PE File Structure Security Enumeration

Enumerating PE File Structure Security and Custom Base 64 Steganography

Special Thanks

To my mentors, without them I wouldn't be here today.

- **Travis Barlow**
- **Kathryn Dumke**

Introduction

Who is the new girl?

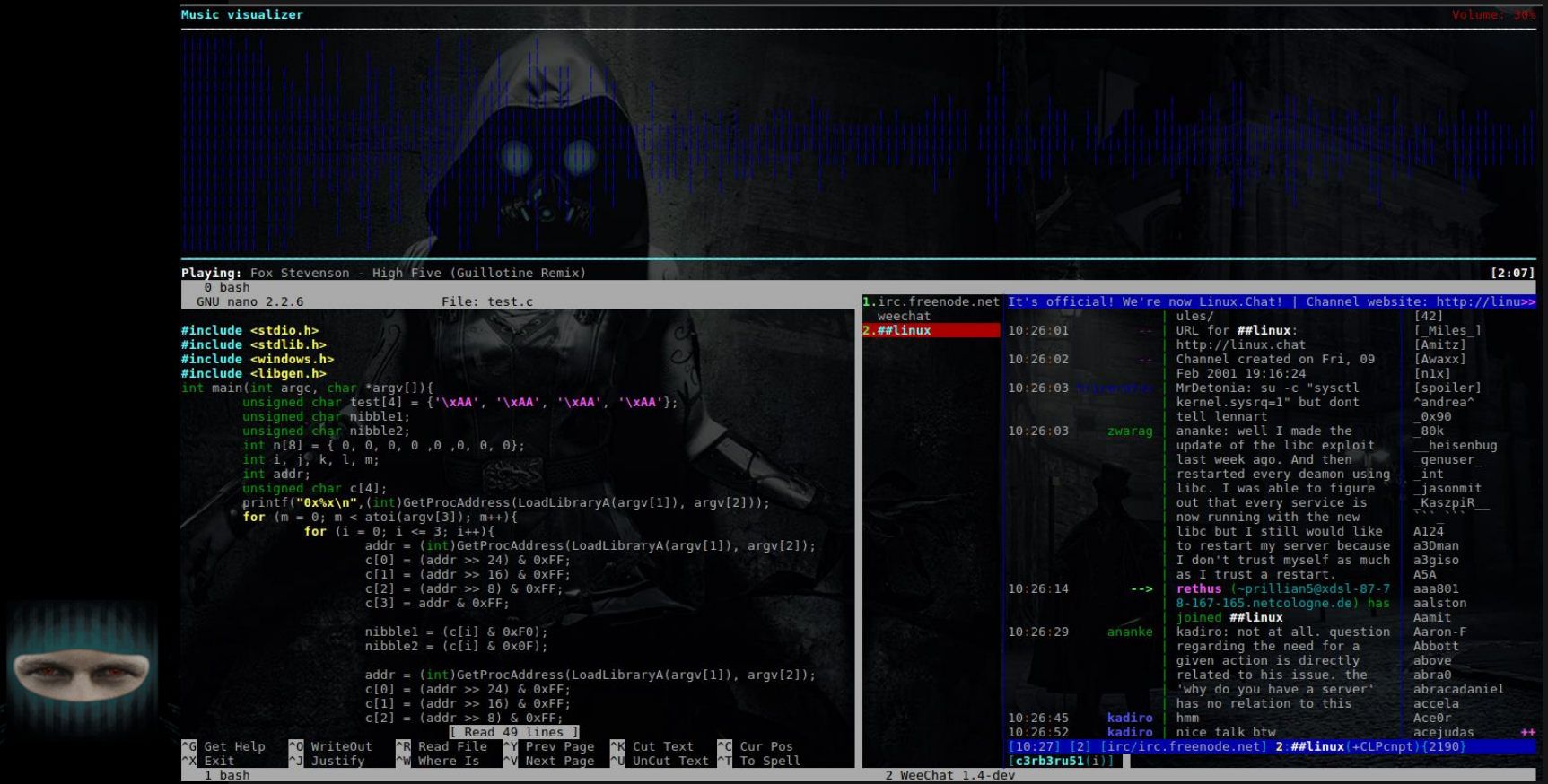
- We will be doing pictures
- They are faster than words trust me

Who I think I am

What I really am

What my family thinks I do

What I actually do



Disclaimer

Presentation Legal Notes

- This presentation is for informational purposes only
- Use this information at your own risk
- I won't bail you out of jail
- This presentation does not reflect the views or interests of GoSecure

PE File Structure

The Ground Rules

- When we talk about PE File Structures we will be referring directly to DLLs (Dynamic Link Libraries)
- We are only interested in gaining information to leverage an exploit on a particular application, all other information we can leave behind
- Slides and PE File Structure Security Roadmap will be available on GitHub after the presentation
- I'm in no way responsible for your actions based on the information presented today

PE File Structure

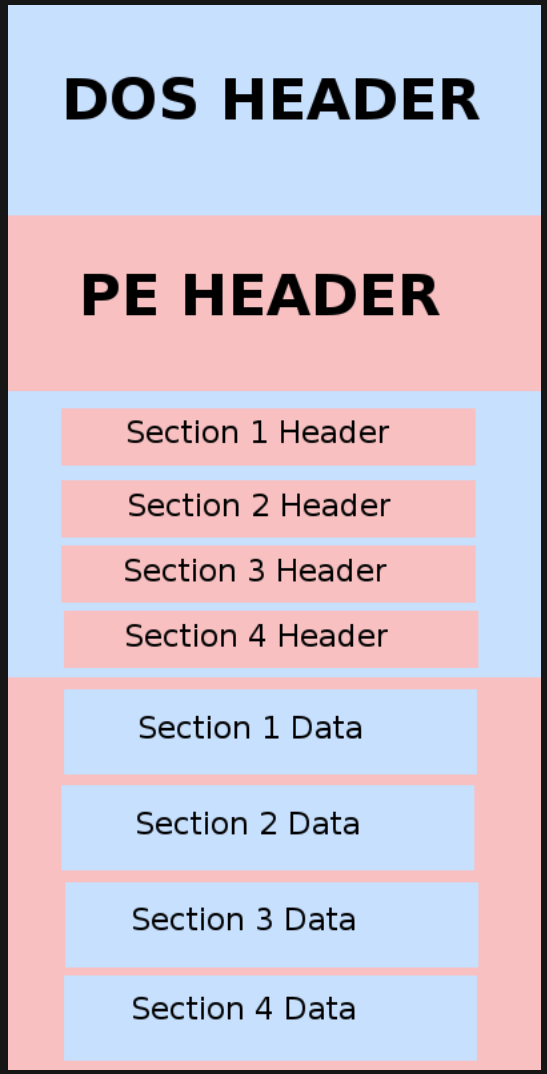
What can this be used for?

- **Analysis of Malware**
- **Enumerate Security Protections**
- **Securing Vendor Applications without Source**
- **Exploit Development**

PE File Structure

High Level Overview

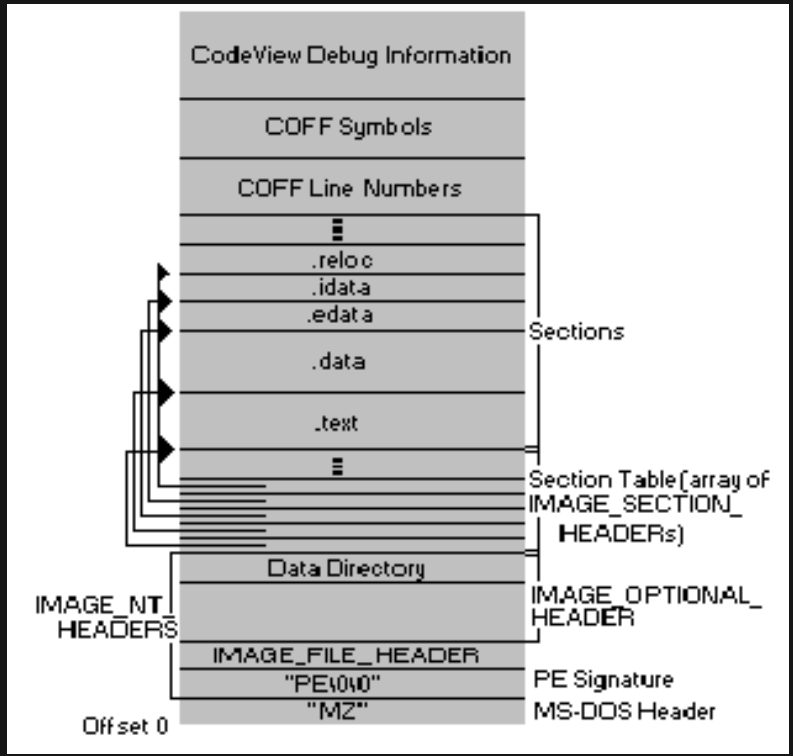
- Microsoft moved to the PE file format for their executable in Windows NT 3.1 (DOS Header)
- It has retained legacy support
- This is where we find data for typical segments when reverse engineering .text, .data, etc.



PE File Structure

Going into more detail...

- Our main focus is in the IMAGE_NT_HEADERS Section
- Take note of how we get pointers to each respective section in the binary from the headers
- .data and .text



PE File Structure

Going Deeper

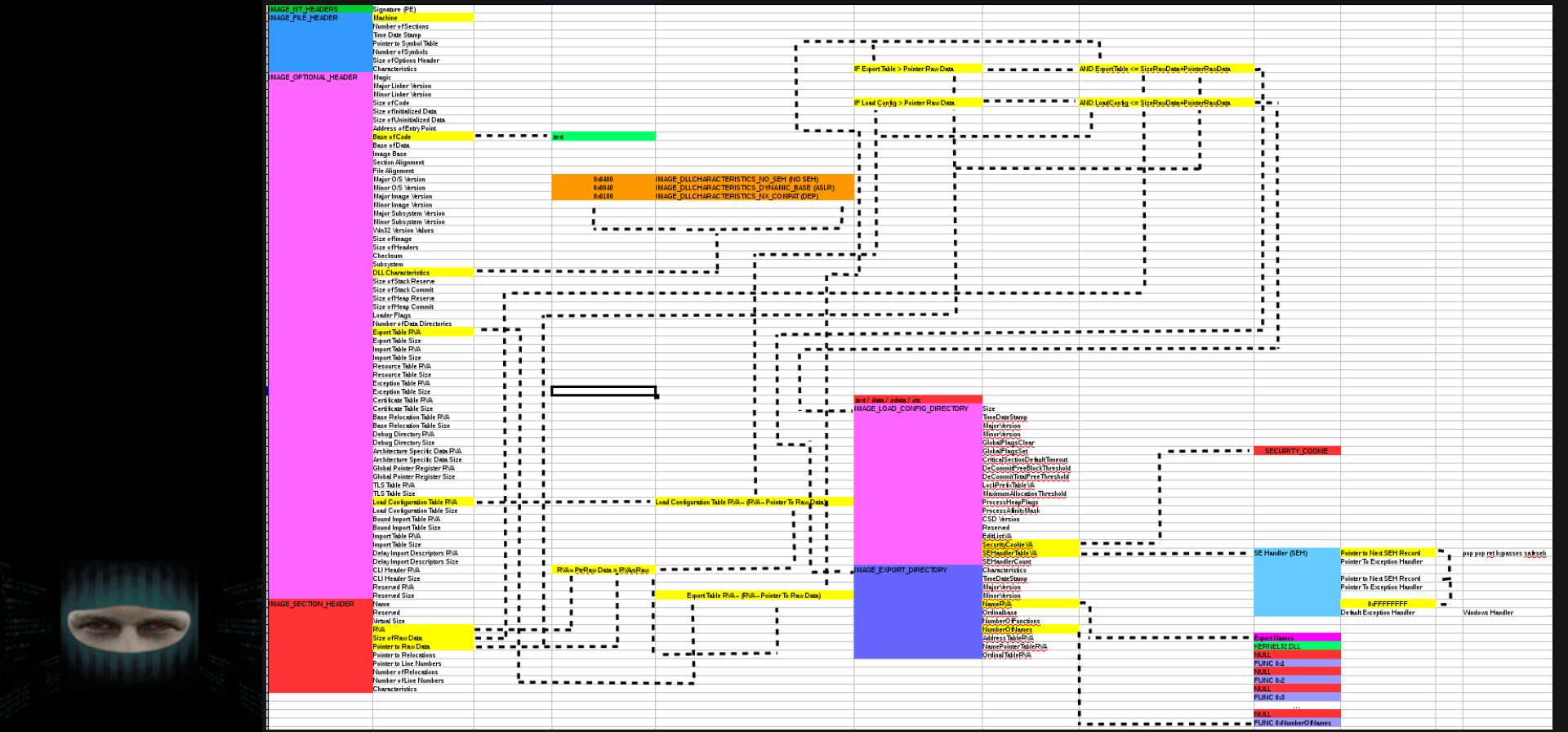
- We will be looking for the Export Names table
- Then we will use a few functions of windows.h to help use extract their location in memory when loaded
- I will then go over a algorithm that can extract how many bits of entropy we are dealing with
- Before we begin we must know the difference between a RVA and a Raw Address.

PE File Structure

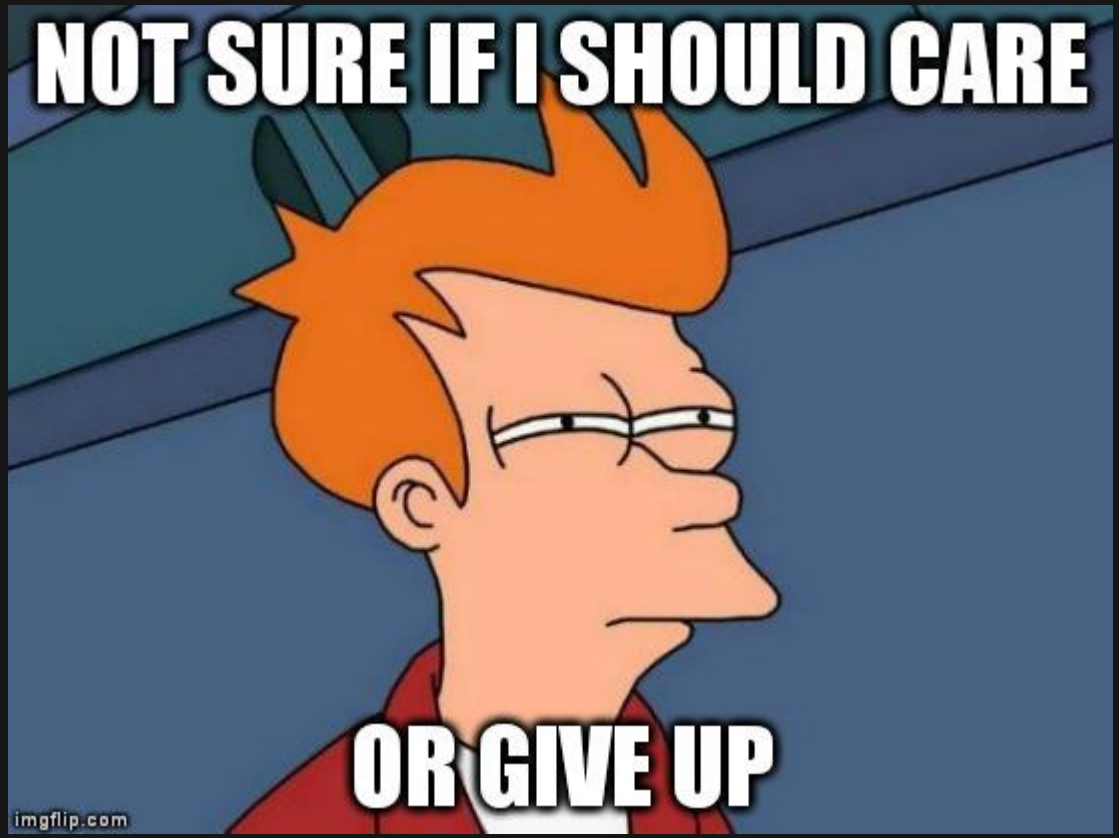
RVA and Raw Pointers

- RVA (Relative Virtual Address) – The address of an item after it's loaded into memory
- If there is a difference between the RVA and Pointer to Raw Data then we must take their difference into consideration
- Now let's zoom in closer to the file structure

PE Security Road Map

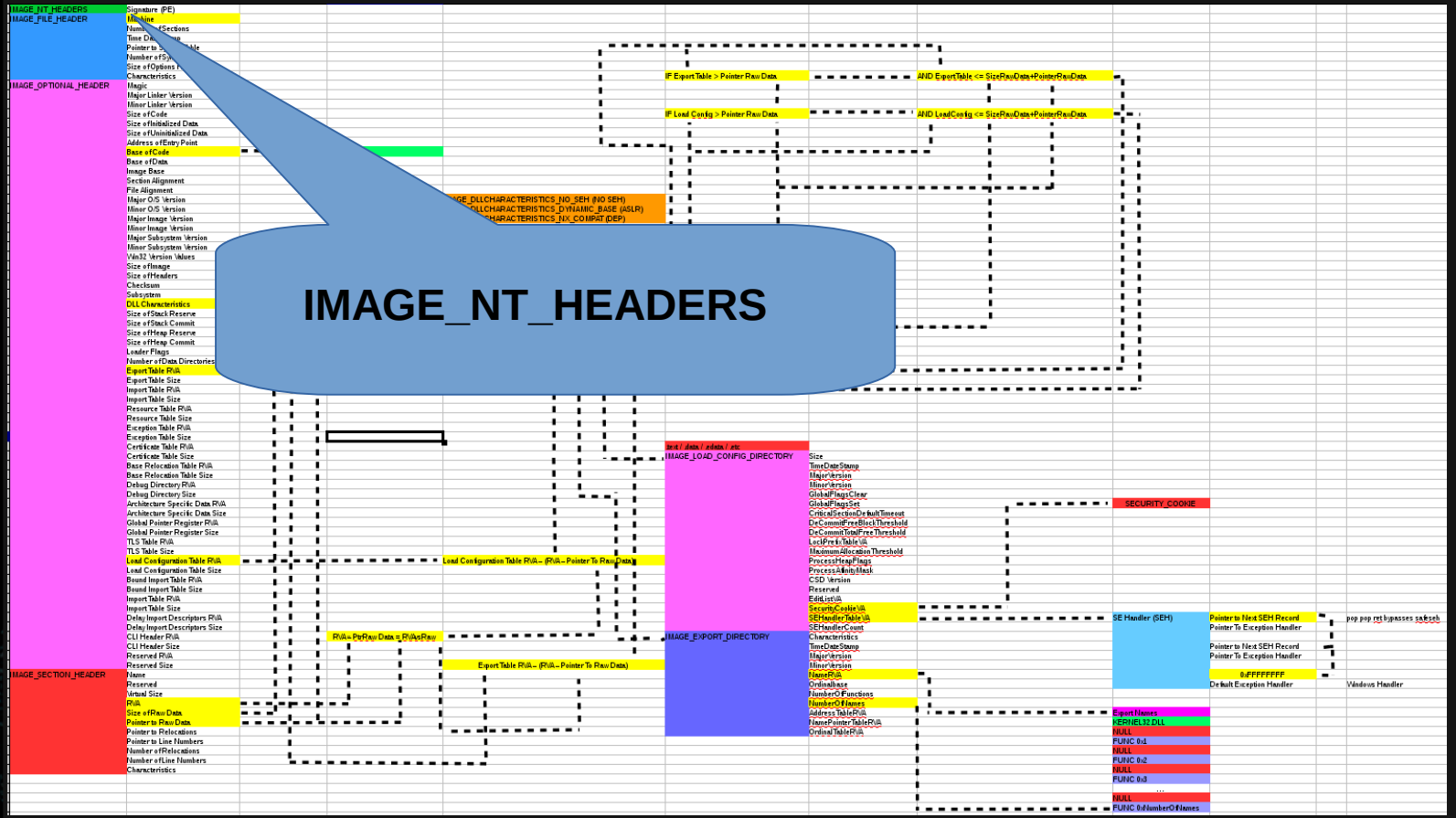


How you probably feel right now



Break it down!

PE Security Road Map



Forgetting the DOS Header

IMAGE_NT_HEADERS

IMAGE_FILE_HEADER

- Contains the generic information about the PE file
- Machine contains information on the architecture
- Number of Sections, .text, .data, .edata, etc.

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

IMAGE_NT_HEADERS

IMAGE_FILE_HEADER → Machine

- Example of the values that can be in the machine entry
- Checking these with bit masking is a good plan
- We are only concerned with x86 for this presentation

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	x86
IMAGE_FILE_MACHINE_IA64 0x0200	Intel Itanium
IMAGE_FILE_MACHINE_AMD64 0x8664	x64

IMAGE_NT_HEADERS

IMAGE_OPTIONAL_HEADER

- Contains information that pertains to security enumeration
- DllCharacteristics (ASLR, DEP, SEH)
- Address of Entry Point
- Reserve for the Heap and the Stack

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD                 Magic;
  BYTE                 MajorLinkerVersion;
  BYTE                 MinorLinkerVersion;
  DWORD                SizeOfCode;
  DWORD                SizeOfInitializedData;
  DWORD                SizeOfUninitializedData;
  DWORD                AddressOfEntryPoint;
  DWORD                BaseOfCode;
  DWORD                BaseOfData;
  DWORD                ImageBase;
  DWORD                SectionAlignment;
  DWORD                FileAlignment;
  WORD                 MajorOperatingSystemVersion;
  WORD                 MinorOperatingSystemVersion;
  WORD                 MajorImageVersion;
  WORD                 MinorImageVersion;
  WORD                 MajorSubsystemVersion;
  WORD                 MinorSubsystemVersion;
  DWORD                Win32VersionValue;
  DWORD                SizeOfImage;
  DWORD                SizeOfHeaders;
  DWORD                CheckSum;
  WORD                 Subsystem;
  WORD                 DllCharacteristics;
  DWORD                SizeOfStackReserve;
  DWORD                SizeOfStackCommit;
  DWORD                SizeOfHeapReserve;
  DWORD                SizeOfHeapCommit;
  DWORD                LoaderFlags;
  DWORD                NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

IMAGE_NT_HEADERS

IMAGE_OPTIONAL_HEADER → DllCharacteristics

- ASLR
- DEP/NX
- SEH

Value	Meaning
0x0001	Reserved.
0x0002	Reserved.
0x0004	Reserved.
0x0008	Reserved.
IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE 0x0040	The DLL can be relocated at load time.
IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY 0x0080	Code integrity checks are forced. If you set this flag and a section contains only uninitialized data, set the **PointerToRawData** member of IMAGE_SECTION_HEADER for that section to zero; otherwise, the image will fail to load because the digital signature cannot be verified.
IMAGE_DLLCHARACTERISTICS_NX_COMPAT 0x0100	The image is compatible with data execution prevention (DEP).
IMAGE_DLLCHARACTERISTICS_NO_ISOLATION 0x0200	The image is isolation aware, but should not be isolated.
IMAGE_DLLCHARACTERISTICS_NO_SEH 0x0400	The image does not use structured exception handling (SEH). No handlers can be called in this image.
IMAGE_DLLCHARACTERISTICS_NO_BIND 0x0800	Do not bind the image.
0x1000	Reserved.
IMAGE_DLLCHARACTERISTICS_WDM_DRIVER 0x2000	A WDM driver.
0x4000	Reserved.
IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE 0x8000	The image is terminal server aware.

IMAGE_NT_HEADERS

DllCharacteristics → The Code

- Bit masking
- Structs
- If/else logic

```
//Microsoft Sets these in Nibbles must use bitwise masking
if ((PEHeader.DllCharacteristics[0] & 0xF0) == '\x40'){
   printf("ASLR                        = Enabled\n");
}
else{
   printf("ASLR                        = Disabled\n");
}
if ((PEHeader.DllCharacteristics[1] & 0x0F)== '\x01'){
   printf("DEP                         = Enabled\n");
}
else{
   printf("DEP                         = Disabled\n");
}
if ((PEHeader.DllCharacteristics[1] & 0x0F)== '\x04'){
   printf("SEH                         = Disabled\n");
}
else{
   printf("SEH                         = Enabled\n");
}
```

IMAGE_NT_HEADERS

IMAGE_DATA_DIRECTORY (within optional header)

- Several of these stacked together create a list of offsets to different tables
- Using this we can find the IMAGE LOAD CONFIG DIRECTORY and the IMAGE EXPORT DIRECTORY

```
typedef struct _IMAGE_DATA_DIRECTORY {
  DWORD VirtualAddress;
  DWORD Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

IMAGE_SECTION_HEADER

IMAGE_SECTION_HEADER

- The number of these in the file are based on the number of sections that were talked about before
- VirtualAddress, SizeOfRawData, PointerToRawData

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

IMAGE_LOAD_CONFIG_DIRECTORY

- SecurityCookie
- SEHandlerTable
- SEHandlerCount

```
typedef struct {
    DWORD   Size;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   GlobalFlagsClear;
    DWORD   GlobalFlagsSet;
    DWORD   CriticalSectionDefaultTimeout;
    DWORD   DeCommitFreeBlockThreshold;
    DWORD   DeCommitTotalFreeThreshold;
    DWORD   LockPrefixTable;            // VA
    DWORD   MaximumAllocationSize;
    DWORD   VirtualMemoryThreshold;
    DWORD   ProcessHeapFlags;
    DWORD   ProcessAffinityMask;
    WORD    CSDVersion;
    WORD    Reserved1;
    DWORD   EditList;                   // VA
    DWORD   SecurityCookie;             // VA
    DWORD   SEHandlerTable;             // VA
    DWORD   SEHandlerCount;
} IMAGE_LOAD_CONFIG_DIRECTORY32, *PIMAGE_LOAD_CONFIG_DIRECTORY32;
```

IMAGE_LOAD_CONFIG_DIRECTORY → GetProcAddress()

Pay Dirt!

```
NAME
       GetProcAddress (KERNEL32.@)

SYNOPSIS
       FARPROC GetProcAddress
        (
        HMODULE hModule,
        LPCSTR  function
        )


DESCRIPTION
       Find the address of an exported symbol in a loaded dll.


PARAMS
       hModule [In]  Handle to the dll returned by LoadLibraryA(3w).

       function [In]  Name of the symbol, or an integer ordinal number < 16384.

RETURNS
       Success: A pointer to the symbol in the process address space.

       Failure: NULL. Use GetLastError(3w) to determine the cause.


IMPLEMENTATION
       Declared in "winbase.h".

       Implemented in "dlls/kernel32/module.c".

       Debug channel "module".
```

LoadLibrary(FileName)

IMAGE_LOAD_CONFIG_DIRECTORY → LoadLibrary()

C++

```
HMODULE WINAPI LoadLibrary(
  _In_ LPCTSTR lpFileName
);
```

Parameters

lpFileName [in]

The name of the module. This can be either a library module (a .dll file) or an executable module (an .exe file). The name specified is the file name of the module and is not related to the name stored in the library module itself, as specified by the **LIBRARY** keyword in the module-definition (.def) file.

If the string specifies a full path, the function searches only that path for the module.

If the string specifies a relative path or a module name without a path, the function uses a standard search strategy to find the module; for more information, see the Remarks.

If the function cannot find the module, the function fails. When specifying a path, be sure to use backslashes (\), not forward slashes (/). For more information about paths, see Naming a File or Directory.

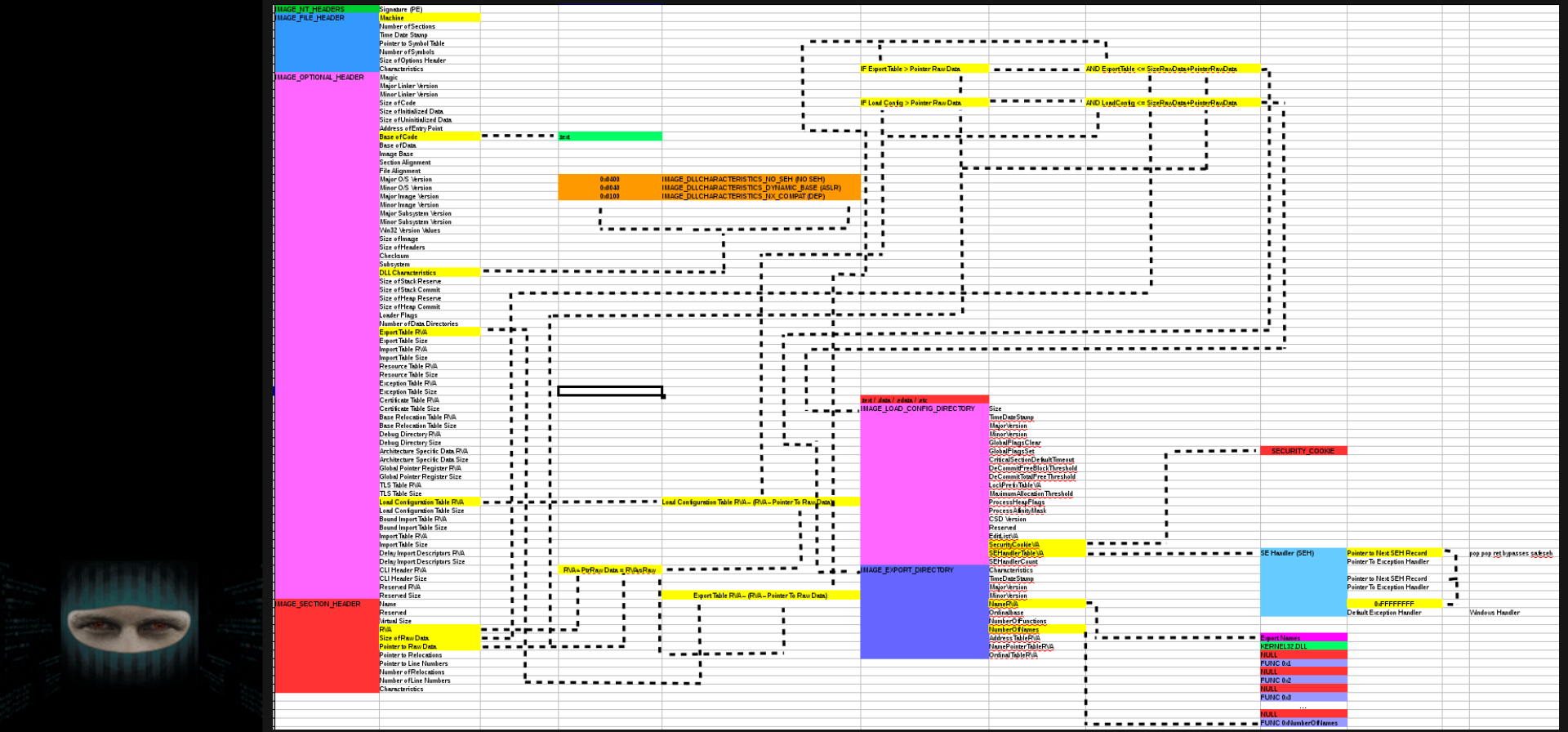
If the string specifies a module name without a path and the file name extension is omitted, the function appends the default library extension .dll to the module name. To prevent the function from appending .dll to the module name, include a trailing point character (.) in the module name string.

Return value

If the function succeeds, the return value is a handle to the module.

If the function fails, the return value is NULL. To get extended error information, call **GetLastError**.

Back to the PE Security Road Map




PUTTING IT ALL
TOGETHER
imgflip.com

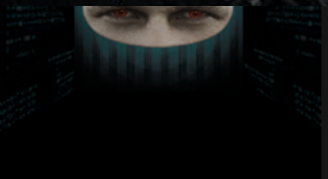
Enumerating DLL Function Calls

```
//Enumerate Functions Table / Virtual Addresses
if (argCheck(argv, "--enum-func", argc)){
        static int arrayRVAvsRAW[50];
        static int arraySectionOffsets[50];
        static int arraySizeOfRawData[50];
        static int arrayPointerToRawData[50];
        static int j;
        static unsigned char singleByte[1];
        static int nullCount = 0;
        static int prevNullPointer = 0;
        static int nextNullPointer = 0;
        static char functionName[MAX_PATH];
        static int funcCount = 0;
        static int funcNameInt = 0;
        bool first = true;
        inputFile = fopen(argv[argNum(argv, "--enum-func", argc)+1], "r");
        if (!inputFile){ fprintf(stderr, "ERROR: Unable to read file specified.\n"); return 1; }
        fread(&DOSHeader, 1, sizeof(DOSHeader)-1, inputFile);
        fseek(inputFile, DOSHeader.PEOffset, SEEK_SET);
        fread(&PEHeader, 1, sizeof(PEHeader)-1, inputFile);
        int i;
        for (i = 0; i <= PEHeader.NumberOfSections-1; i++){
                arraySectionOffsets[i] = DOSHeader.PEOffset + sizeof(PEHeader) + ((sizeof(dotHeader)-sizeof(dotHeader.pad))*i);
                fseek(inputFile, DOSHeader.PEOffset + sizeof(PEHeader) + ((sizeof(dotHeader)-sizeof(dotHeader.pad))*i), SEEK_SET);
                fread(&dotHeader, 1, sizeof(dotHeader)-1, inputFile);
                arrayRVAvsRAW[i] = (wordToInt(dotHeader.RVA) - wordToInt(dotHeader.PointerToRawData));
                arraySizeOfRawData[i] = wordToInt(dotHeader.SizeOfRawData);
                arrayPointerToRawData[i] = wordToInt(dotHeader.PointerToRawData);
                if (( (wordToInt(PEHeader.ExportTableRVA)-arrayRVAvsRAW[i]) < (arraySizeOfRawData[i]+arrayPointerToRawData[i]) )
                && ( (wordToInt(PEHeader.ExportTableRVA)-arrayRVAvsRAW[i]) >= arrayPointerToRawData[i] )) {
                        fseek(inputFile, (wordToInt(PEHeader.ExportTableRVA) - arrayRVAvsRAW[i]), SEEK_SET);
                        fread(&IMAGE_EXPORT_DIRECTORY, 1, sizeof(IMAGE_EXPORT_DIRECTORY)-1, inputFile);
                        printf("---BEGIN FUNCTION ENUMERATION---\n");
                        printf("Number Of Functions: %d\n", wordToInt(IMAGE_EXPORT_DIRECTORY.NumberOfNames));
                        printf("Export Names Offset: 0x%x\n", (wordToInt(IMAGE_EXPORT_DIRECTORY.NameRVA) - arrayRVAvsRAW[i]));
```

Enumerating DLL Function Calls

```
printf("Number Of Functions: %d\n", wordToInt(IMAGE_EXPORT_DIRECTORY.NumberOfNames));
printf("Export Names Offset: 0x%x\n", (wordToInt(IMAGE_EXPORT_DIRECTORY.NameRVA) - arrayRVAvsRAW[i]));
printf("Virtual Address:          Function Names:\n");
for (j = (wordToInt(IMAGE_EXPORT_DIRECTORY.NameRVA) - arrayRVAvsRAW[i]); j <= getFileSize(inputFile); j++){
        fseek(inputFile, j, SEEK_SET);
        fread(singleByte, sizeof(singleByte)+1, 1, inputFile);
        functionName[funcNameInt] = singleByte[0];
        funcNameInt++;
        if (singleByte[0] == '\x00'){
                if (first == false){
                        printAddress(argv[argNum(argv, "--enum-func", argc)+1], functionName);
                        printf("                %s\n", functionName);
                }
                first = false;
                funcNameInt = 0;
                funcCount++;
                if (funcCount == wordToInt(IMAGE_EXPORT_DIRECTORY.NumberOfNames)+1 ){
                        printf("--END FUNCTION ENUMERATION--\n");
                        break;;
                }
        }
}
}
}
fclose(inputFile);
return 0;
}
```

```
//Print Function Address
void printAddress(char fileName[MAX_PATH], char functionName[MAX_PATH]){
        printf("0x%x", GetProcAddress(LoadLibraryA(fileName), functionName));
}
```



Enumerating DEP, SEH, and ASLR

```
//Enumerate Protections
if (argCheck(argv, "--check-security", argc)){
        inputFile = fopen(argv[argNum(argv, "--check-security", argc)+1], "r");
        if (!inputFile){ fprintf(stderr, "ERROR: Unable to read file specified.\n"); return 1; }
        fread(&DOSHeader, 1, sizeof(DOSHeader)-1, inputFile);
        fseek(inputFile, DOSHeader.PEOffset, SEEK_SET);
        fread(&PEHeader, 1, sizeof(PEHeader)-1, inputFile);
        printf("---BEGIN SECURITY---\n");
        //Microsoft Sets these in Nibbles must use bitwise masking
        if ((PEHeader.DllCharacteristics[0] & 0xF0) == '\x40'){
                printf("ASLR                    = Enabled\n");
        }
        else{
                printf("ASLR                    = Disabled\n");
        }
        if ((PEHeader.DllCharacteristics[1] & 0x0F)== '\x01'){
                printf("DEP                     = Enabled\n");
        }
        else{
                printf("DEP                     = Disabled\n");
        }
        if ((PEHeader.DllCharacteristics[1] & 0x0F)== '\x04'){
                printf("SEH                     = Disabled\n");
        }
        else{
                printf("SEH                     = Enabled\n");
        }
```

Enumerating DEP, SEH, and ASLR

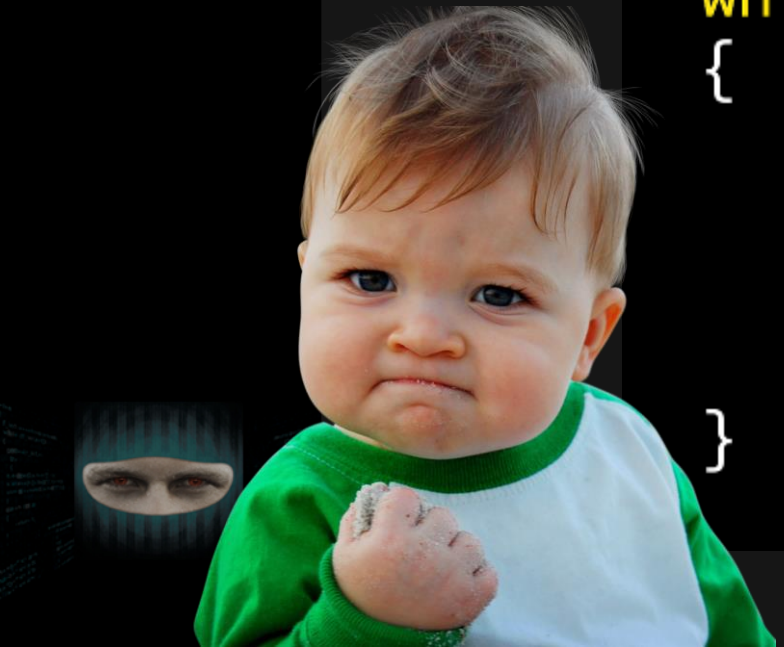
```
        //Get SEH and Security Cookies
        static int arrayRVAvsRAW[50];
        static int arraySectionOffsets[50];
        static int arraySizeOfRawData[50];
        static int arrayPointerToRawData[50];
        int i;
        for(i = 0; i <= PEHeader.NumberOfSections-1; i++){
                arraySectionOffsets[i] = DOSHeader.PEOffset + sizeof(PEHeader) + ((sizeof(dotHeader)-sizeof(dotHeader.pad))*i);
                fseek(inputFile, DOSHeader.PEOffset + sizeof(PEHeader) + ((sizeof(dotHeader)-sizeof(dotHeader.pad))*i), SEEK_SET);
                fread(&dotHeader, 1, sizeof(dotHeader)-1, inputFile);
                arrayRVAvsRAW[i] = (wordToInt(dotHeader.RVA) - wordToInt(dotHeader.PointerToRawData));
                arraySizeOfRawData[i] = wordToInt(dotHeader.SizeOfRawData);
                arrayPointerToRawData[i] = wordToInt(dotHeader.PointerToRawData);
                if (( (wordToInt(PEHeader.LoadConfigurationTableRVA)-arrayRVAvsRAW[i]) < (arraySizeOfRawData[i]+arrayPointerToRawData[i]) )
                && ( (wordToInt(PEHeader.LoadConfigurationTableRVA)-arrayRVAvsRAW[i]) >= arrayPointerToRawData[i] )) {
                        fseek(inputFile, (wordToInt(PEHeader.LoadConfigurationTableRVA) - arrayRVAvsRAW[i]), SEEK_SET);
                        fread(&IMAGE_LOAD_CONFIG_DIRECTORY, 1, sizeof(IMAGE_LOAD_CONFIG_DIRECTORY)-1, inputFile);
                        printf("LOAD_CONFIGURATION_TABLE = 0x%x\n", (wordToInt(PEHeader.LoadConfigurationTableRVA) - arrayRVAvsRAW[i]) );
                        printHexLE("Security Cookie VA       = ",IMAGE_LOAD_CONFIG_DIRECTORY.SecurityCookieVA, sizeof(IMAGE_LOAD_CONFIG_DIRECTORY.SecurityCookieVA));
                        printHexLE("SEH Table VA             = ", IMAGE_LOAD_CONFIG_DIRECTORY.SEHandlerTableVA, sizeof(IMAGE_LOAD_CONFIG_DIRECTORY.SEHandlerTableVA));
                        printHexLE("SEH Count                = ", IMAGE_LOAD_CONFIG_DIRECTORY.SEHandlerCount, sizeof(IMAGE_LOAD_CONFIG_DIRECTORY.SEHandlerCount));
                }
        }
        printf("---END SECURITY---\n");
        fclose(inputFile);
        return 0;
}
```

ASLR Entropy Algorithm

Algorithm of Success

```
while(noSuccess)
{
        tryAgain();

        if(Dead)
                break;
}
```



ASLR Entropy Algorithm

Loading and Unloading

Bit Masking

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

int main(){
        static int addr;
        static HINSTANCE dllHandle;
        static unsigned char c[4];
        static unsigned char d[4];
        static int l;
        static int n[8] = {0, 0, 0, 0, 0, 0, 0, 0};
        static int iter = 0;
        for(iter = 0; iter <= 1000; iter++){
                dllHandle = LoadLibraryA("aeinv.dll");
                addr = (int)GetProcAddress(dllHandle, "CollectMatchingInfo");
                FreeLibrary(dllHandle);
                c[0] = (addr >> 24) & 0xFF;
                c[1] = (addr >> 16) & 0xFF;
                c[2] = (addr >> 8) & 0xFF;
                c[3] = addr & 0xFF;

                dllHandle = LoadLibraryA("aeinv.dll");
                addr = (int)GetProcAddress(dllHandle, "CollectMatchingInfo");
                FreeLibrary(dllHandle);
                d[0] = (addr >> 24) & 0xFF;
                d[1] = (addr >> 16) & 0xFF;
                d[2] = (addr >> 8) & 0xFF;
                d[3] = addr & 0xFF;

                static int i = 0;
                if ((c[0] & 0xF0) != (d[0] & 0xF0)){
                        n[0] = 1;
                }
                if ((c[0] & 0x0F) != (d[0] & 0x0F)){
                        n[1] = 1;
                }
                //-------
                if ((c[1] & 0xF0) != (d[1] & 0xF0)){
                        n[2] = 1;
                }
                if ((c[1] & 0x0F) != (d[1] & 0x0F)){
                        n[3] = 1;
                }
                //---------
                if ((c[2] & 0xF0) != (d[2] & 0xF0)){
                        n[4] = 1;
                }
                if ((c[2] & 0x0F) != (d[2] & 0x0F)){
                        n[5] = 1;
                }
```

ASLR Entropy Algorithm

Print Flipped Nibbles

Calculate Entropy

```
                //--------
                if ((c[3] & 0xF0) != (d[3] & 0xF0)){
                        n[6] = 1;
                }
                if ((c[3] & 0x0F) != (d[3] & 0x0F)){
                        n[7] = 1;
                }
        }

        static int k = 0;
        printf("Flipped Nibbles: ");
        for (k = 0; k < 8; k++){
                        if(k ==0){l=0;}
                        printf("%d", n[k]);
                        if (n[k] == 1){
                                l++;
                        }
        }
        printf("\n");
        printf("Entropy: %dbit\n", l*4);
        return 0;
}
```

ASLR Entropy Algorithm

Setting the Limitations

- Currently only works on libraries that aren't currently loaded into memory as kernel32.dll and user32.dll only change addresses upon reboot as they are loaded into memory on boot.
- Only x86 at this time
- Use as much itterations as you like however don't let your computer get hot enough to catch fire or fry eggs (this totally didn't happen to me)
- Suggestions are welcome after the talk

Badger Demo

Exploitation Knowledge Base

GCC DEP/NX and SSP Protections Overview

- Canaries
  - Smashing Stack Protection (SSP)
  - --fno-stack-protector disables the feature
  - Default since GCC 4.1
- DEP/NX
  - Data Execution Prevention
  - Non-Executable Stack
  - -z execstac disables the feature
  - Default since GCC 4.1
- ASLR
  - Address Space Layout Randomization
  - Kernel Level

Exploitation Knowledge Base

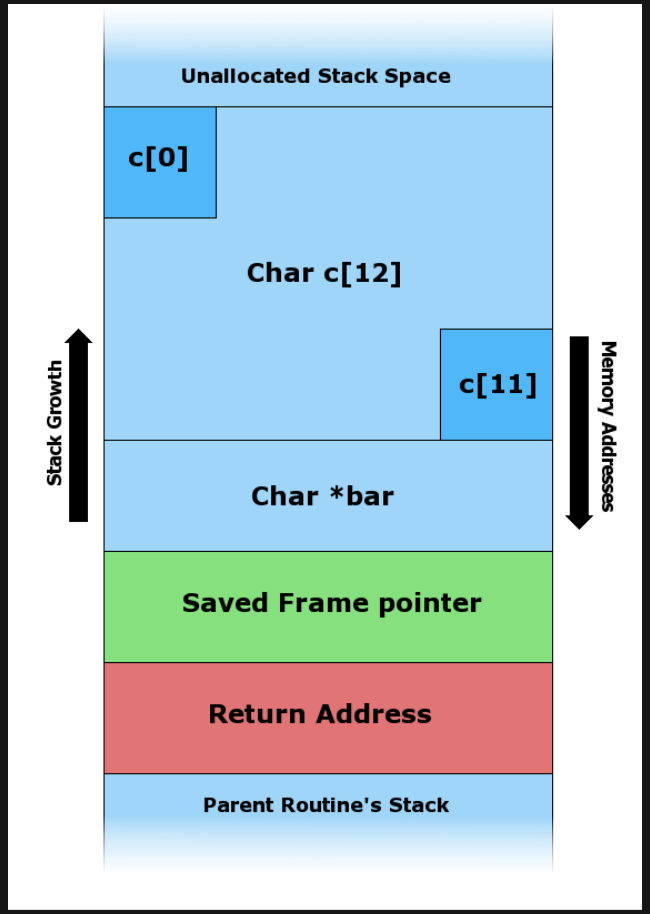
Smashing the Stack

- Used to overwrite eip/rip
- Avoid null bytes for code execution
- Happens when a buffer receives too much data and proper error checking isn't present
- Allows an attacker to obtain code execution or remote code execution
- Can be used for privledge escalation

Exploitation Knowledge Base

Smashing the Stack

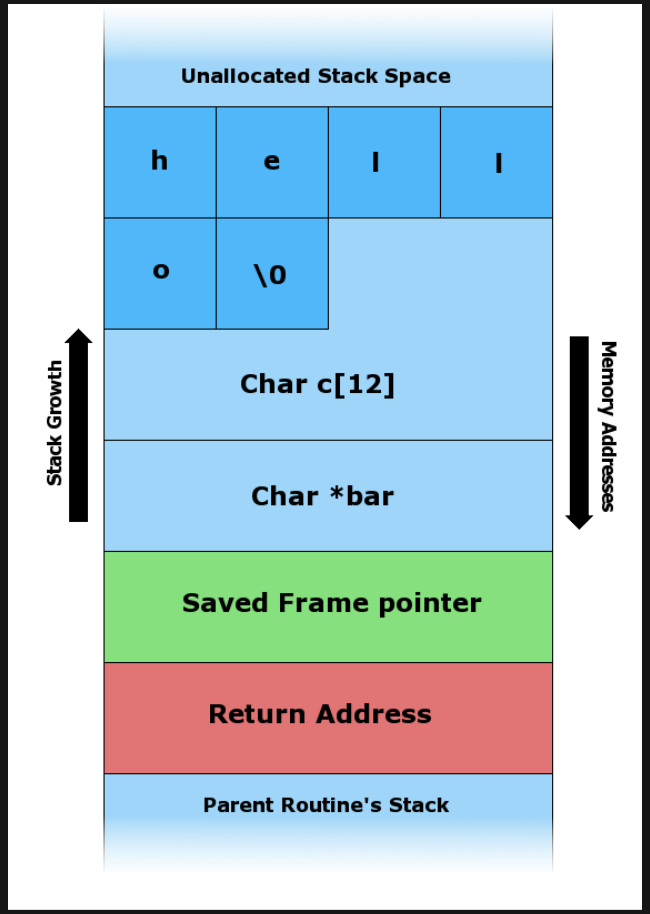
- Buffer starts at c[0]
- Buffer ends at c[11]
- Pointer to char *bar
- Saved Frame Pointer (ebp)
- Return Address (eip)
- Step through the process



Exploitation Knowledge Base

Smashing the Stack

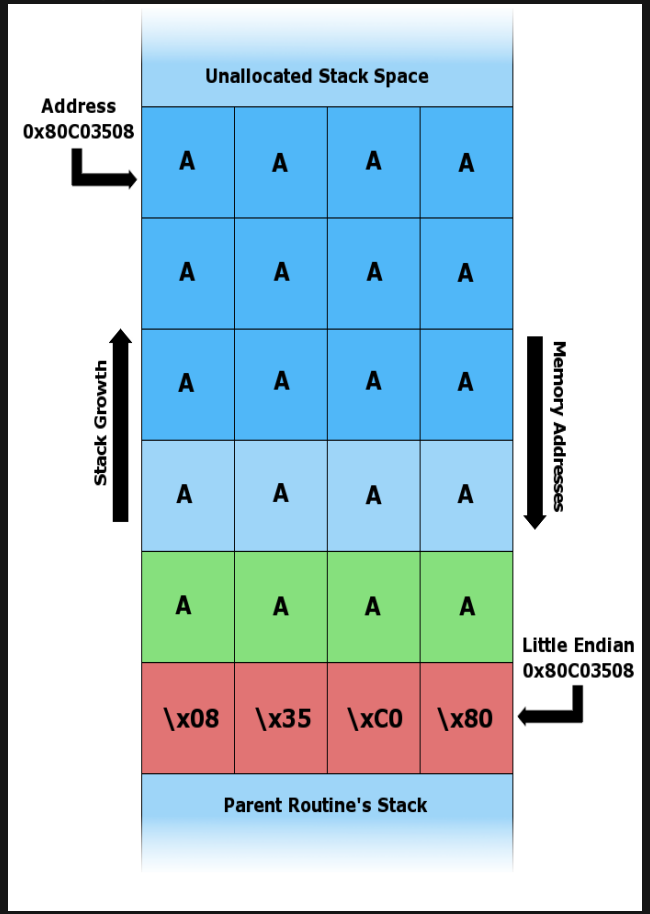
- Normal buffer
- '\x00' / null / terminator
- Return Address (eip) OK
- Normal execution



Exploitation Knowledge Base

Smashing the Stack

- Control User Input
- Enter too much data
- Check for security controls
- Find offset of eip/rip
- Addresses stored in memory are in Little Endian format
- Point to your code



Exploitation Knowledge Base

Smashing The Stack → Example Code

- No error checking
- Argv[1] moved into buffer with no check if size is over 256 bytes
- Vulnerable to overflow

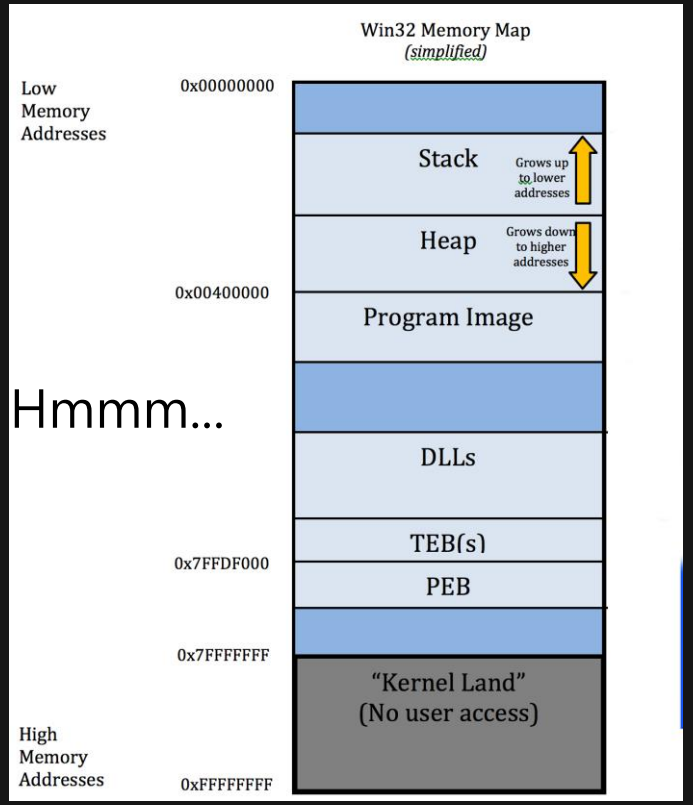
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
        char buffer[256];
        strcpy(buffer, argv[1]);
        printf("%s\n", buffer);
        return 0;
}
```

Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP

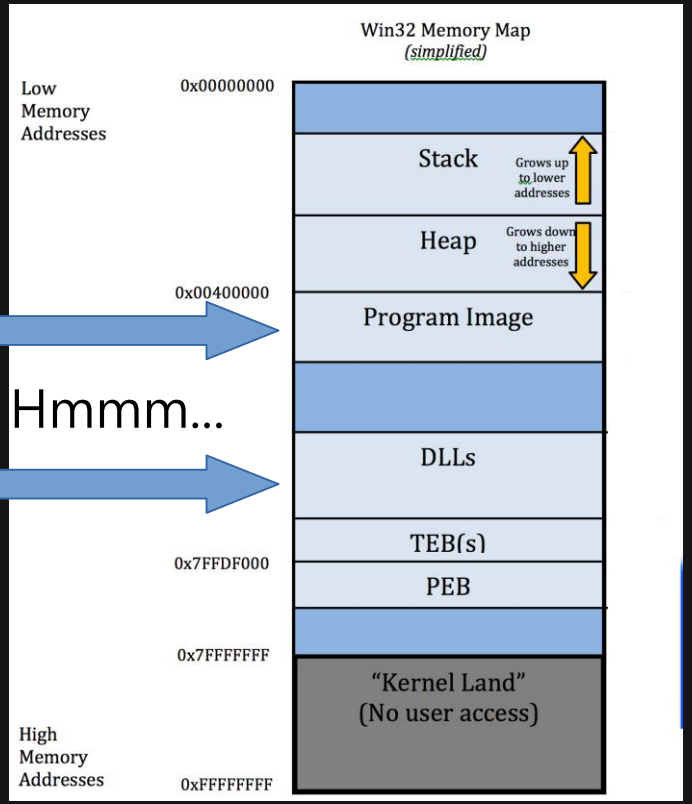
- Since DEP (Data Execution Prevention) makes certain parts of memory NX how can we bypass this?
- Feel free to shout your answers to me!



Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP

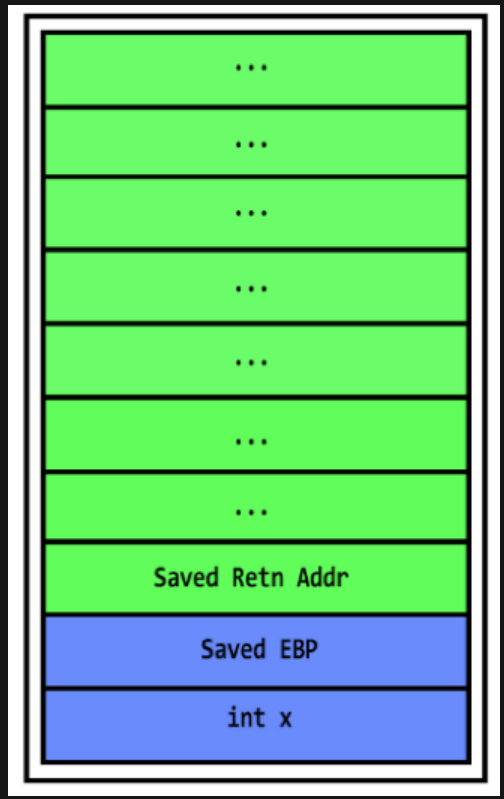
- DLLs → (why can we use this?)
- Why can we use the Program Image?
- What instructions are useful to us?
- What technique is it called?



Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP with ROP

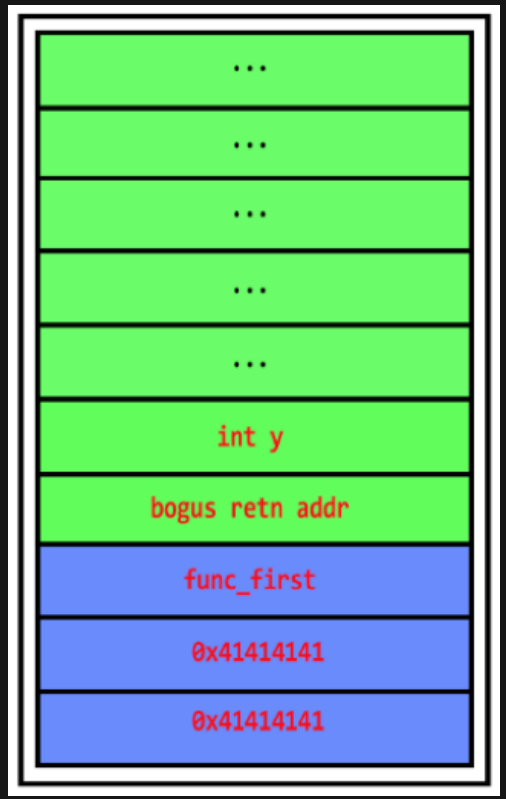
- Before the overflow



Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP with ROP

- After the overflow
- In this case we used a bogus return address
- '\x41' is = 'A'
- How do we chain this together?



Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP with ROP

- We can chain these together using pop-ret or pop-pop-ret or any combination of pop-ret
- We use these pop-ret sections from parts of the memory space that is marked executable
- These little pieces of code are called ROP Gadgets

Exploitation Knowledge Base

Smashing The Stack → Bypassing DEP with ROP

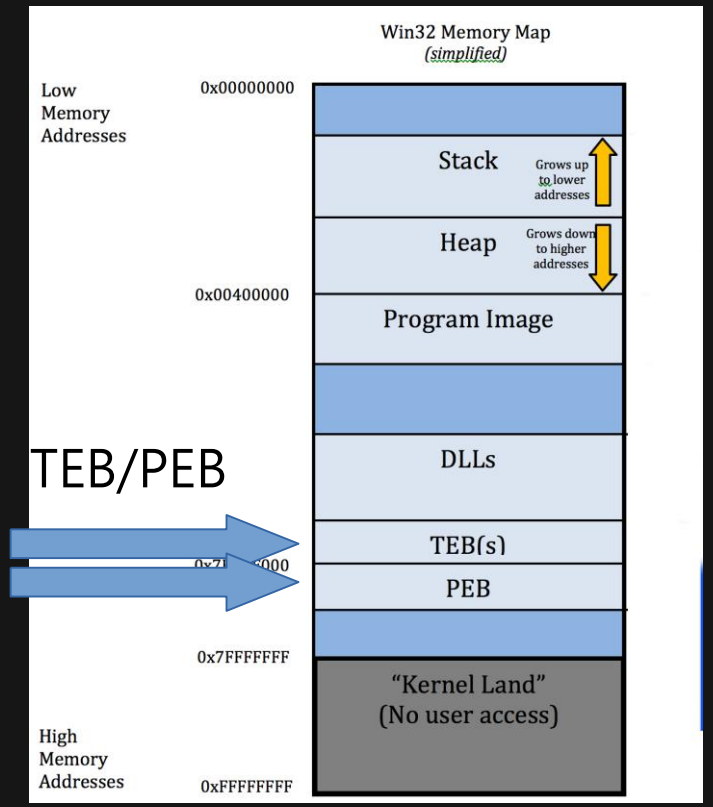
- The code to jmp esp works as well if DEP is only enabled for Windows Services or a library has protection disabled.
- Code: jmp esp = '\xff\xe4'
- Code: pop esp; ret; = '\x5c\xc3'
- Same idea however not chaining multiple gadgets


IT WAS SAD TIMES
FOR DEP
imgflip.com

Exploitation Knowledge Base

What is TEBs and PEB?

- TEB – Thread Environment Block
- PEB – Process Environment Block
- Let's go over what these blocks contain as well



TEB and PEB Overview

What is TEB and PEB, how do I access them?

- **This isn't required knowledge**
- **Since it's part of memory space we will briefly touch on the subject**

Accessing TEB

What is TEB, how do I access it?

- **TEB is simply a data structure that hold information about the current thread.**
- **Here is an example of how to get the pointer to TIB**
- **Let's have a look at what TIB contains**

```c
// Microsoft C
void *getTIB() {
    void *pTIB;
    __asm {
        mov EAX, FS:[0x18]
        mov pTIB, EAX
    }
    return pTIB;
}
```

Accessing TEB

Contents of the TIB (32-bit Windows) [edit]

Position	Length	Windows Versions	Description
FS:[0x00]	4	Win9x and NT	Current Structured Exception Handling (SEH) frame
FS:[0x04]	4	Win9x and NT	Stack Base / Bottom of stack (high address)
FS:[0x08]	4	Win9x and NT	Stack Limit / Ceiling of stack (low address)
FS:[0x0C]	4	NT	SubSystemTib
FS:[0x10]	4	NT	Fiber data
FS:[0x14]	4	Win9x and NT	Arbitrary data slot
FS:[0x18]	4	Win9x and NT	Linear address of TIB
---- End of NT subsystem independent part ----			
FS:[0x1C]	4	NT	Environment Pointer
FS:[0x20]	4	NT	Process ID (in some windows distributions this field is used as 'DebugContext')
FS:[0x24]	4	NT	Current thread ID
FS:[0x28]	4	NT	Active RPC Handle
FS:[0x2C]	4	Win9x and NT	Linear address of the thread-local storage array
FS:[0x30]	4	NT	Linear address of Process Environment Block (PEB)
FS:[0x34]	4	NT	Last error number
FS:[0x38]	4	NT	Count of owned critical sections
FS:[0x3C]	4	NT	Address of CSR Client Thread
FS:[0x40]	4	NT	Win32 Thread Information
FS:[0x44]	124	NT, Wine	Win32 client information (NT), user32 private data (Wine), 0x60 = LastError (Win95), 0x74 = LastError (WinME)
FS:[0xC0]	4	NT	Reserved for Wow64. Contains a pointer to FastSysCall in Wow64.
FS:[0xC4]	4	NT	Current Locale
FS:[0xC8]	4	NT	FP Software Status Register
FS:[0xCC]	216	NT, Wine	Reserved for OS (NT), kernel32 private data (Wine) herein: FS:[0x124] 4 NT Pointer to KTHREAD (ETHREAD) structure
FS:[0x1A4]	4	NT	Exception code
FS:[0x1A8]	18	NT	Activation context stack

Accessing TEB

FS:[0x1BC]	24	NT, Wine	Spare bytes (NT), ntdll private data (Wine)
FS:[0x1D4]	40	NT, Wine	Reserved for OS (NT), ntdll private data (Wine)
FS:[0x1FC]	1248	NT, Wine	GDI TEB Batch (OS), vm86 private data (Wine)
FS:[0x6DC]	4	NT	GDI Region
FS:[0x6E0]	4	NT	GDI Pen
FS:[0x6E4]	4	NT	GDI Brush
FS:[0x6E8]	4	NT	Real Process ID
FS:[0x6EC]	4	NT	Real Thread ID
FS:[0x6F0]	4	NT	GDI cached process handle
FS:[0x6F4]	4	NT	GDI client process ID (PID)
FS:[0x6F8]	4	NT	GDI client thread ID (TID)
FS:[0x6FC]	4	NT	GDI thread locale information
FS:[0x700]	20	NT	Reserved for user application
FS:[0x714]	1248	NT	Reserved for GL
FS:[0xBF4]	4	NT	Last Status Value
FS:[0xBF8]	532	NT	Static UNICODE_STRING buffer
FS:[0xE0C]	4	NT	Pointer to deallocation stack
FS:[0xE10]	256	NT	TLS slots, 4 byte per slot
FS:[0xF10]	8	NT	TLS links (LIST_ENTRY structure)
FS:[0xF18]	4	NT	VDM
FS:[0xF1C]	4	NT	Reserved for RPC
FS:[0xF28]	4	NT	Thread error mode (RtlSetThreadErrorMode)

Accessing PEB

What is PEB, how do I access it?

- **PEB – is a data structure that is opaque. It's used internally by the Windows Operating System itself**
- **Handles Mutual Exclusion**
- **Close to EPROCESS or Kernel Space**
- **Pointer located inside TEB**

Accessing PEB

Fields from a PEB that are initialized from kernel global variables[3]

Field	is initialized from	overridable by PE information?
`NumberOfProcessors`	`KeNumberOfProcessors`	No
`NtGlobalFlag`	`NtGlobalFlag`	No
`CriticalSectionTimeout`	`MmCriticalSectionTimeout`	No
`HeapSegmentReserve`	`MmHeapSegmentReserve`	No
`HeapSegmentCommit`	`MmHeapSegmentCommit`	No
`HeapDeCommitTotalFreeThreshold`	`MmHeapDeCommitTotalFreeThreshold`	No
`HeapDeCommitFreeBlockThreshold`	`MmHeapDeCommitFreeBlockThreshold`	No
`MinimumStackCommit`	`MmMinimumStackCommitInBytes`	No
`ImageProcessAffinityMask`	`KeActiveProcessors`	`ImageLoadConfigDirectory.ProcessAffinityMask`
`OSMajorVersion`	`NtMajorVersion`	`OptionalHeader.Win32VersionValue & 0xFF`
`OSMinorVersion`	`NtMinorVersion`	`(OptionalHeader.Win32VersionValue >> 8) & 0xFF`
`OSBuildNumber`	`NtBuildNumber & 0x3FFF` combined with `CmNtCSDVersion`	`(OptionalHeader.Win32VersionValue >> 16) & 0x3FFF` combined with `ImageLoadConfigDirectory.CmNtCSDVersion`
`OSPlatformId`	`VER_PLATFORM_WIN32_NT`	`(OptionalHeader.Win32VersionValue >> 30) ^ 0x2`

Make Way for the Shellcode

Making Space for your Shellcode

Make Way!

- **VirtualAlloc(MEM_COMMIT + PAGE READWRITE EXECUTE) + copy memory**
  - **Allows creation of new executable memory region, now copy your shellcode to it, and execute**
- **HeapCreate(HEAP_CREATE_ENABLE_EXECUTE) + HeapAlloc() + copy memory**
  - **A very similar technique to VirtualAlloc()**

Making Space for your Shellcode

Make Way!

- **SetProcessDEPPolicy()**
  - **Changes DEP policy for the current process (Vista SP1, XP SP3, Server 2008, and only when DEP Policy is set to OptIn or OptOut)**
- **NtSetInformationProcess()**
  - **Changes the DEP policy for the current process**

Making Space for your Shellcode

Make Way!

- **VirtualProtect(PAGE_READ_WRITE_EXECUTE)**
  - **Change the access protection level to executable of a given memory page.**
- **WriteProcessMemory(). Copies shellcode to another executable location, jump to it and execute. (Must be a writable executable)**

Choose your Weapon

MAC

Windows

Linux

Choose your Weapon

API / OS	XP SP2	XP SP3	Vista SP0	Vista SP1	Windows 7	Windows 2003 SP1	Windows 2008
VirtualAlloc	yes	yes	yes	yes	yes	yes	yes
HeapCreate	yes	yes	yes	yes	yes	yes	yes
SetProcessDEPPolicy	no (1)	yes	no (1)	yes	no (2)	no (1)	yes
NtSetInformationProcess	yes	yes	yes	no (2)	no (2)	yes	no (2)
VirtualProtect	yes	yes	yes	yes	yes	yes	yes
WriteProcessMemory	yes	yes	yes	yes	yes	yes	yes

(1) = doesn't exist

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

VirtualProtect() Overview

Starting Address Pointer

Size of Shellcode

Protection Options

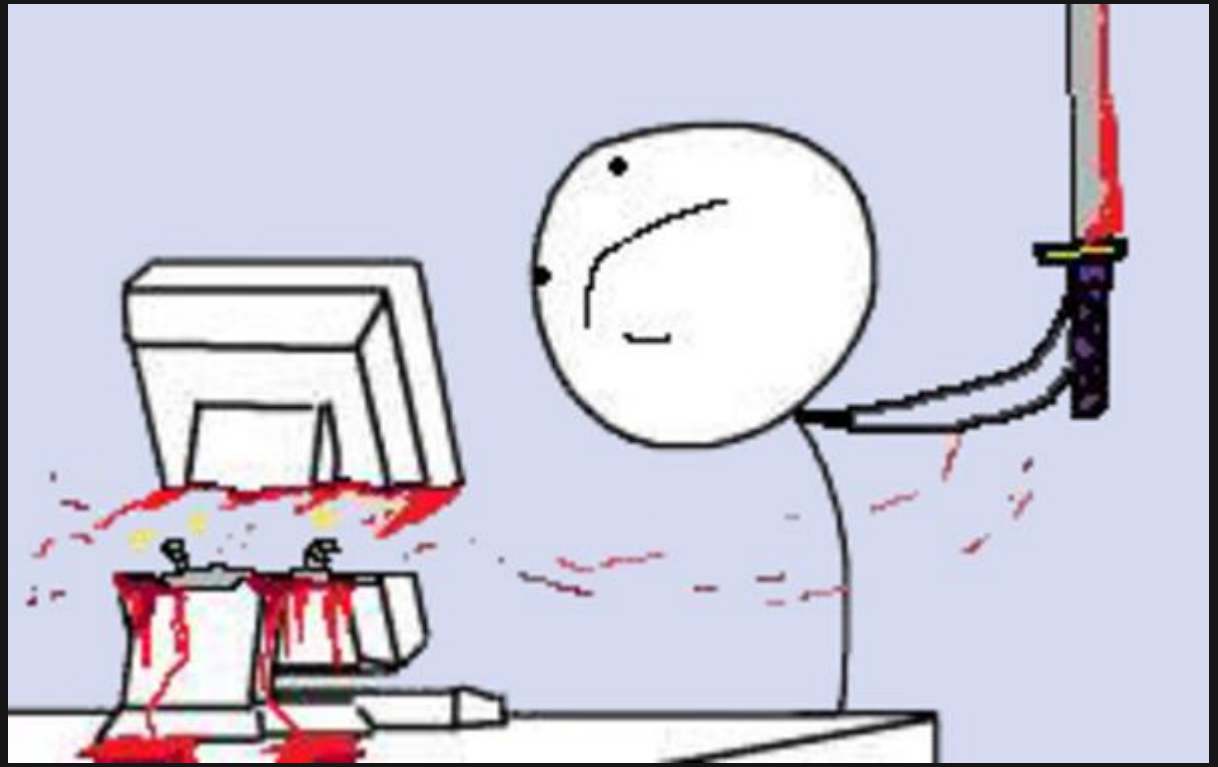
```
BOOL WINAPI VirtualProtect(
  _In_  LPVOID lpAddress,
  _In_  SIZE_T dwSize,
  _In_  DWORD  flNewProtect,
  _Out_ PDWORD lpflOldProtect
);
```

A Place to Save your Settings

***A Writable Memory Location**

PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages. **Windows Server 2003 and Windows XP:** This attribute is not supported by the **CreateFileMapping** function until Windows XP with SP2 and Windows Server 2003 with SP1.

ROP Demo



Chameleon Demo

Questions?