7440**, który poszerza standardowy protokoł TFTP o opcje windowsize. W standardowym protokole wysyłamy w jednej paczce jeden blok danych i oczekujemy odebrania danych przez klienta

i jego potwierdzenia, w tym protokole zaś serwer z klientem dogadują się co do wartości **windowsize**, która to wartość mówi ile bloków serwer wysyła pod rząd nim zacznie nasłuchiwanie potwierdzenia klienta, tzn nie każdy blok danych jest potwierdzany przez klienta, tylko sekwencje kilku bloków wysłanych przez serwer, a klient potwierdza ostatni blok w zestawie, który otrzymał. Dokładniej rzecz biorąc mamy sobie plik **clientTFTP_7440.py**, a w nim np funkcję **def receivePacket(self, last)**, która poprostu pobiera od serwera jedną konkretną paczkę danych. No a funkcja **def start(self)** to tam właśnie startuje całą zabawę. Łaczymy się z serwerem i wysyłamy zmodyfikowaną o windowsize ramkę RRQ, w odpowiedzi pobieramy pierwszy pakiet. I w pętli while wyodrębniamy numer paczki i blok danych, jeżeli jest to blok, na który czekaliśmy to super zapisujemy go i czekamy na następny dopóki nie dostaniemy pod rząd poprawnej ustalonej liczby bloków, wtedy potwierdzamy serwerowi co dostaliśmy ostatnie, a jeżeli podczas tego procesu nastąpią błędy typu przyjdzie paczka nie o tym numerze co miała albo jakieś inne błędy, wysyłamy serwerowi numer ostatniej dobrej paczki, a on nam teraz zacznie wysyłać od pierwszej, której nie otrzymaliśmy. No i kontrolujemy zmienną finish i jak ona wynosi 1 to jest równoważne temu, że serwer przysłał ostatnią paczkę. Uruchomianie w postaci **python2 clientTFTP_7440.py serwer(localhost) nazwa_pliku port windowsize**

- **J - Serwer TFTP (RFC 1350 + RFC 7440)**
  Z serwerem jest podobnie jak z klientem, ale chyba szybciej kod powstał. Generalnie także obsługuje RFC 7440 jak klient z I. Za każdym razem gdy wysyłamy klientowi dane wysyłamy tyle paczek ile wynosi wartośc **windowsize** poczynając od pierwszej paczki, której klient nam nie potwierdził. Uruchamianie w postaci podobnej ja poprzednio **python2 serverTFTP_7440.py port sciezka_do_folderu**.