Memoria Proxy Web - ProxPy

David Carrascal Acebron

Abril 2019

Índice general

Capítulo 1

Proxy Web

1.1. Proxy (proxy.py) - ProxPy source code

```python
#/usr/bin/env python3
import socket
import sys
import os
import select
import time
import datetime
import signal
import pickle
import argparse
from random import *

#Note:
#
#   ProxPy(v1.1) is a web proxy with capabilities to address HTTP request 1.1.
#   It has a layer of filtering controlled with its powerful CLI.
#
#   The realization of this project has been carried out in the subject of
#   Laboratory of networks, systems and services at the University of Alcalá
#   by David Carrascal.
#
#   Have fun :)
#
#   For more info: github.com/davidcawork

#Global vars
VERSION_MAJOR_NUMBER = 1
VERSION_MINOR_NUMBER = 1
DEBUG_LEVEL_MAX = 3
DEBUG_LEVEL_NORMAL = 2
DEBUG_LEVEL_LOW = 1
MAX_MSG_SAVED = 20
MSG_PROXPY_INACTIVE = '[ProxPy] ProxPy inactive ...'
MSG_PROXPY_HI = '[ProxPy] Welcome to ProxPy CLI'
MSG_PROXPY_BYE = '[ProxPy] Turning off ProxPy ....'
MSG_PROXPY_VERSION = '[ProxPy] Current version is: ProxPy v'+ str(VERSION_MAJOR_NUMBER) + '.' + str(VERSION_MIN
MSG_PROXPY_NEW_INPUT_CONN = '[ProxPy] New input connection from: '
MSG_PROXPY_BLCK = '[ProxPy] Blocking request to: '
MSG_PROXPY_BLCK_CONN = '[ProxPy] Blocking connection to: '
MSG_PROXPY_LOG_DATA = '[ProxPy] Log data'
MSG_PROXPY_LOG_BYE = '[ProxPy] Bye!'
MSG_PROXPY_LOG_REQ = '[ProxPy] Log data: Request'
```

```python
MSG_PROXPY_LOG_RPLY = '[ProxPy] Log data: Reply'
ERROR_BAD_ARGVS_FROM_USER = '[ProxPy] Error, incorrect arguments: '
ERROR_TO_RCV_FROM_NAV = '[ProxPy] Error, cannot recover the request from: '
ERROR_TO_RCV_FROM_SW =  '[ProxPy] Error, cannot recover the server reply from: '
ERROR_TO_SEND_REQUEST = '[ProxPy] Error, cannot send the request to the server, connecting again...'
ERROR_TO_CONN_WITH_SW = '[ProxPy] Error, cannot connect with Server: '
ERROR_TO_CLOSE_INPUT_CONN = '[ProxPy] Error, cannot close the input connections: '
ERROR_TO_CLOSE_OUTPUT_CONN = '[ProxPy] Error, cannot close the output connections: '
ERROR_TO_CLOSE_CONN = '[ProxPy] Error, cannot close the connections: '
ERROR_TO_BIND_OUR_PORT = '[ProxPy] Error, our listening port is already in use, instead we use port: '
ERROR_TO_PREPARE_REQUEST= '[ProxPy] Error, cannot prepare the request: '
ERROR_TO_REPLY_NAV = '[ProxPy] Error, cannot process the request: '
ERROR_TO_LOG_REQUEST = '[ProxPy] Error, cannot log the request'
ERROR_TO_LOG_REPLY = '[ProxPy] Error, cannot log the reply'

#MACROS (str)
CRLF = '\r\n'  #Carriage return AND line feed
WSP = ' '
NSP = ''
COLON = ':'

#MACROS (byte)
CRLF_B = b'\r\n'  #Carriage return AND line feed
WSP_B = b' '
NSP_B = b''
COLON_B = b':'

#To get our socket TCP, where we will hear connections from web navigators
def get_our_socket(port,msg_history):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setblocking(0)
        s.bind(('',port))
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.listen(5)
        return s
    except:
        new_port = randint(8000, 9000)
        #print(get_str_time_ProxPy()+ERROR_TO_BIND_OUR_PORT+ str(new_port))
        add_to_msgHistory(msg_history,get_str_time_ProxPy()+ERROR_TO_BIND_OUR_PORT+ str(new_port)+ '\n')
        print_msgs(msg_history)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setblocking(0)
        s.bind(('',new_port))
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.listen(5)
        return s

#Aux func to know if one item is in one list
def is_in_the_list(list_, element):

    if list_.count(element):
        return True

    return False

#To get ProxPy str time format
def get_str_time_ProxPy():
    return ('['+(datetime.datetime.now()).strftime('%H:%M:%S')+'] ')

```

```python
#To parse all incoming HTTP request
def http_request_parser(data):

    #Request dic
    request = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-'}

    http_request_parser_line(request, data.split(CRLF)[0])
    http_request_parser_headers(request,data.split(CRLF)[1:])
    http_request_parser_body(request, data.split(CRLF)[request['header_count'] + 1 :])

    return request

#To handle bad argvs
def bad_argvs_handler():
    print( get_str_time_ProxPy() + ERROR_BAD_ARGVS_FROM_USER +  '\n\n\t Usage: python3 ' +
                sys.argv[0] + ' <Port> .... \n\n\n For more help you can chek: python3 '+ sys.argv[0] + ' -h\n'

#To parse all incoming HTTP request(Just first line)
def http_request_parser_line(request, data):

    request['method'] = data.split(WSP)[0]
    request['uri'] = data.split(WSP)[1]
    request['version'] = data.split(WSP)[2]

#To parse all incoming HTTP request(headers)
def http_request_parser_headers(request,data):

    for items in data:
        if items == NSP:
            break
        else:
            request['headers_list'].append([items.split(COLON)[0], (items.split(COLON + WSP)[1]).strip()])
            request['header_count'] += 1

#To parse all incoming HTTP request(To get the body)
def http_request_parser_body(request, data):

    if data[0] is NSP:
        request['body'] = '-'
    else:
        request['body'] = data[0]


#To parse all incoming HTTP request (bin)
def http_request_parser_bin(data):

    #Request dic
    request = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-'}

    list_str_data= str(data).split('\\r\\n')
    list_str_data.remove("'")

    http_request_parser_line(request, list_str_data[0][2:])
    http_request_parser_headers(request,list_str_data[1:])
    http_request_parser_body(request, list_str_data[request['header_count'] + 1 :])

    return request


#To get host from request
```

```python
def get_host_from_header_list(list_):

    for item in list_:
        if item[0] == 'Host':
            return item[1]

#To read
def read_port_url(request):

    str_uri = request['uri']

    if len(str_uri.split(COLON)) == 2:
        return 80
    else:
        return int(str_uri.split(COLON)[2])


#To get a conn with WS
def get_conn_to_server(output_conn_request_reply, request):


    port = read_port_url(request)

    sock_aux = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock_aux.connect((socket.gethostbyname(get_host_from_header_list(request['headers_list'])),port))
    except:
        #print( get_str_time_ProxPy() + ERROR_TO_CONN_WITH_SW + get_host_from_header_list(request['headers_list
        add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CONN_WITH_SW + get_host_from_header_list
        print_msgs(msg_history)


    output_conn_request_reply.append([sock_aux,socket.gethostbyname(get_host_from_header_list(request['headers_

    return sock_aux

#To update socket descriptor
def update_socket_output_conn(output_conn_request_reply,ip,sock):

    for item in output_conn_request_reply:
        if item[1] == ip:
            item[0] = sock
            break


#Returns True if we have a conn | False if we havent
def is_already_conn_sw(output_conn_request_reply, ip_addr, sock_to_rcv = 'default'):

    for conns in output_conn_request_reply:
        if conns[1] == ip_addr and conns[0] == sock_to_rcv:
            return True

    return False

#To append a request to active conn | return socket descriptor
def append_request(conn_request,ip_addr, request_to_append ):

    for conns in conn_request:
        if conns[1] == ip_addr:
            conns[2].append(request_to_append)
            return conns[0]

#To append a request to input_conn_request_reply list
def add_to_input_conn_request(input_conn_request_reply, sock_to_rcv, request):

    if not is_already_conn_sw(input_conn_request_reply, sock_to_rcv.getsockname()[0], sock_to_rcv):
        input_conn_request_reply.append([sock_to_rcv, sock_to_rcv.getsockname()[0],[request],[]])
    else:
        append_request(input_conn_request_reply, sock_to_rcv.getsockname()[0],request)

#To send a request to server web
def send_request_to_sw(host_uri, request, output_conn_request_reply):
    #Var aux
    pet = b''

    #We have to re-create the request
    #Add first line
    pet += WSP_B.join([(request['method']).encode(),(request['uri']).encode(),(request['version']).encode()]) +

    #Add headers (Here we can add 'if' to change some header )
    for item in request['headers_list']:
        if item[0] == 'Connection':
            pet += item[0].encode() + b': '+ b'Close' + CRLF_B
        elif item[0] == 'Upgrade-Insecure-Requests':
            pass
        else:
            pet += item[0].encode() + b': '+item[1].encode() + CRLF_B

    #Fin headers
    pet += CRLF_B

    #Add Body
    if request['body'] is not '-':
        pet += (request['body']).encode()

    #Fin request
    pet += CRLF_B


    #Send it
    try:
        host_uri.sendall(pet)
    except:
        #print(get_str_time_ProxPy()+ ERROR_TO_SEND_REQUEST)
        add_to_msgHistory(msg_history,get_str_time_ProxPy()+ ERROR_TO_SEND_REQUEST)
        print_msgs(msg_history)
        host_uri = get_conn_to_server(output_conn_request_reply, request)
        host_uri.sendall(pet)


#To get the socket where we will send the reply
def get_input_socket_from_request(list_intput, request):

    #HTTP replys in order
    for item in input_conn_request_reply:
        if item[2][0] == request:
            return item[0]

#To get the socket where we will send the request
def get_output_socket_from_request(list_out, ip):
```

```python

    for item in list_out:
        if item[1] == ip:
            return item[0]

#To print msg_history
def print_msgs(msg_history):
    sys.stdout.flush()
    os.system('clear')
    for msg in msg_history:
        print(msg)
#To manege msg's
def add_to_msgHistory(msg_history,msg):

    if len(msg_history) == MAX_MSG_SAVED:
        msg_history.pop(0)

    msg_history.append(msg)

#to get cmd written
def is_command(msg, str_cmd):
    return msg.count(str_cmd)

#To print /help cmd
def print_help():
    sys.stdout.flush()
    os.system('clear')
    print('Hi User !\n\n')
    print('These are the commands that you can use:\n')
    print('\t/help\t\tTo consult the commands and guides for using the ProxPy CLI')
    print('\t/quit\t\tTo exit, it close all connections')
    print('\t/timeup\t\tTo get the time you have connected in ProxPy CLI')
    print('\t/stats\t\tTo get statistics about the activity of ProxPy and attributes of it')
    print('\t/reload\t\tTo reload all connections')
    print('\t/showfilter\tTo show the current filter rules')
    print('\t/filter_server\tTo add an URL to permit in filter rules')
    print('\t/filter_client\tTo add an User/s to permit in filter rules [Netmask avaible /0 /8 /16 /24 /32]')
    print('\t/showfilter\tTo show the current filter rules')
    print('\t/debug [id]\tTo set debug level')
    print('\t/max_conn [num]\tTo set max connection number')
    print('\t/timeout [sec]\tTo set the activity timer (seconds)')

    print('\n\nFor more help you can check: https://github.com/davidcawork\n\n')

    input("Press Enter to continue...")
    os.system('clear')

#To get time up in ProxPy
def timeup_cmd(time_init):
    os.system('clear')
    print('Hi User !\n\n')
    time_b = datetime.datetime.now()
    print('You have '+str(time_b - time_init)+' time in ProxPy n.n\n\n\n')
    input("Press Enter to continue...")
    os.system('clear')

#To get the request associate with a socket
def get_request_from_output_conn(output_conn_request_reply, sock_to_rcv):

    #HTTP replys in order
    for item in output_conn_request_reply:
        if item[0] == sock_to_rcv:
            return item[2][0]

#To remove a conn from list
def remove_conn(list_to_rm, socket_to_rm):

    try:
        for item in list_to_rm:
            if item[0] == socket_to_rm:
                list_to_rm.remove(item)
    except:
        add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN +' Value conn list '+str(list
        print_msgs(msg_history)
        #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN +' Value conn list '+str(list_to_rm))


# To close al connections (Web navigators and SW)
def close_all_conn(sockets_rd, input_conn, output_conn):

    try:
        for sck_in in input_conn:
            sck_in.close()
        input_conn.clear()
    except:
        #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_INPUT_CONN +' Value input conn list '+str(input_conn))
        add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CLOSE_INPUT_CONN +' Value input conn lis
        print_msgs(msg_history)
    try:
        for sck_out in output_conn:
            sck_out.close()
        output_conn.clear()
    except:
        #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_OUTPUT_CONN +' Value output conn list '+str(output_conn
        add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CLOSE_OUTPUT_CONN +' Value output conn l
        print_msgs(msg_history)
    try:
        for sck in sockets_rd:
            if sck != sys.stdin:
                sck.close()
        sockets_rd.clear()
        sockets_rd.append(sys.stdin)

    except:
        #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN +' Value conn list '+str(sockets_rd))
        add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN +' Value conn list '+str(sock
        print_msgs(msg_history)

#Welcome msg
def welcome(msg_history):
    #print(get_str_time_ProxPy() + MSG_PROXPY_HI +'\n' +get_str_time_ProxPy()+ MSG_PROXPY_VERSION)
    add_to_msgHistory(msg_history,get_str_time_ProxPy() + MSG_PROXPY_HI +'\n' +get_str_time_ProxPy()+ MSG_PROXP
    print_msgs(msg_history)

#Handler CTRL+C
def signal_handler(sig, frame):
    print( '\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
    #Close al connections (Web navigators and SW) and exit
    close_all_conn(sockets_rd, input_conn, output_conn)
    sys.exit(0)
```

```

#To get UDP sockets desc.
def get_logger_socket():
    return socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#To send to our logger  request info
def send_to_logger_request(logger, logger_id, ip_client, ip_dest_, port_client,request, msg_history):
    try:
        # Our pkt : [DATA, REQ/RPLY, [method, version, server(url), server(ip), client(ip), client(port)]]
        logger.sendto(pickle.dumps([MSG_PROXPY_LOG_DATA, MSG_PROXPY_LOG_REQ,[request['method'], request['versio
    except:
        #print( get_str_time_ProxPy() + ERROR_TO_LOG_REQUEST )
        add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_LOG_REQUEST )
        print_msgs(msg_history)

#To send to our logger  reply info
def send_to_logger_reply(logger, logger_id, ip_client, ip_dest_, port_client,request, request_2,msg_history):
    try:
        # Our pkt : [DATA, REQ/RPLY, [method, version, server(url), server(ip), client(ip), client(port)]]
        logger.sendto(pickle.dumps([MSG_PROXPY_LOG_DATA, MSG_PROXPY_LOG_RPLY,[request['method'], request['uri']
    except:
        #print( get_str_time_ProxPy() + ERROR_TO_LOG_REPLY )
        add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_LOG_REPLY )
        print_msgs(msg_history)

#Init argparse
def init_argvs():

    parser = argparse.ArgumentParser(description="Welcome to ProxPy's help page", epilog='For more help you can
    parser.add_argument('-p','--port',metavar='Port',type= int,default=8080,help='Provide an integer that will
    parser.add_argument('-d','--debug',metavar='Debug',type= int,default=3,help='Provide an integer that will b
    parser.add_argument('-t','--timeout',metavar='Timeout',type= int,default=300,help='Provide an integer that
    parser.add_argument('-b','--buffer',type= int,default=1024*1000,help='Provide an integer that will be our b
    parser.add_argument('-c','--max_conn',type= int,default=8,help='Provide an integer that will be max client
    parser.add_argument('-fs','--filter_server',type= str,default="",help='Provide an [url] to restrict access
    parser.add_argument('-fc','--filter_client',type= str,default="",help='Provide an IP range that will be per

    return parser

#To prepare and parse all argvs
def prepare_argvs(parser,proxy_port,proxy_timeout,debug_mode,max_client_conn,BUFFER_SIZE,list_filter_server,lis

    my_args = parser.parse_args()

    proxy_port= my_args.port
    proxy_timeout= float(my_args.timeout)
    debug_mode=my_args.debug
    BUFFER_SIZE=my_args.buffer
    max_client_conn=my_args.max_conn

    if my_args.filter_server != '':
        list_filter_server.append(my_args.filter_server)

    if my_args.filter_client != '':
        list_filter_client.append(my_args.filter_client)

    return len(vars(my_args))

def http_reply_parser_bin(reply_container):

```

```python
    #Request dic
    reply = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-'}

    list_str_data= str(reply_container).split('\\r\\n')
    #list_str_data.remove("'")

    http_request_parser_line(reply, list_str_data[0][2:])

    return reply

#To filter via SW and ip range
def should_process_request(request,client_ip, list_filter_server, list_filter_client):
    permit = False

    if len(list_filter_server) == 0 and len(list_filter_client) == 0:
        #No filters :)
        return True

    elif len(list_filter_server) != 0 and len(list_filter_client) == 0:
        #In case there is a filter by allowed servers
        for item in list_filter_server:
            if item == get_host_from_header_list(request['headers_list']):
                permit = True
        return permit

    elif len(list_filter_server) == 0 and len(list_filter_client) != 0:
        #In case there is a filter by ip range
        same_net= 0
        mask = int(int(list_filter_client[0].split('/')[1])/8)

        if mask == 0:
            return True
        else:
            mask_numbers = (list_filter_client[0].split('/')[0]).split('.')
            mask_numbers_int =  []

            for number in mask_numbers:
                mask_numbers_int.append(int(number))

            client_list_str = client_ip.split('.')
            client_ip_int = []
            for number in client_list_str:
                client_ip_int.append(int(number))

            for i in range(0,mask):
                if client_ip_int[i] == mask_numbers_int[i]:
                    same_net += 1

        if same_net == mask:
            return True
        else:
            return False

    else:
        #In case there is a filter by ip range and allowed servers

        same_net= 0
        mask = int(int(list_filter_client[0].split('/')[1])/8)

        if mask == 0:
            permit = True
        else:
            mask_numbers = (list_filter_client[0].split('/')[0]).split('.')
            mask_numbers_int =  []

            for number in mask_numbers:
                mask_numbers_int.append(int(number))

            client_list_str = client_ip.split('.')
            client_ip_int = []
            for number in client_list_str:
                client_ip_int.append(int(number))

            for i in range(0,mask):
                if client_ip_int[i] == mask_numbers_int[i]:
                    same_net += 1

        if same_net == mask:
            permit = True
        else:
            return False

        for item in list_filter_server:
            if item == get_host_from_header_list(request['headers_list']):
                permit = True

        return permit
#CLI utils
def getServer_Ulr(msg):

    try:
        return msg.split(' ')[1]
    except:
        return ''

def getInt_msg(msg):

    try:
        return int(msg.split(' ')[1])
    except:
        return 15

#To get our stats :)
def stats_cmd(cmd_used,time_init,n_reply,n_request,debug_mode,max_client_conn,BUFFER_SIZE,proxy_timeout,len_msg

    os.system('clear')
    print('Hi User !\n\n')
    time_b = datetime.datetime.now()
    print('ProxPy stats:\n')
    print('1.\t Request sent : '+str(n_request))
    print('2.\t Reply rcv    : '+str(n_reply))
    print('3.\t Debug level  : '+str(debug_mode))
    print('4.\t Max conn     : '+str(max_client_conn))
    print('5.\t Buffer size  : '+str(BUFFER_SIZE))
    print('6.\t Time out     : '+str(proxy_timeout))
    print('7.\t Commands used: '+str(cmd_used))
    print('8.\t msg_History  : '+str(len_msg_history))
    print('9.\t Time up      : '+str(time_b -time_init)+'\n\n')

    input("Press Enter to continue...")
    os.system('clear')

def print_filter_table(filter_client, filter_server):
    os.system('clear')
    print('Hi User !\n\n')

    print('\t\t-- Permit Server table --\n')
    for peer in filter_server:
        print('+ \t\tName: '+peer)

    print('\n\n\t\t-- Permit Clients table --\n')
    for peer in filter_client:
        print('+ \t\tIP_range: '+peer)

    print('\n\n')
    input("Press Enter to continue...")
    os.system('clear')

#Main
if __name__ == "__main__":


    # --- Vars ----
    msg_history = []
    parser = init_argvs()
    proxy_port = 8080
    proxy_timeout = 300.0
    list_filter_server = []
    list_filter_client = []
    max_client_conn = 8
    curr_conn = 0
    debug_mode = 1
    BUFFER_SIZE = 1024*1000
    reply_container= b''
    logger_id = ['localhost', 8010]


    #To parse argvs
    len_argvs = prepare_argvs(parser,proxy_port,proxy_timeout,debug_mode,max_client_conn,BUFFER_SIZE,list_filte
    my_args = parser.parse_args()
    proxy_port= my_args.port
    proxy_timeout= float(my_args.timeout)
    debug_mode=my_args.debug
    BUFFER_SIZE=my_args.buffer
    max_client_conn=my_args.max_conn

    #Check argv's and init args parser
    if len_argvs < 2:
            bad_argvs_handler()
            exit(0)

    else:

        # --- Say welcome to ProxPy and print current version ---
        welcome(msg_history)

        #Let's to prepare the CTRL + C signal to handle it and be able  to show the statistics before it comes
        signal.signal(signal.SIGINT, signal_handler)

            # --- Prepare our TCP socket where we will hear connections from web navigators ---
```

```python
        our_proxy_socket = get_our_socket(proxy_port, msg_history)

        # --- Prepare our udp socket where w'ill log every single pet ---
        logger = get_logger_socket()

        #To save all msg and stats
        time_init = datetime.datetime.now()
        cmd_used = 0
        n_request = 0
        n_reply = 0

        #Sockets to read
        sockets_rd = [sys.stdin, our_proxy_socket]

        #To save
        #Input connections
        input_conn = []
        input_conn_request_reply = []

        #Output connections
        output_conn = []
        output_conn_request_reply = []

        #We'ill store the request like this : [ [sockets_descriptor, str_host, [current_request_1, current_requ

        # We can exit by CTRL+C signal :)
        while True:
            try:
                            # The optional timeout argument specifies a time-out as a floating point number in
                events_rd,events_wr,events_excp = select.select( sockets_rd,[],[], proxy_timeout)

            except KeyboardInterrupt:
                add_to_msgHistory(msg_history,'\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE + '\n')
                print_msgs(msg_history)
                #print( '\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
                #Close al connections (Web navigators and SW) and exit
                close_all_conn(sockets_rd, input_conn, output_conn)
                sys.exit(0)

            for event in events_rd:

                if event == our_proxy_socket:

                    if curr_conn <= max_client_conn:
                        #Accept input conn from web navigator
                        conn, addr = our_proxy_socket.accept()
                        conn.setblocking(0)
                        sockets_rd.append(conn)
                        input_conn.append(conn)
                        curr_conn+=1
                    else:
                        conn,addr =  our_proxy_socket.accept()
                        conn.close()
                        if debug_mode >= DEBUG_LEVEL_NORMAL:
                            add_to_msgHistory(msg_history,get_str_time_ProxPy() + MSG_PROXPY_BLCK_CONN + addr[
                            print_msgs(msg_history)
                            #print(get_str_time_ProxPy() + MSG_PROXPY_BLCK_CONN + addr[0] +':'+ str(addr[1]))

                #CLI ProxPy v1.1
                # Stdin event
                elif event is sys.stdin:
                    msg = input()

                    if is_command(msg,'/quit'):
                        #To shutdown ProxPy
                        os.system('clear')
                        close_all_conn(sockets_rd, input_conn, output_conn)
                        add_to_msgHistory(msg_history,'\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE + '\n')
                        print_msgs(msg_history)
                        sys.exit(0)

                    elif is_command(msg,'/help'):
                        #To print help msg
                        cmd_used +=1
                        print_help()
                        print_msgs(msg_history)

                    elif is_command(msg,'/filter_client'):
                        #To permit some client
                        cmd_used +=1
                        client_to_permit = getServer_Ulr(msg)
                        list_filter_client.append(client_to_permit)
                        print_msgs(msg_history)

                    elif is_command(msg,'/filter_server'):
                        #To permit some url
                        cmd_used +=1
                        server_to_permit = getServer_Ulr(msg)
                        list_filter_server.append(server_to_permit)
                        print_msgs(msg_history)

                    elif is_command(msg,'/debug'):

                        cmd_used +=1
                        debug_mode = getInt_msg(msg)
                        print_msgs(msg_history)

                    elif is_command(msg,'/max_conn'):

                        cmd_used +=1
                        max_client_conn = getInt_msg(msg)
                        print_msgs(msg_history)

                    elif is_command(msg,'/timeout'):

                        cmd_used +=1
                        proxy_timeout = float(getInt_msg(msg))
                        print_msgs(msg_history)

                    elif is_command(msg,'/showfilter'):

                        cmd_used +=1
                        print_filter_table(list_filter_client, list_filter_server)
                        print_msgs(msg_history)

                    elif is_command(msg,'/timeup'):
                        #To print time up
                        cmd_used +=1
                        timeup_cmd(time_init)
                        print_msgs(msg_history)

                    elif is_command(msg,'/stats'):
                        #To print our stats
                        cmd_used +=1
                        stats_cmd(cmd_used,time_init,n_reply,n_request,debug_mode,max_client_conn,BUFFER_SIZE,p
                        print_msgs(msg_history)
                    else:
                        #To support non cmd data
                        now = datetime.datetime.now()
                        add_to_msgHistory(msg_history,'['+now.strftime('%H:%M:%S')+'] ProxPy(CLI): ~/$  '+msg)
                        print_msgs(msg_history)


                else:
                        #Handle other conn
                    for sock_to_rcv in sockets_rd:
                        #To manage request from web nav. connections
                        if sock_to_rcv != our_proxy_socket and sock_to_rcv is event and is_in_the_list(input_co

                            #Recover request from Web nav
                            try:
                                data = sock_to_rcv.recv(BUFFER_SIZE)
                            except:
                                #print( get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + sock_to_rcv.getsocknam
                                add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + s
                                print_msgs(msg_history)
                                continue

                            if data:
                                #Parse the request
                                try:
                                    if debug_mode > DEBUG_LEVEL_NORMAL:
                                        print("{}".format(data.decode('utf-8')))
                                    #request = http_request_parser(data.decode('utf-8'))
                                    request = http_request_parser_bin(data)

                                except:
                                    if debug_mode >= DEBUG_LEVEL_NORMAL:
                                        #print( get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + ' \n\n'+str(da
                                        add_to_msgHistory(msg_history,'\n\n' + get_str_time_ProxPy() + ERROR_TO
                                        print_msgs(msg_history)
                                    curr_conn -= 1
                                    sock_to_rcv.close()
                                    sockets_rd.remove(sock_to_rcv)
                                    input_conn.remove(sock_to_rcv)
                                    remove_conn(input_conn_request_reply, sock_to_rcv)
                                    continue

                                #Process the request

                                #Filter
                                if request['method'] == 'GET' and get_host_from_header_list(request['headers_li
                                    if should_process_request(request,sock_to_rcv.getsockname()[0], list_filter
                                        try:
                                            #Open connection to get uri
                                            host_uri = get_conn_to_server(output_conn_request_reply, request)

                                            #Add to input_conn_request_reply the request
                                            add_to_input_conn_request(input_conn_request_reply, sock_to_rcv, re

                                            #Add to sockets_rd only if its necessary
                                            if not is_in_the_list(sockets_rd, host_uri):
                                                sockets_rd.append(host_uri)

                                            #Add to output_conn only if its necessary
                                            if not is_in_the_list(output_conn, host_uri):
                                                output_conn.append(host_uri)

                                            #Send the request and add to the list output_conn_request_reply, an
                                            send_to_logger_request(logger, logger_id, sock_to_rcv.getsockname()
                                            send_request_to_sw(host_uri, request, output_conn_request_reply)
                                            n_request += 1
                                        except:
                                            if debug_mode >= DEBUG_LEVEL_NORMAL:
                                                #print(get_str_time_ProxPy() + ERROR_TO_PREPARE_REQUEST + reque
                                                add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_
                                                print_msgs(msg_history)
                                            continue

                                        #Main handler request
                                        while True:
                                            try:
                                                #If host_uri sockets is closed, we try reconnect with the web s
                                                #Then we recv the reply to our request
                                                if host_uri._closed:
                                                    host_uri = get_conn_to_server(output_conn_request_reply, re
                                                    data_rpl = host_uri.recv(BUFFER_SIZE)
                                                else:
                                                    data_rpl = host_uri.recv(BUFFER_SIZE)
                                                    reply_container += data_rpl
                                                if data_rpl:
                                                    #If data is not b' ' we send it back to web navigator
                                                    sock_to_rcv.send(data_rpl)
                                                else:
                                                    #When we have sent it the reply close the host_uri socket a
                                                    try:
                                                        reply = http_reply_parser_bin(reply_container)
                                                        send_to_logger_reply(logger, logger_id,logger_id[0],soc
                                                    except:
                                                        pass
                                                    n_reply += 1
                                                    host_uri.close()
                                                    sockets_rd.remove(host_uri)
                                                    output_conn.remove(host_uri)
                                                    remove_conn(output_conn_request_reply, host_uri)
                                                    reply_container = b''
                                                    break
                                            except:
                                                if debug_mode >= DEBUG_LEVEL_NORMAL:
                                                    #print(get_str_time_ProxPy() + ERROR_TO_REPLY_NAV + request
                                                    add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR
                                                    print_msgs(msg_history)
                                                break

                                    else:
                                        if debug_mode >= DEBUG_LEVEL_NORMAL:
                                            #print(get_str_time_ProxPy() + MSG_PROXPY_BLCK + get_host_from_head
                                            add_to_msgHistory(msg_history,get_str_time_ProxPy() + MSG_PROXPY_BL
                                            print_msgs(msg_history)
                                        curr_conn -= 1
                                        sock_to_rcv.close()
                                        sockets_rd.remove(sock_to_rcv)
                                        input_conn.remove(sock_to_rcv)
                                        remove_conn(input_conn_request_reply, sock_to_rcv)

                                else:
                                    curr_conn -= 1
                                    sock_to_rcv.close()
                                    sockets_rd.remove(sock_to_rcv)
                                    input_conn.remove(sock_to_rcv)
                                    remove_conn(input_conn_request_reply, sock_to_rcv)

                            else:
                                #Only when we have rcv b' ' from web navigator, close the conn and remove the
                                # descriptor from our input list
                                curr_conn -= 1
                                sock_to_rcv.close()
                                sockets_rd.remove(sock_to_rcv)
                                input_conn.remove(sock_to_rcv)
                                remove_conn(input_conn_request_reply, sock_to_rcv)


            #Prepare timeout msg :)
            if not (events_rd or events_wr or events_excp):
                #print( get_str_time_ProxPy() + MSG_PROXPY_INACTIVE )
                add_to_msgHistory(msg_history,get_str_time_ProxPy() + MSG_PROXPY_INACTIVE )
                print_msgs(msg_history)
```

Capítulo 2

Logger

2.1. Proxy logger app (logger.py) - ProxPy logger source code

```
#usr/bin/env python3

import socket
import sys
import os
import pickle
import datetime
import time

#Note:
#   ProxPy app logger :)
#   For more info: github.com/davidcawork

#Global Vars
MSG_PROXPY_HI = '[ProxPy] Logger activated!'
MSG_PROXPY_LOG_DATA = '[ProxPy] Log data'
MSG_PROXPY_LOG_BYE = '[ProxPy] Bye!'
MSG_PROXPY_LOG_REQ = '[ProxPy] Log data: Request'
MSG_PROXPY_LOG_RPLY = '[ProxPy] Log data: Reply'
MSG_PROXPY_BYE = '[ProxPy] Turning off ProxPy Logger ....'
BUFFER_SIZE = 1024*5


#To get our socket UDP, where we will hear logs from ProxPy
def get_our_socket(port = '8010'):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind(('',port))
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        return s
    except:
        new_port = randint(8000, 9000)
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind(('',new_port))
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        return s

#To create log dir and get fd
```

```
def create_logs(name,current_time):
    isLogDirCreate = False
    path = os.getcwd()
    list_dir = os.listdir(path)
    LogDir = 'logs'
    for files in list_dir:
        if files == LogDir:
            try:
                isLogDirCreate = True
                log_file=open(path +'/'+LogDir+'/log_'+name+'_'+current_time.
                strftime('%Y-%m-%d')+'.log','a+')
            except:
                print('Error: cannot create log files: '+path +'/'+LogDir+'/log_'+
                name+'_'+current_time.strftime('%Y-%m-%d')+'.log')

    if not isLogDirCreate:
        os.mkdir(path+'/'+LogDir)
        try:
            log_file=open(path +'/'+LogDir+'/log_'+name+'_'+
            current_time.strftime('%Y-%m-%d')+'.log','a+')
        except:
            print('Error: cannot create log files: '+path +'/'+LogDir+'/log_'
            +name+'_'+current_time.strftime('%Y-%m-%d')+'.log')

    return log_file


#To get ProxPy str time format
def get_str_time_ProxPy():
    return ('['+(datetime.datetime.now()).strftime('%H:%M:%S')+']')

#To say welcome
def welcome():
    print(get_str_time_ProxPy() + MSG_PROXPY_HI)
#To log incoming data
def logger(file_to_log, data):

    try:
        if data[1] == MSG_PROXPY_LOG_REQ:
            file_to_log.write(get_str_time_ProxPy() +'(REQUEST) Method: ' +
            data[2][0]+' | Version: '+data[2][1]
            +' | IP_server: '+data[2][3]+' | IP_client: '+data[2][4]+' | Port_client: '
            +str(data[2][5])+' | URL: '+data[2][2]+'\n' )
        elif data[1] == MSG_PROXPY_LOG_RPLY:
            file_to_log.write(get_str_time_ProxPy() +'(Reply)    State: ' +data[2][1]
            +' | Version: '+data[2][0]
            +' | IP_server: '+data[2][3]+' | IP_client: '+data[2][4]+' | Port_client: '
            +str(data[2][5])+' | URL: '+data[2][2]+'\n' )

    except:
        file_to_log.close()
        exit(-1)



```

```python
if __name__ == "__main__":
    #Check argv's
    if len(sys.argv) != 2:
        print('Error: Usage: pyhton3 ' + sys.argv[0] + ' <Port>')
        exit(0)
    else:

        #To say welcome
        welcome()


        #Just create a socket, and bind it
        our_port = int(sys.argv[1])
        name = 'ProxPy'
        s = get_our_socket(our_port)

        #To create log dir and get fd
        current_time = datetime.datetime.now()
        logs = create_logs(name,current_time)


        try:
            while True:
                #Wait for logs :))
                data_b,addr = s.recvfrom(BUFFER_SIZE)

                #Recover the list with pickle
                data = pickle.loads(data_b)

                if data:

                    if data[0] == MSG_PROXPY_LOG_DATA:
                        logger(logs, data)

                    elif data[0] == MSG_PROXPY_LOG_BYE:
                        #Only for SSOO releases the bind made to the port
                        s.close()
                        break
                else:
                    break


        except KeyboardInterrupt:
            #Only for SSOO releases the bind made to the port
            s.close()

        print('\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
```