Zadanie 1 – SIP Proxy (telefónna ústredňa)

Filip Vida

Použité knižnice

```
import sipfullproxy as proxy
import socketserver
import netifaces
```

sipfullproxy

Pre implementáciu tohto zadania sme využili sip knižnicu získanú z https://github.com/tirfil/PySipFullProxy .Z uvedeného zdroja bol využitý program sipfullproxy.py, ktorý sme si importovali ako knižnicu. Knižnica bola následne upravená pre fungovanie s Python 3, bol z nej vymazaní existujúci main a boli v nej vykonané drobné zmeny na implementáciu doplnkových funkcionalít.

Socketserver

Knižnicu socketserver využívame na spustenie UDP servera, ktorému poskytneme serverovú adresu vo forme ip adresy a portu, a funckiu ktorá slúži ako handler pre prijatú komunikáciu.

Netifaces

Knižnicu netifaces využívame pre získavanie ip adresy z vybratého sieťového rozhrania. Využívame ju na jednoduché, automatické vybratie ip adresy na ktorej bude bežať server.

Úpravy knižnice

V použitej knižnice bolo urobených niekoľko zmien s účeľmi:

Sfunkčnenie knižnice

Využitú knižnicu bolo v prvom rade nutné sfunkčniť pre Python 3. Vykonané zmeny boli napríklad:

Zmena mena knižnice SocketServer

```
import SocketServer
```

```
import socketserver as SocketServer
```

Zmena bytes like objectu na string. Bolo využité kódovanie windows-1252 namiesto utf-8, kvôli chybe pri dekódovaní 'utf-8' codec can't decode byte 0x80.

```
request_uri = self.data[0]
```

```
data = self.request[0].decode('windows-1252')
```

Stringové dáta bolo treba enkódovať pred ich poslaním.

```
self.socket.sendto(text,self.client_address)
```

```
self.socket.sendto(text.encode(),self.client_address)
```

IDE v ktorom sme pracovali tiež hlásilo varovania, že štýl akým knižnica kontroluje či sa v dict nachádza určitý kľúč je v Python 3 nepodporovaná.

```
if registrar.has_key(destination):
```

```
if destination in registrar:
```

Úprava knižnice kvôli doplnkovým úlohám

2 doplnkové úlohy vyžadovali úpravu či doplnenie kódu knižnice, a to Logovanie denníka hovorov a úprava sip stavových kódov. Pri úprave SIP stavových kódov boli originálne, anglické stavové kódy nahradené ich slovenským prekladom, napríklad 480 Temporarily Unavailable bola zmenená na 480 Dočasne nedostupné.

```
self.sendResponse("480 Temporarily Unavailable")
```

```
self.sendResponse("480 Dočasne nedostupné")
```

Pre dosiahnutie funkcionality denníka hovorov boli všetky originálne logovania prepísané z logging.info na logging.debug. Jedine nami implementované logovanie sa nachádza na úrovni info, originálne logovanie je zachované, ale bolo zmenené na úroveň debug.

```
logging.info("destination %s" % destination)
```

```
logging.debug("destination %s" % destination)
```

Následne boli pridané nové regexy, ktoré nám pomohly s identifikovaním žiadanej komunikácie, ktorú chceme zaznamenať.

```
#Regex for logging book
rx_decline = re.compile('Decline')
rx_ok = re.compile('Ok')
```

```
#For logging book
if rx_decline.search(request_uri):
    logging.info(str(datetime.now()) + " " + self.getDestination() + " is hanging up before the call starts")
elif rx_ok.search(request_uri):
    logging.info(str(datetime.now()) + " " + self.getDestination() + " is accepting the call")
```

Niektoré si však takéto pridanie novej logiky nežiadali, boli len pridané do originálneho kódu.

```
logging.info(str(datetime.now()) + " " +origin + " is calling " + destination)
```

Vytvorenie vlastného mainu pre "owrapovanie" knižnice

Ako už bolo spomenuté využitá knižnica je importovaná do mainu programu, kde sú následne v maine inicializované všetky potrebné premenné pre funkčnosť proxy, a nasledovne je spustený UDP server. V maine využijeme už spomínané knižnice na získanie ip adresy z wifi adaptéra, inicializujeme potrebné premenné pre správne fungovanie knižnice(recordroute a topvia, ktoré pridávajú do sip komunikácie polia s ich menom), zapneme logovanie pre hovory a nakoniec zapneme server.

```
if __name__ == "__main__":
    #print(netifaces.interfaces())
    #We get the ip of the wireless interface, different pc's need to use different interface name
    ipaddress = netifaces.ifaddresses('{1F207080-FFAA-4C43-9106-B53397C0F3F7}')[netifaces.AF_INET][0]['addr']
    proxy.HOST = ipaddress
    proxy.logging.basicConfig(filename='phonebook.txt',level=proxy.logging.INFO)
    proxy.recordroute = "Record-Route: <sip:%s:%d;lr>" % (ipaddress,proxy.PORT)
    proxy.topvia = "Via: SIP/2.0/UDP %s:%d" % (ipaddress,proxy.PORT)
    server = socketserver.UDPServer((proxy.HOST, proxy.PORT), proxy.UDPHandler)
    print("The SIP proxy is running on port: " + str(proxy.PORT))
    print("The proxy is running on ip address: "+ ipaddress)
    server.serve_forever()
```

PCAP súbory

Na github sa tiež nachádzajú pcap traces z odskúšaných scenárov. Odchytené scenára sú v názve pcap-ov. Keďže komunikácia pobiehala medzi 2 klientami(ani jeden z klientov nebol proxy), vo väčšine súborov sa nenachádza rtp komunikácia medzi zariadeniami, len v jednom v ktorom je naschvál jeden klient zároveň aj proxy. Pre vyfiltrovanie relevantnej komunikácie je nutné použiť filter sip || rtp a komunikácie vieme pekne vidieť v Telephony -> Voip Calls.

Git repozitár

https://github.com/xvidaf/Vida_MTAA_1