

Memoria Proxy Web - ProxPy

David Carrascal Acebron

Abril 2019

Índice general

1. Proxy Web	2
1.1. Proxy (proxy.py) - ProxPy source code	2
2. Logger	18
2.1. Proxy logger app (logger.py) - ProxPy logger source code	18

Capítulo 1

Proxy Web

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

```
1  #!/usr/bin/env python3
2  import socket
3  import sys
4  import os
5  import select
6  import time
7  import datetime
8  import signal
9  import pickle
10 import argparse
11 from random import *
12
13 #Note:
14 #
15 #   ProxPy(v1.1) is a web proxy with capabilities to address HTTP request 1.1.
16 #   It has a layer of filtering controlled with its powerful CLI.
17 #
18 #   The realization of this project has been carried out in the subject of
19 #   Laboratory of networks, systems and services at the University of Alcalá
20 #   by David Carrascal.
21 #
22 #   Have fun :)
23 #
24 #   For more info: github.com/davidcawork
25
26 #Global vars
27 VERSION_MAJOR_NUMBER = 1
28 VERSION_MINOR_NUMBER = 1
29 DEBUG_LEVEL_MAX = 3
30 DEBUG_LEVEL_NORMAL = 2
31 DEBUG_LEVEL_LOW = 1
32 MAX_MSG_SAVED = 20
33 MSG_PROXPY_INACTIVE = '[ProxPy] ProxPy inactive ...'
34 MSG_PROXPY_HI = '[ProxPy] Welcome to ProxPy CLI'
35 MSG_PROXPY_BYE = '[ProxPy] Turning off ProxPy ....'
36 MSG_PROXPY_VERSION = '[ProxPy] Current version is: ProxPy v'+ str(VERSION_MAJOR_NUMBER) + '.' + str(VERSION_MINOR_NUMBER)
37 MSG_PROXPY_NEW_INPUT_CONN = '[ProxPy] New input connection from: '
38 MSG_PROXPY_BLK = '[ProxPy] Blocking request to: '
39 MSG_PROXPY_BLK_CONN = '[ProxPy] Blocking connection to: '
40 MSG_PROXPY_LOG_DATA = '[ProxPy] Log data'
41 MSG_PROXPY_LOG_BYE = '[ProxPy] Bye!'
42 MSG_PROXPY_LOG_REQ = '[ProxPy] Log data: Request'
```

```

43 MSG_PROXYPY_LOG_RPLY = '[ProxPy] Log data: Reply'
44 ERROR_BAD_ARGVS_FROM_USER = '[ProxPy] Error, incorrect arguments: '
45 ERROR_TO_RCV_FROM_NAV = '[ProxPy] Error, cannot recover the request from: '
46 ERROR_TO_RCV_FROM_SW = '[ProxPy] Error, cannot recover the server reply from: '
47 ERROR_TO_SEND_REQUEST = '[ProxPy] Error, cannot send the request to the server, connecting again...'
48 ERROR_TO_CONN_WITH_SW = '[ProxPy] Error, cannot connect with Server: '
49 ERROR_TO_CLOSE_INPUT_CONN = '[ProxPy] Error, cannot close the input connections: '
50 ERROR_TO_CLOSE_OUTPUT_CONN = '[ProxPy] Error, cannot close the output connections: '
51 ERROR_TO_CLOSE_CONN = '[ProxPy] Error, cannot close the connections: '
52 ERROR_TO_BIND_OUR_PORT = '[ProxPy] Error, our listening port is already in use, instead we use port: '
53 ERROR_TO_PREPARE_REQUEST= '[ProxPy] Error, cannot prepare the request: '
54 ERROR_TO_REPLY_NAV = '[ProxPy] Error, cannot process the request: '
55 ERROR_TO_LOG_REQUEST = '[ProxPy] Error, cannot log the request'
56 ERROR_TO_LOG_REPLY = '[ProxPy] Error, cannot log the reply'
57
58 #MACROS (str)
59 CRLF = '\r\n' #Carriage return AND line feed
60 WSP = ' '
61 NSP = ''
62 COLON = ':'
63
64 #MACROS (byte)
65 CRLF_B = b'\r\n' #Carriage return AND line feed
66 WSP_B = b' '
67 NSP_B = b''
68 COLON_B = b': '
69
70 #To get our socket TCP, where we will hear connections from web navigators
71 def get_our_socket(port,msg_history):
72     try:
73         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
74         s.setblocking(0)
75         s.bind('',port)
76         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
77         s.listen(5)
78         return s
79     except:
80         new_port = randint(8000, 9000)
81         #print(get_str_time_ProxPy()+ERROR_TO_BIND_OUR_PORT+ str(new_port))
82         add_to_msgHistory(msg_history,get_str_time_ProxPy()+ERROR_TO_BIND_OUR_PORT+ str(new_port)+ '\n')
83         print_msgs(msg_history)
84         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
85         s.setblocking(0)
86         s.bind('',new_port)
87         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
88         s.listen(5)
89         return s
90
91 #Aux func to know if one item is in one list
92 def is_in_the_list(list_, element):
93
94     if list_.count(element):
95         return True
96
97     return False
98
99 #To get ProxPy str time format
100 def get_str_time_ProxPy():
101     return ('['+(datetime.datetime.now()).strftime('%H:%M:%S')+'] ')
102

```

```

103 #To parse all incoming HTTP request
104 def http_request_parser(data):
105
106     #Request dic
107     request = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-' }
108
109     http_request_parser_line(request, data.split(CRLF)[0])
110     http_request_parser_headers(request, data.split(CRLF)[1:])
111     http_request_parser_body(request, data.split(CRLF)[request['header_count'] + 1 :])
112
113     return request
114
115 #To handle bad args
116 def bad_argvs_handler():
117     print( get_str_time_ProxPy() + ERROR_BAD_ARGVS_FROM_USER + '\n\n\t Usage: python3 ' +
118           sys.argv[0] + ' <Port> .... \n\n\t For more help you can chek: python3 ' + sys.argv[0] + ' -h\n\n')
119
120 #To parse all incoming HTTP request(Just first line)
121 def http_request_parser_line(request, data):
122
123     request['method'] = data.split(WSP)[0]
124     request['uri'] = data.split(WSP)[1]
125     request['version'] = data.split(WSP)[2]
126
127 #To parse all incoming HTTP request(headers)
128 def http_request_parser_headers(request, data):
129
130     for items in data:
131         if items == NSP:
132             break
133         else:
134             request['headers_list'].append([items.split(COLON)[0], (items.split(COLON + WSP)[1]).strip()])
135             request['header_count'] += 1
136
137 #To parse all incoming HTTP request(To get the body)
138 def http_request_parser_body(request, data):
139
140     if data[0] is NSP:
141         request['body'] = '-'
142     else:
143         request['body'] = data[0]
144
145
146 #To parse all incoming HTTP request (bin)
147 def http_request_parser_bin(data):
148
149     #Request dic
150     request = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-' }
151
152     list_str_data= str(data).split('\r\n')
153     list_str_data.remove("")
154
155     http_request_parser_line(request, list_str_data[0][2:])
156     http_request_parser_headers(request, list_str_data[1:])
157     http_request_parser_body(request, list_str_data[request['header_count'] + 1 :])
158
159     return request
160
161
162 #To get host from request

```

```

163 def get_host_from_header_list(list_):
164
165     for item in list_:
166         if item[0] == 'Host':
167             return item[1]
168
169 #To read
170 def read_port_url(request):
171
172     str_uri = request['uri']
173
174     if len(str_uri.split(COLON)) == 2:
175         return 80
176     else:
177         return int(str_uri.split(COLON)[2])
178
179 #To get a conn with WS
180 def get_conn_to_server(output_conn_request_reply, request):
181
182
183
184     port = read_port_url(request)
185
186     sock_aux = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
187     try:
188         sock_aux.connect((socket.gethostbyname(get_host_from_header_list(request['headers_list'])),port))
189     except:
190         #print( get_str_time_ProxPy() + ERROR_TO_CONN_WITH_SW + get_host_from_header_list(request['headers_list'])
191         add_to_msgHistory(msg_history,get_str_time_ProxPy() + ERROR_TO_CONN_WITH_SW + get_host_from_header_list
192         print_msgs(msg_history)
193
194
195     output_conn_request_reply.append([sock_aux,socket.gethostbyname(get_host_from_header_list(request['headers
196
197     return sock_aux
198
199 #To update socket descriptor
200 def update_socket_output_conn(output_conn_request_reply,ip,sock):
201
202     for item in output_conn_request_reply:
203         if item[1] == ip:
204             item[0] = sock
205             break
206
207
208 #Returns True if we have a conn / False if we havent
209 def is_already_conn_sw(output_conn_request_reply, ip_addr, sock_to_rcv = 'default'):
210
211     for conns in output_conn_request_reply:
212         if conns[1] == ip_addr and conns[0] == sock_to_rcv:
213             return True
214
215     return False
216
217 #To append a request to active conn / return socket descriptor
218 def append_request(conn_request,ip_addr, request_to_append ):
219
220     for conns in conn_request:
221         if conns[1] == ip_addr:
222             conns[2].append(request_to_append)

```

```

223         return conns[0]
224
225 #To append a request to input_conn_request_reply list
226 def add_to_input_conn_request(input_conn_request_reply, sock_to_rcv, request):
227
228     if not is_already_conn_sw(input_conn_request_reply, sock_to_rcv.getsockname()[0], sock_to_rcv):
229         input_conn_request_reply.append([sock_to_rcv, sock_to_rcv.getsockname()[0], [request], []])
230     else:
231         append_request(input_conn_request_reply, sock_to_rcv.getsockname()[0], request)
232
233 #To send a request to server web
234 def send_request_to_sw(host_uri, request, output_conn_request_reply):
235     #Var aux
236     pet = b''
237
238     #We have to re-create the request
239     #Add first line
240     pet += WSP_B.join([(request['method']).encode(), (request['uri']).encode(), (request['version']).encode()]) +
241
242     #Add headers (Here we can add 'if' to change some header )
243     for item in request['headers_list']:
244         if item[0] == 'Connection':
245             pet += item[0].encode() + b': ' + b'Close' + CRLF_B
246         elif item[0] == 'Upgrade-Insecure-Requests':
247             pass
248         else:
249             pet += item[0].encode() + b': ' + item[1].encode() + CRLF_B
250
251     #Fin headers
252     pet += CRLF_B
253
254     #Add Body
255     if request['body'] is not '-':
256         pet += (request['body']).encode()
257
258     #Fin request
259     pet += CRLF_B
260
261
262     #Send it
263     try:
264         host_uri.sendall(pet)
265     except:
266         #print(get_str_time_ProxPy()+ ERROR_TO_SEND_REQUEST)
267         add_to_msgHistory(msg_history, get_str_time_ProxPy()+ ERROR_TO_SEND_REQUEST)
268         print_msgs(msg_history)
269         host_uri = get_conn_to_server(output_conn_request_reply, request)
270         host_uri.sendall(pet)
271
272
273 #To get the socket where we will send the reply
274 def get_input_socket_from_request(list_intput, request):
275
276     #HTTP replies in order
277     for item in input_conn_request_reply:
278         if item[2][0] == request:
279             return item[0]
280
281 #To get the socket where we will send the request
282 def get_output_socket_from_request(list_out, ip):

```

```

283
284     for item in list_out:
285         if item[1] == ip:
286             return item[0]
287
288     #To print msg_history
289     def print_msgs(msg_history):
290         sys.stdout.flush()
291         os.system('clear')
292         for msg in msg_history:
293             print(msg)
294     #To manege msg's
295     def add_to_msgHistory(msg_history,msg):
296
297         if len(msg_history) == MAX_MSG_SAVED:
298             msg_history.pop(0)
299
300         msg_history.append(msg)
301
302     #to get cmd written
303     def is_command(msg, str_cmd):
304         return msg.count(str_cmd)
305
306     #To print /help cmd
307     def print_help():
308         sys.stdout.flush()
309         os.system('clear')
310         print('Hi User !\n\n')
311         print('These are the commands that you can use:\n')
312         print('\t/help\t\tTo consult the commands and guides for using the ProxPy CLI')
313         print('\t/quit\t\tTo exit, it close all connections')
314         print('\t/timeup\t\tTo get the time you have connected in ProxPy CLI')
315         print('\t/stats\t\tTo get statistics about the activity of ProxPy and attributes of it')
316         print('\t/reload\t\tTo reload all connections')
317         print('\t/showfilter\tTo show the current filter rules')
318         print('\t/filter_server\tTo add an URL to permit in filter rules')
319         print('\t/filter_client\tTo add an User/s to permit in filter rules [Netmask avaible /0 /8 /16 /24 /32]')
320         print('\t/showfilter\tTo show the current filter rules')
321         print('\t/debug [id]\tTo set debug level')
322         print('\t/max_conn [num]\tTo set max connection number')
323         print('\t/timeout [sec]\tTo set the activity timer (seconds)')
324
325         print('\n\nFor more help you can check: https://github.com/davidcawork\n\n')
326
327         input("Press Enter to continue...")
328         os.system('clear')
329
330     #To get time up in ProxPy
331     def timeup_cmd(time_init):
332         os.system('clear')
333         print('Hi User !\n\n')
334         time_b = datetime.datetime.now()
335         print('You have '+str(time_b - time_init)+' time in ProxPy n.n\n\n')
336         input("Press Enter to continue...")
337         os.system('clear')
338
339     #To get the request associate with a socket
340     def get_request_from_output_conn(output_conn_request_reply, sock_to_rcv):
341
342         #HTTP replys in order

```



```

343     for item in output_conn_request_reply:
344         if item[0] == sock_to_rcv:
345             return item[2][0]
346
347     #To remove a conn from list
348     def remove_conn(list_to_rm, socket_to_rm):
349
350         try:
351             for item in list_to_rm:
352                 if item[0] == socket_to_rm:
353                     list_to_rm.remove(item)
354         except:
355             add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN + ' Value conn list '+str(list_to_rm))
356             print_msgs(msg_history)
357             #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN + ' Value conn list '+str(list_to_rm))
358
359
360     # To close al connections (Web navigators and SW)
361     def close_all_conn(sockets_rd, input_conn, output_conn):
362
363         try:
364             for sck_in in input_conn:
365                 sck_in.close()
366             input_conn.clear()
367         except:
368             #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_INPUT_CONN + ' Value input conn list '+str(input_conn))
369             add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_CLOSE_INPUT_CONN + ' Value input conn list '+str(input_conn))
370             print_msgs(msg_history)
371         try:
372             for sck_out in output_conn:
373                 sck_out.close()
374             output_conn.clear()
375         except:
376             #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_OUTPUT_CONN + ' Value output conn list '+str(output_conn))
377             add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_CLOSE_OUTPUT_CONN + ' Value output conn list '+str(output_conn))
378             print_msgs(msg_history)
379         try:
380             for sck in sockets_rd:
381                 if sck != sys.stdin:
382                     sck.close()
383             sockets_rd.clear()
384             sockets_rd.append(sys.stdin)
385         except:
386             #print( get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN + ' Value conn list '+str(sockets_rd))
387             add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_CLOSE_CONN + ' Value conn list '+str(sockets_rd))
388             print_msgs(msg_history)
389
390
391     #Welcome msg
392     def welcome(msg_history):
393         #print(get_str_time_ProxPy() + MSG_PROXPY_HI + '\n' + get_str_time_ProxPy() + MSG_PROXPY_VERSION)
394         add_to_msgHistory(msg_history, get_str_time_ProxPy() + MSG_PROXPY_HI + '\n' + get_str_time_ProxPy() + MSG_PROXPY_VERSION)
395         print_msgs(msg_history)
396
397     #Handler CTRL+C
398     def signal_handler(sig, frame):
399         print( '\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
400         #Close al connections (Web navigators and SW) and exit
401         close_all_conn(sockets_rd, input_conn, output_conn)
402         sys.exit(0)

```

```

403
404 #To get UDP sockets desc.
405 def get_logger_socket():
406     return socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
407
408 #To send to our logger request info
409 def send_to_logger_request(logger, logger_id, ip_client, ip_dest_, port_client,request, msg_history):
410     try:
411         # Our pkt : [DATA, REQ/RPLY, [method, version, server(url), server(ip), client(ip), client(port)]]
412         logger.sendto(pickle.dumps([MSG_PROXYPY_LOG_DATA, MSG_PROXYPY_LOG_REQ,[request['method'], request['version']
413     except:
414         #print( get_str_time_ProxPy() + ERROR_TO_LOG_REQUEST )
415         add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_LOG_REQUEST )
416         print_msgs(msg_history)
417
418 #To send to our logger reply info
419 def send_to_logger_reply(logger, logger_id, ip_client, ip_dest_, port_client,request, request_2,msg_history):
420     try:
421         # Our pkt : [DATA, REQ/RPLY, [method, version, server(url), server(ip), client(ip), client(port)]]
422         logger.sendto(pickle.dumps([MSG_PROXYPY_LOG_DATA, MSG_PROXYPY_LOG_RPLY,[request['method'], request['uri']
423     except:
424         #print( get_str_time_ProxPy() + ERROR_TO_LOG_REPLY )
425         add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_LOG_REPLY )
426         print_msgs(msg_history)
427
428 #Init argparse
429 def init_argvs():
430
431     parser = argparse.ArgumentParser(description="Welcome to ProxPy's help page", epilog='For more help you can
432     parser.add_argument('-p', '--port',metavar='Port',type= int,default=8080,help='Provide an integer that will
433     parser.add_argument('-d', '--debug',metavar='Debug',type= int,default=3,help='Provide an integer that will b
434     parser.add_argument('-t', '--timeout',metavar='Timeout',type= int,default=300,help='Provide an integer that
435     parser.add_argument('-b', '--buffer',type= int,default=1024*1000,help='Provide an integer that will be our b
436     parser.add_argument('-c', '--max_conn',type= int,default=8,help='Provide an integer that will be max client
437     parser.add_argument('-fs', '--filter_server',type= str,default="",help='Provide an [url] to restrict access
438     parser.add_argument('-fc', '--filter_client',type= str,default="",help='Provide an IP range that will be per
439
440     return parser
441
442 #To prepare and parse all argvs
443 def prepare_argvs(parser,proxy_port,proxy_timeout,debug_mode,max_client_conn,BUFFER_SIZE,list_filter_server,li
444
445     my_args = parser.parse_args()
446
447     proxy_port= my_args.port
448     proxy_timeout= float(my_args.timeout)
449     debug_mode=my_args.debug
450     BUFFER_SIZE=my_args.buffer
451     max_client_conn=my_args.max_conn
452
453     if my_args.filter_server != '':
454         list_filter_server.append(my_args.filter_server)
455
456     if my_args.filter_client != '':
457         list_filter_client.append(my_args.filter_client)
458
459     return len(vars(my_args))
460
461 def http_reply_parser_bin(reply_container):
462

```

```

463     #Request dic
464     reply = { 'method': '-', 'version': '-', 'uri': '-', 'header_count': 0, 'headers_list': [], 'body': '-' }
465
466     list_str_data= str(reply_container).split('\\r\\n')
467     #list_str_data.remove("")
468
469     http_request_parser_line(reply, list_str_data[0][2:])
470
471     return reply
472
473     #To filter via SW and ip range
474     def should_process_request(request,client_ip, list_filter_server, list_filter_client):
475         permit = False
476
477         if len(list_filter_server) == 0 and len(list_filter_client) == 0:
478             #No filters :)
479             return True
480
481         elif len(list_filter_server) != 0 and len(list_filter_client) == 0:
482             #In case there is a filter by allowed servers
483             for item in list_filter_server:
484                 if item == get_host_from_header_list(request['headers_list']):
485                     permit = True
486             return permit
487
488         elif len(list_filter_server) == 0 and len(list_filter_client) != 0:
489             #In case there is a filter by ip range
490             same_net= 0
491             mask = int(int(list_filter_client[0].split('/')[1])/8)
492
493             if mask == 0:
494                 return True
495             else:
496                 mask_numbers = (list_filter_client[0].split('/')[0]).split('.')
497                 mask_numbers_int = []
498
499                 for number in mask_numbers:
500                     mask_numbers_int.append(int(number))
501
502                 client_list_str = client_ip.split('.')
503                 client_ip_int = []
504                 for number in client_list_str:
505                     client_ip_int.append(int(number))
506
507                 for i in range(0,mask):
508                     if client_ip_int[i] == mask_numbers_int[i]:
509                         same_net += 1
510
511                 if same_net == mask:
512                     return True
513                 else:
514                     return False
515
516         else:
517             #In case there is a filter by ip range and allowed servers
518
519             same_net= 0
520             mask = int(int(list_filter_client[0].split('/')[1])/8)
521
522             if mask == 0:

```

```

523         permit = True
524     else:
525         mask_numbers = (list_filter_client[0].split('/')[0]).split('.')
526         mask_numbers_int = []
527
528         for number in mask_numbers:
529             mask_numbers_int.append(int(number))
530
531         client_list_str = client_ip.split('.')
532         client_ip_int = []
533         for number in client_list_str:
534             client_ip_int.append(int(number))
535
536         for i in range(0,mask):
537             if client_ip_int[i] == mask_numbers_int[i]:
538                 same_net += 1
539
540         if same_net == mask:
541             permit = True
542         else:
543             return False
544
545         for item in list_filter_server:
546             if item == get_host_from_header_list(request['headers_list']):
547                 permit = True
548
549         return permit
550 #CLI utils
551 def getServer_Ulrl(msg):
552
553     try:
554         return msg.split(' ')[1]
555     except:
556         return ''
557
558 def getInt_msg(msg):
559
560     try:
561         return int(msg.split(' ')[1])
562     except:
563         return 15
564
565 #To get our stats :)
566 def stats_cmd(cmd_used,time_init,n_reply,n_request,debug_mode,max_client_conn,BUFFER_SIZE,proxy_timeout,len_msg_history):
567
568     os.system('clear')
569     print('Hi User !\n\n')
570     time_b = datetime.datetime.now()
571     print('ProxPy stats:\n')
572     print('1.\t Request sent : '+str(n_request))
573     print('2.\t Reply rcv    : '+str(n_reply))
574     print('3.\t Debug level   : '+str(debug_mode))
575     print('4.\t Max conn      : '+str(max_client_conn))
576     print('5.\t Buffer size    : '+str(BUFFER_SIZE))
577     print('6.\t Time out      : '+str(proxy_timeout))
578     print('7.\t Commands used: '+str(cmd_used))
579     print('8.\t msg_History   : '+str(len_msg_history))
580     print('9.\t Time up       : '+str(time_b -time_init)+'\n\n')
581
582     input("Press Enter to continue...")

```

```

583     os.system('clear')
584
585 def print_filter_table(filter_client, filter_server):
586     os.system('clear')
587     print('Hi User !\n\n')
588
589     print('\t\t-- Permit Server table --\n')
590     for peer in filter_server:
591         print('+ \t\tName: '+peer)
592
593     print('\n\n\t\t-- Permit Clients table --\n')
594     for peer in filter_client:
595         print('+ \t\tIP_range: '+peer)
596
597     print('\n\n')
598     input("Press Enter to continue...")
599     os.system('clear')
600
601 #Main
602 if __name__ == "__main__":
603
604
605     # --- Vars ---
606     msg_history = []
607     parser = init_argvs()
608     proxy_port = 8080
609     proxy_timeout = 300.0
610     list_filter_server = []
611     list_filter_client = []
612     max_client_conn = 8
613     curr_conn = 0
614     debug_mode = 1
615     BUFFER_SIZE = 1024*1000
616     reply_container= b''
617     logger_id = ['localhost', 8010]
618
619
620     #To parse args
621     len_argvs = prepare_argvs(parser,proxy_port,proxy_timeout,debug_mode,max_client_conn,BUFFER_SIZE,list_filt
622     my_args = parser.parse_args()
623     proxy_port= my_args.port
624     proxy_timeout= float(my_args.timeout)
625     debug_mode=my_args.debug
626     BUFFER_SIZE=my_args.buffer
627     max_client_conn=my_args.max_conn
628
629     #Check argv's and init args parser
630     if len_argvs < 2:
631         bad_argvs_handler()
632         exit(0)
633
634     else:
635
636         # --- Say welcome to ProxPy and print current version ---
637         welcome(msg_history)
638
639         #Let's to prepare the CTRL + C signal to handle it and be able to show the statistics before it comes
640         signal.signal(signal.SIGINT, signal_handler)
641
642         # --- Prepare our TCP socket where we will hear connections from web navigators ---

```

```

643         our_proxy_socket = get_our_socket(proxy_port, msg_history)
644
645         # --- Prepare our udp socket where w'll log every single pet ---
646         logger = get_logger_socket()
647
648         #To save all msg and stats
649         time_init = datetime.datetime.now()
650         cmd_used = 0
651         n_request = 0
652         n_reply = 0
653
654         #Sockets to read
655         sockets_rd = [sys.stdin, our_proxy_socket]
656
657         #To save
658         #Input connections
659         input_conn = []
660         input_conn_request_reply = []
661
662         #Output connections
663         output_conn = []
664         output_conn_request_reply = []
665
666         #We'll store the request like this : [ [sockets_descriptor, str_host, [current_request_1, current_request_2, ...], ... ] ]
667
668         # We can exit by CTRL+C signal :)
669         while True:
670             try:
671                 # The optional timeout argument specifies a time-out as a floating point number in seconds
672                 events_rd, events_wr, events_excp = select.select( sockets_rd, [], [], proxy_timeout)
673
674                 except KeyboardInterrupt:
675                     add_to_msgHistory(msg_history, '\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE + '\n')
676                     print_msgs(msg_history)
677                     #print( '\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
678                     #Close al connections (Web navigators and SW) and exit
679                     close_all_conn(sockets_rd, input_conn, output_conn)
680                     sys.exit(0)
681
682                 for event in events_rd:
683
684                     if event == our_proxy_socket:
685
686                         if curr_conn <= max_client_conn:
687                             #Accept input conn from web navigator
688                             conn, addr = our_proxy_socket.accept()
689                             conn.setblocking(0)
690                             sockets_rd.append(conn)
691                             input_conn.append(conn)
692                             curr_conn+=1
693                         else:
694                             conn,addr = our_proxy_socket.accept()
695                             conn.close()
696                             if debug_mode >= DEBUG_LEVEL_NORMAL:
697                                 add_to_msgHistory(msg_history, get_str_time_ProxPy() + MSG_PROXPY_BLCK_CONN + addr[0] + '\n')
698                                 print_msgs(msg_history)
699                                 #print(get_str_time_ProxPy() + MSG_PROXPY_BLCK_CONN + addr[0] + ':' + str(addr[1]))
700
701                 #CLI ProxPy v1.1
702                 # Stdin event

```

```

703         elif event is sys.stdin:
704             msg = input()
705
706             if is_command(msg, '/quit'):
707                 #To shutdown ProxPy
708                 os.system('clear')
709                 close_all_conn(sockets_rd, input_conn, output_conn)
710                 add_to_msgHistory(msg_history, '\n\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE + '\n')
711                 print_msgs(msg_history)
712                 sys.exit(0)
713
714             elif is_command(msg, '/help'):
715                 #To print help msg
716                 cmd_used +=1
717                 print_help()
718                 print_msgs(msg_history)
719
720             elif is_command(msg, '/filter_client'):
721                 #To permit some client
722                 cmd_used +=1
723                 client_to_permit = getServer_Ulr(msg)
724                 list_filter_client.append(client_to_permit)
725                 print_msgs(msg_history)
726
727             elif is_command(msg, '/filter_server'):
728                 #To permit some url
729                 cmd_used +=1
730                 server_to_permit = getServer_Ulr(msg)
731                 list_filter_server.append(server_to_permit)
732                 print_msgs(msg_history)
733
734             elif is_command(msg, '/debug'):
735
736                 cmd_used +=1
737                 debug_mode = getInt_msg(msg)
738                 print_msgs(msg_history)
739
740             elif is_command(msg, '/max_conn'):
741
742                 cmd_used +=1
743                 max_client_conn = getInt_msg(msg)
744                 print_msgs(msg_history)
745
746             elif is_command(msg, '/timeout'):
747
748                 cmd_used +=1
749                 proxy_timeout = float(getInt_msg(msg))
750                 print_msgs(msg_history)
751
752             elif is_command(msg, '/showfilter'):
753
754                 cmd_used +=1
755                 print_filter_table(list_filter_client, list_filter_server)
756                 print_msgs(msg_history)
757
758             elif is_command(msg, '/timeup'):
759                 #To print time up
760                 cmd_used +=1
761                 timeup_cmd(time_init)
762                 print_msgs(msg_history)

```

```

763
764         elif is_command(msg, '/stats'):
765             #To print our stats
766             cmd_used += 1
767             stats_cmd(cmd_used, time_init, n_reply, n_request, debug_mode, max_client_conn, BUFFER_SIZE, P
768             print_msgs(msg_history)
769         else:
770             #To support non cmd data
771             now = datetime.datetime.now()
772             add_to_msgHistory(msg_history, '[' + now.strftime('%H:%M:%S') + '] ProxPy(CLI): ~/$ ' + msg)
773             print_msgs(msg_history)
774
775
776     else:
777         #Handle other conn
778         for sock_to_rcv in sockets_rd:
779             #To manage request from web nav. connections
780             if sock_to_rcv != our_proxy_socket and sock_to_rcv is event and is_in_the_list(input_co
781
782                 #Recover request from Web nav
783                 try:
784                     data = sock_to_rcv.recv(BUFFER_SIZE)
785                 except:
786                     #print( get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + sock_to_rcv.getsockname() + '\n')
787                     add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + s
788                     print_msgs(msg_history)
789                     continue
790
791                 if data:
792                     #Parse the request
793                     try:
794                         if debug_mode > DEBUG_LEVEL_NORMAL:
795                             print("{}".format(data.decode('utf-8')))
796                             #request = http_request_parser(data.decode('utf-8'))
797                             request = http_request_parser_bin(data)
798
799                     except:
800                         if debug_mode >= DEBUG_LEVEL_NORMAL:
801                             #print( get_str_time_ProxPy() + ERROR_TO_RCV_FROM_NAV + ' \n\n'+str(da
802                             add_to_msgHistory(msg_history, '\n\n' + get_str_time_ProxPy() + ERROR_TO
803                             print_msgs(msg_history)
804                             curr_conn -= 1
805                             sock_to_rcv.close()
806                             sockets_rd.remove(sock_to_rcv)
807                             input_conn.remove(sock_to_rcv)
808                             remove_conn(input_conn_request_reply, sock_to_rcv)
809                             continue
810
811                 #Process the request
812
813                 #Filter
814                 if request['method'] == 'GET' and get_host_from_header_list(request['headers_li
815                 if should_process_request(request, sock_to_rcv.getsockname()[0], list_filter
816                 try:
817                     #Open connection to get uri
818                     host_uri = get_conn_to_server(output_conn_request_reply, request)
819
820                     #Add to input_conn_request_reply the request
821                     add_to_input_conn_request(input_conn_request_reply, sock_to_rcv, re
822

```



```

823         #Add to sockets_rd only if its necessary
824         if not is_in_the_list(sockets_rd, host_uri):
825             sockets_rd.append(host_uri)
826
827         #Add to output_conn only if its necessary
828         if not is_in_the_list(output_conn, host_uri):
829             output_conn.append(host_uri)
830
831         #Send the request and add to the list output_conn_request_reply, and
832         send_to_logger_request(logger, logger_id, sock_to_rcv.getsockname())
833         send_request_to_sw(host_uri, request, output_conn_request_reply)
834         n_request += 1
835     except:
836         if debug_mode >= DEBUG_LEVEL_NORMAL:
837             #print(get_str_time_ProxPy() + ERROR_TO_PREPARE_REQUEST + request)
838             add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_PREPARE_REQUEST + request)
839             print_msgs(msg_history)
840         continue
841
842     #Main handler request
843     while True:
844         try:
845             #If host_uri sockets is closed, we try reconnect with the web server
846             #Then we recv the reply to our request
847             if host_uri._closed:
848                 host_uri = get_conn_to_server(output_conn_request_reply, request)
849                 data_rpl = host_uri.recv(BUFFER_SIZE)
850             else:
851                 data_rpl = host_uri.recv(BUFFER_SIZE)
852                 reply_container += data_rpl
853             if data_rpl:
854                 #If data is not b' ' we send it back to web navigator
855                 sock_to_rcv.send(data_rpl)
856             else:
857                 #When we have sent it the reply close the host_uri socket
858                 try:
859                     reply = http_reply_parser_bin(reply_container)
860                     send_to_logger_reply(logger, logger_id, logger_id[0], sock_to_rcv.getsockname())
861                 except:
862                     pass
863                 n_reply += 1
864                 host_uri.close()
865                 sockets_rd.remove(host_uri)
866                 output_conn.remove(host_uri)
867                 remove_conn(output_conn_request_reply, host_uri)
868                 reply_container = b''
869                 break
870         except:
871             if debug_mode >= DEBUG_LEVEL_NORMAL:
872                 #print(get_str_time_ProxPy() + ERROR_TO_REPLY_NAV + request)
873                 add_to_msgHistory(msg_history, get_str_time_ProxPy() + ERROR_TO_REPLY_NAV + request)
874                 print_msgs(msg_history)
875             break
876
877     else:
878         if debug_mode >= DEBUG_LEVEL_NORMAL:
879             #print(get_str_time_ProxPy() + MSG_PROXPY_BLACK + get_host_from_header(request))
880             add_to_msgHistory(msg_history, get_str_time_ProxPy() + MSG_PROXPY_BLACK + get_host_from_header(request))
881             print_msgs(msg_history)
882         curr_conn -= 1

```

```

883         sock_to_rcv.close()
884         sockets_rd.remove(sock_to_rcv)
885         input_conn.remove(sock_to_rcv)
886         remove_conn(input_conn_request_reply, sock_to_rcv)
887
888     else:
889         curr_conn -= 1
890         sock_to_rcv.close()
891         sockets_rd.remove(sock_to_rcv)
892         input_conn.remove(sock_to_rcv)
893         remove_conn(input_conn_request_reply, sock_to_rcv)
894
895     else:
896         #Only when we have rcv b' ' from web navigator, close the conn and remove the s
897         # descriptor from our input list
898         curr_conn -= 1
899         sock_to_rcv.close()
900         sockets_rd.remove(sock_to_rcv)
901         input_conn.remove(sock_to_rcv)
902         remove_conn(input_conn_request_reply, sock_to_rcv)
903
904
905     #Prepare timeout msg :)
906     if not (events_rd or events_wr or events_excp):
907         #print( get_str_time_ProxPy() + MSG_PROXY_INACTIVE )
908         add_to_msgHistory(msg_history, get_str_time_ProxPy() + MSG_PROXY_INACTIVE )
909         print_msgs(msg_history)

```

Capítulo 2

Logger

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

```
1  #usr/bin/env python3
2
3  import socket
4  import sys
5  import os
6  import pickle
7  import datetime
8  import time
9
10 #Note:
11 #   ProxPy app logger :)
12 #   For more info: github.com/davidcawork
13
14 #Global Vars
15 MSG_PROXPY_HI = '[ProxPy] Logger activated!'
16 MSG_PROXPY_LOG_DATA = '[ProxPy] Log data'
17 MSG_PROXPY_LOG_BYE = '[ProxPy] Bye!'
18 MSG_PROXPY_LOG_REQ = '[ProxPy] Log data: Request'
19 MSG_PROXPY_LOG_RPLY = '[ProxPy] Log data: Reply'
20 MSG_PROXPY_BYE = '[ProxPy] Turning off ProxPy Logger ....'
21 BUFFER_SIZE = 1024*5
22
23
24 #To get our socket UDP, where we will hear logs from ProxPy
25 def get_our_socket(port = '8010'):
26     try:
27         s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
28         s.bind(('',port))
29         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
30         return s
31     except:
32         new_port = randint(8000, 9000)
33         s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
34         s.bind(('',new_port))
35         s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
36         return s
37
38 #To create log dir and get fd
```

```

39 def create_logs(name,current_time):
40     isLogDirCreate = False
41     path = os.getcwd()
42     list_dir = os.listdir(path)
43     LogDir = 'logs'
44     for files in list_dir:
45         if files == LogDir:
46             try:
47                 isLogDirCreate = True
48                 log_file=open(path + '/' + LogDir + '/log_'+name+'_'+current_time.
49                     strftime('%Y-%m-%d')+'.log','a')
50             except:
51                 print('Error: cannot create log files: '+path + '/' + LogDir + '/log_'+
52                     name+'_'+current_time.strftime('%Y-%m-%d')+'.log')
53
54     if not isLogDirCreate:
55         os.mkdir(path + '/' + LogDir)
56         try:
57             log_file=open(path + '/' + LogDir + '/log_'+name+'_'+
58                 current_time.strftime('%Y-%m-%d')+'.log','a')
59         except:
60             print('Error: cannot create log files: '+path + '/' + LogDir + '/log_'+
61                 name+'_'+current_time.strftime('%Y-%m-%d')+'.log')
62
63     return log_file
64
65
66 #To get ProxPy str time format
67 def get_str_time_ProxPy():
68     return ('['+(datetime.datetime.now()).strftime('%H:%M:%S')+']')
69
70 #To say welcome
71 def welcome():
72     print(get_str_time_ProxPy() + MSG_PROXPY_HI)
73 #To log incoming data
74 def logger(file_to_log, data):
75
76     try:
77         if data[1] == MSG_PROXPY_LOG_REQ:
78             file_to_log.write(get_str_time_ProxPy() + '(REQUEST) Method: ' +
79                 data[2][0]+' | Version: '+data[2][1]
80                 +' | IP_server: '+data[2][3]+' | IP_client: '+data[2][4]+' | Port_client: '
81                 +str(data[2][5])+' | URL: '+data[2][2]+'\\n' )
82         elif data[1] == MSG_PROXPY_LOG_RPLY:
83             file_to_log.write(get_str_time_ProxPy() + '(Reply) State: ' +data[2][1]
84                 +' | Version: '+data[2][0]
85                 +' | IP_server: '+data[2][3]+' | IP_client: '+data[2][4]+' | Port_client: '
86                 +str(data[2][5])+' | URL: '+data[2][2]+'\\n' )
87
88     except:
89         file_to_log.close()
90         exit(-1)
91
92
93

```

```

94  if __name__ == "__main__":
95      #Check argv's
96      if len(sys.argv) != 2:
97          print('Error: Usage: pyhton3 ' + sys.argv[0] + ' <Port>')
98          exit(0)
99      else:
100
101          #To say welcome
102          welcome()
103
104
105          #Just create a socket, and bind it
106          our_port = int(sys.argv[1])
107          name = 'ProxPy'
108          s = get_our_socket(our_port)
109
110          #To create log dir and get fd
111          current_time = datetime.datetime.now()
112          logs = create_logs(name,current_time)
113
114
115      try:
116          while True:
117              #Wait for logs :))
118              data_b,addr = s.recvfrom(BUFFER_SIZE)
119
120              #Recover the list with pickle
121              data = pickle.loads(data_b)
122
123              if data:
124
125                  if data[0] == MSG_PROXPY_LOG_DATA:
126                      logger(logs, data)
127
128                  elif data[0] == MSG_PROXPY_LOG_BYE:
129                      #Only for SS00 releases the bind made to the port
130                      s.close()
131                      break
132              else:
133                  break
134
135      except KeyboardInterrupt:
136          #Only for SS00 releases the bind made to the port
137          s.close()
138
139      print('\n\n'+get_str_time_ProxPy() + MSG_PROXPY_BYE)
140

```