

Intro to Windows Exploit Techniques for Linux PWNers

Atum

About me

- Atum
 - @blue-lotus
 - @Tea Deliverers
 - @Peking University
- Keywords
 - Software Security, System Security
 - CTF PWNer, Weak Chicken
- lgcpku@gmail.com

Outline

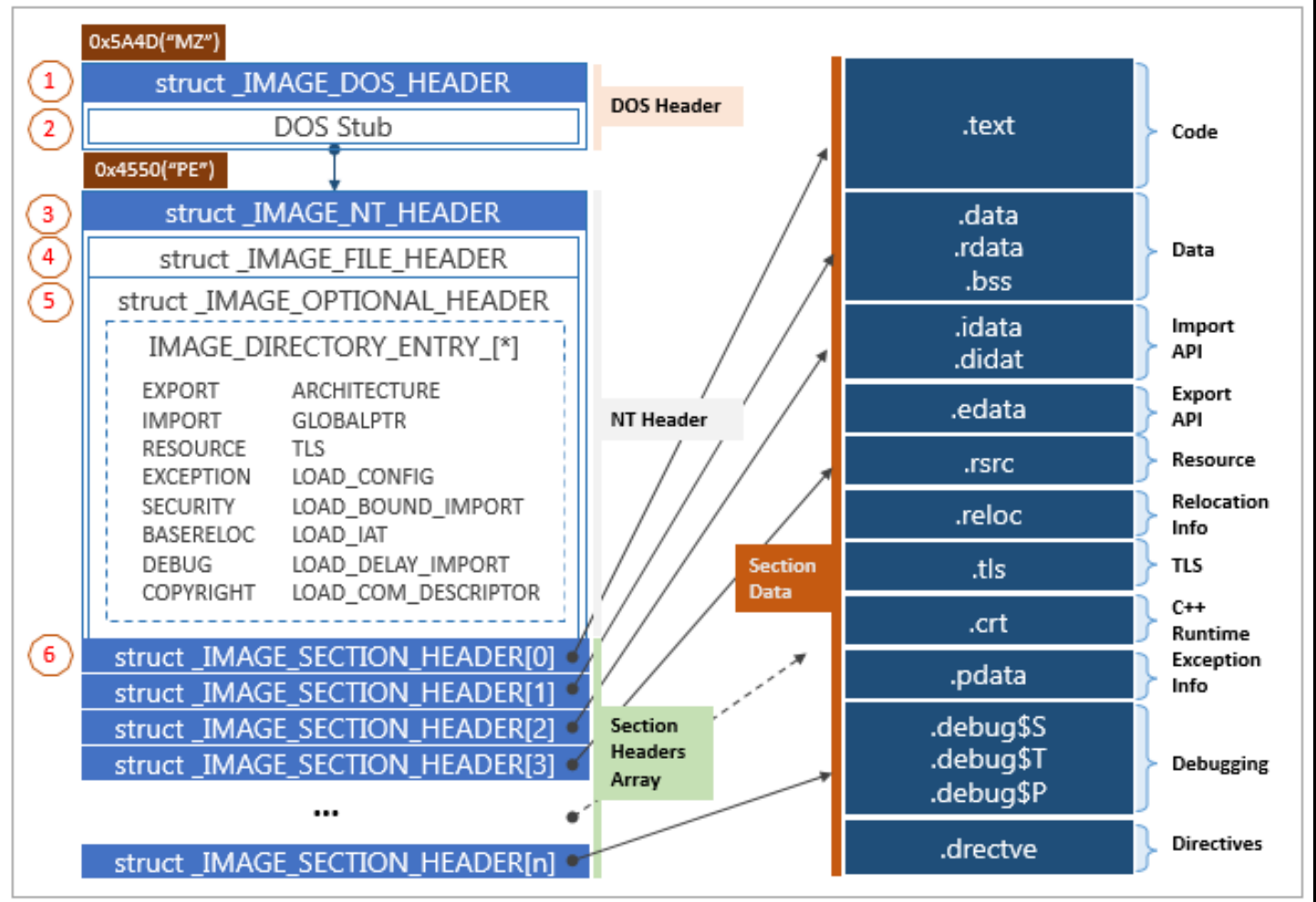
- Basics
- Windows Exploit Toolchains
- Windows Security Mitigations
- Linux Exploit Techniques on Windows
- Windows-only Exploit Techniques

Basics

PE/COFF FILE Format

- DOS Header
 - MZ signature
- PE FILE HEADER
 - EntryPoint
 - DataDirectory
- Section Table
 - Table of Section Headers

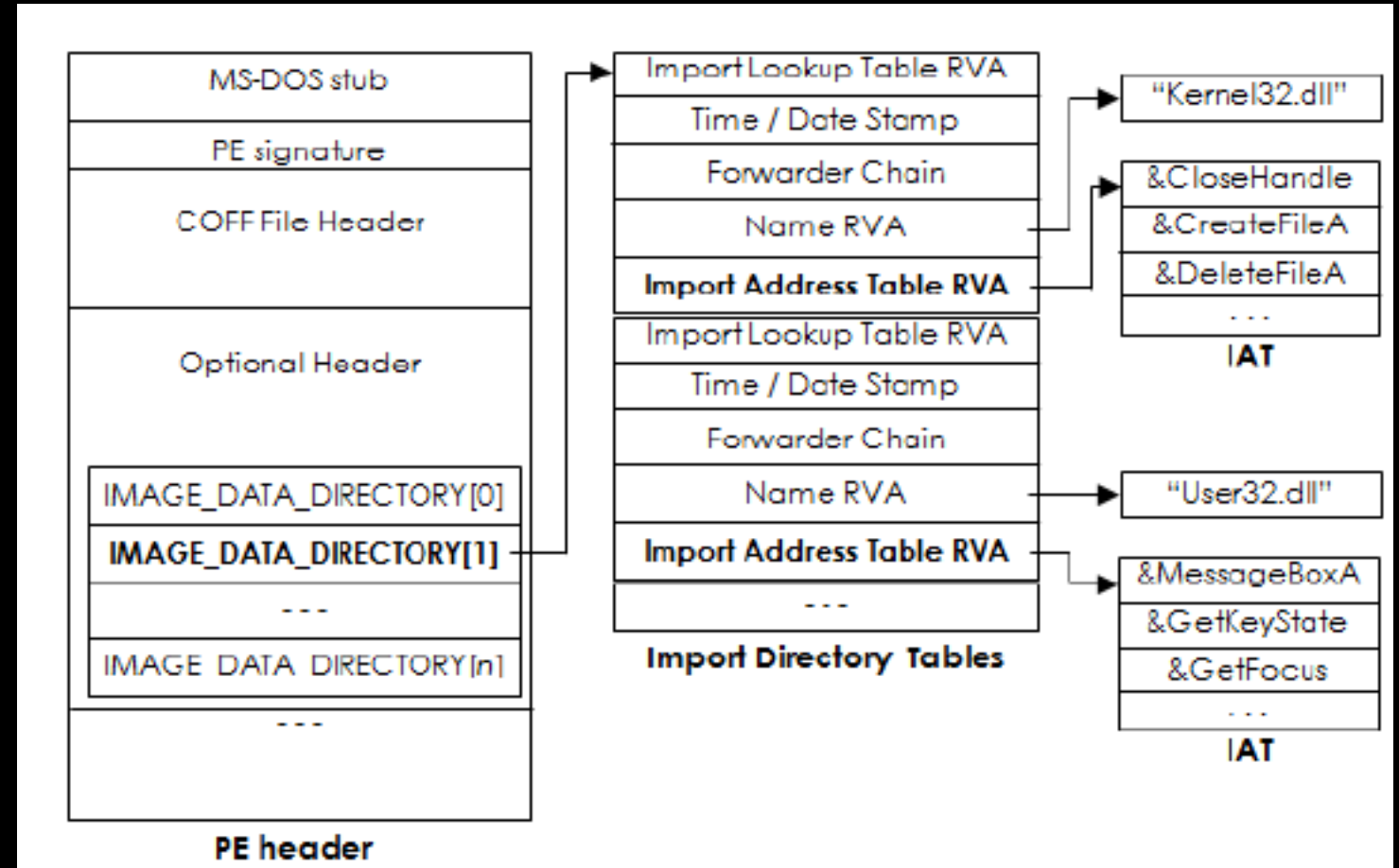
PE Format



Basics

PE/COFF FILE Format

- Import Address Table
 - Similar as ELF GOT
 - **Read Only**
- Export Address Table
 - Exported functions of a Module
 - **Read Only**



Basics

Important DLLs

- ntdll.dll
 - Interface of userspace and kernel
 - exports the Windows Native API
 - Reside in write-protected page; shared base among processes
- kernel32.dll
 - Imports ntdll.dll
 - exports the Windows API
 - Reside in write-protected page; shared base among processes
- msvcrtxxx.dll/vcruntimexxx.dll&ucrtbase.dll
 - Microsoft C runtime library(similar to glibc)

Windows Exploit Toolchains

General Tools

- Cygwin
 - A bash environment on Windows
- socket&&popen
 - Interact with challenges
- Process Hacker
 - An enhanced version of tasklist
- Visual Studio
 - Developer Command Prompt

Windows Exploit Toolchains

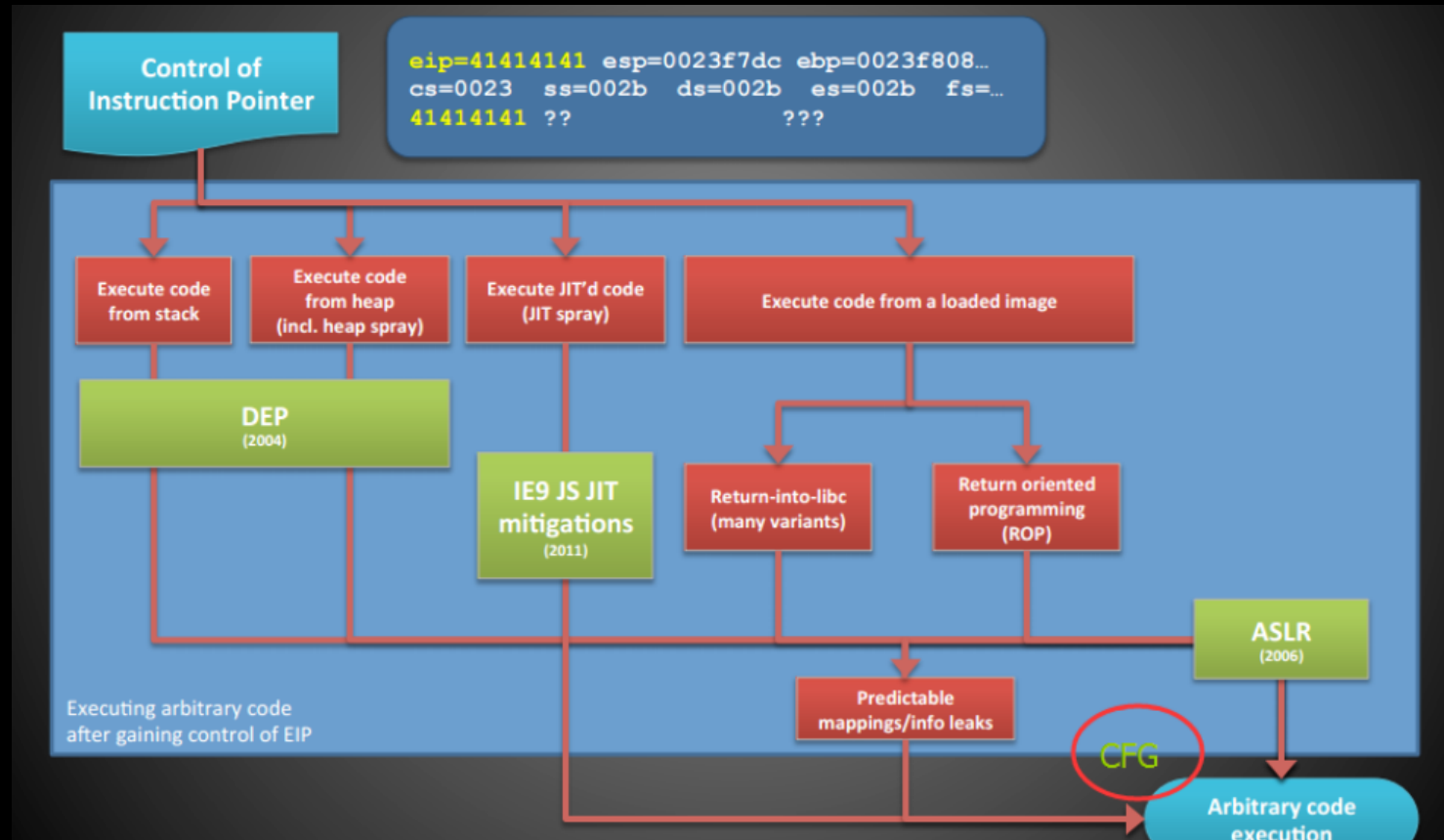
Debuggers

- Windbg
 - Recommend, very powerful
- IDA Pro Debugger
 - A debugger front-end, support multi-backend debugger such as gdb, windbg.
- Ollydbg
 - Easy & powerful, but cannot debug x64 program
- X64dbg
 - Similar to ollydbg, can debug x64 program, but short of plugins

Windows Security Mitigations

General Exploit Mitigations

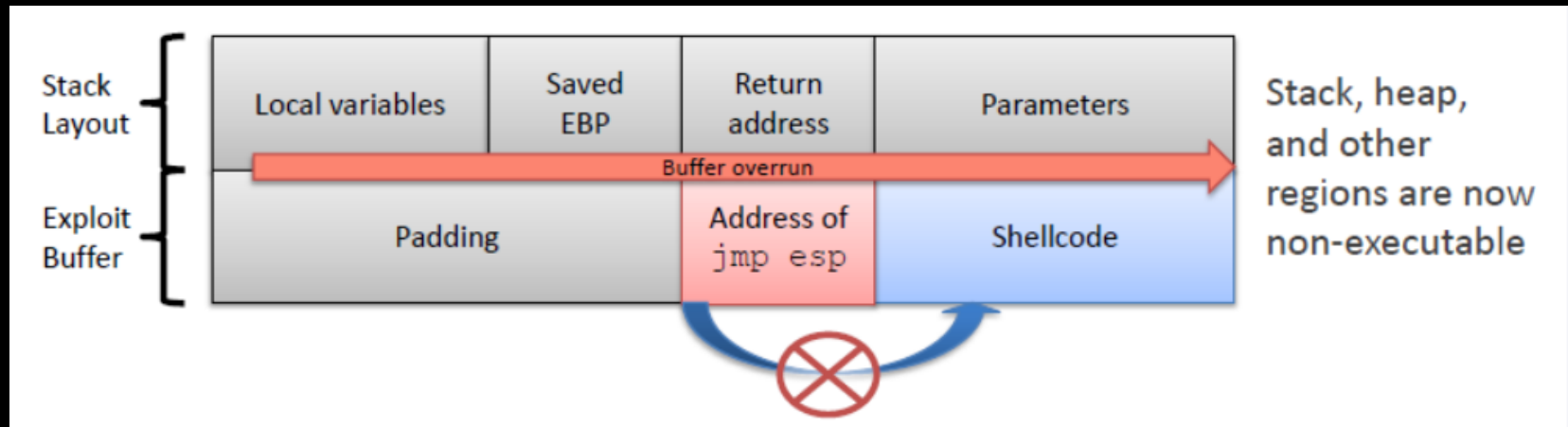
- DEP
- ASLR
- CFG



Windows Security Mitigations

DEP

- NX on Linux
- Bypassed by
 - ROP
 - JIT page, VirtualProtect etc.

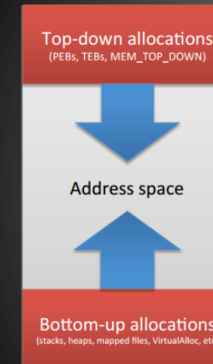


Windows Security Mitigations

ASLR

- Slightly different from PIE&ASLR on Linux
 - Image randomization base changed every time system booted
 - TEB/PEB/heap/stack randomization base changed every time process start
 - Some kernel related dlls (such as ntdll.dll kernel32.dll) share base among all processes
- Bypassed by
 - Info leak(cross process is OK)
 - brute-force (win7 x64, win10 x86)
 - Attack Non-ASLR images or top down alloc(win7)

Bottom-up & top-down randomization



Windows 7

- Heaps and stacks are randomized
- PEBs/TEBs are randomized, but with limited entropy
- VirtualAlloc and MapViewOfFile are not randomized
- Predictable memory regions can exist as a result

Windows 8

- All bottom-up/top-down allocations are randomized
- Accomplished by biasing start address of allocations
- PEBs/TEBs now receive much more entropy
- Both are opt-in (EXE must be dynamicbase)

ASLR entropy improvements

Entropy (in bits) by region	Windows 7		Windows 8		
	32-bit	64-bit	32-bit	64-bit	64-bit (HE)
Bottom-up allocations (opt-in)	0	0	8	8	24
Stacks	14	14	17	17	33
Heaps	5	5	8	8	24
Top-down allocations (opt-in)	0	0	8	17	17
PEBs/TEBs	4	4	8	17	17
EXE images	8	8	8	17*	17*
DLL images	8	8	8	19*	19*
Non-ASLR DLL images (opt-in)	0	0	8	8	24

* 64-bit DLLs based below 4GB receive 14 bits, EXEs below 4GB receive 8 bits

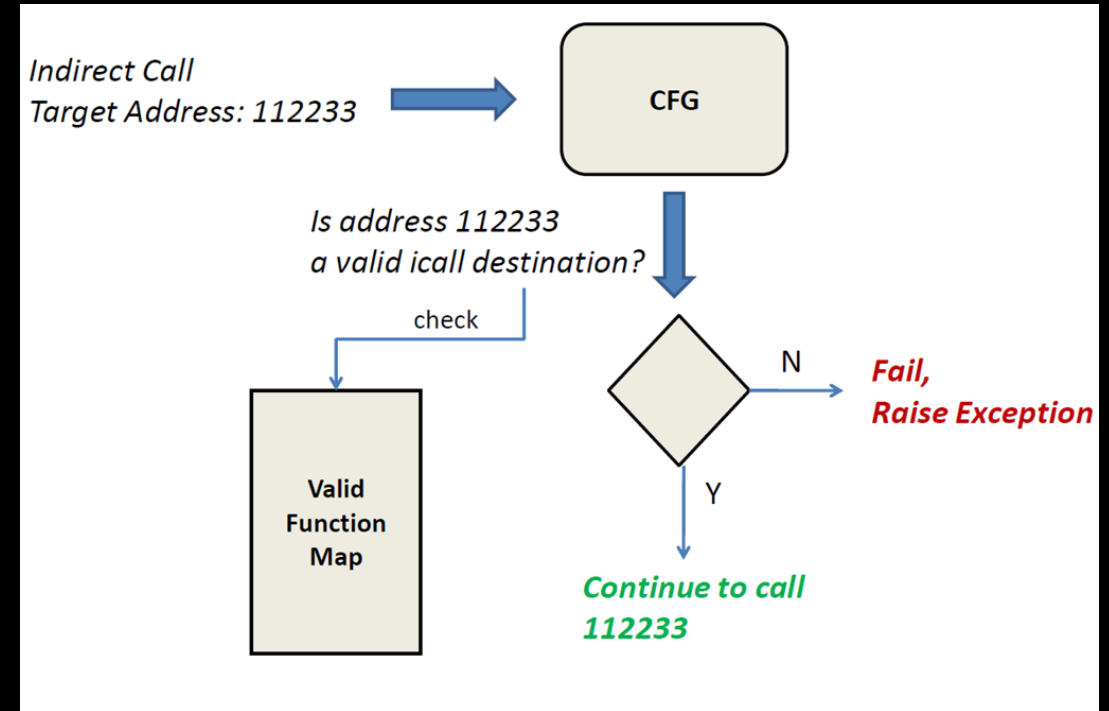
ASLR entropy is the same for both 32-bit and 64-bit processes on Windows 7

64-bit processes receive much more entropy on Windows 8, especially with high entropy (HE) enabled

Windows Security Mitigations

Control Flow Guard

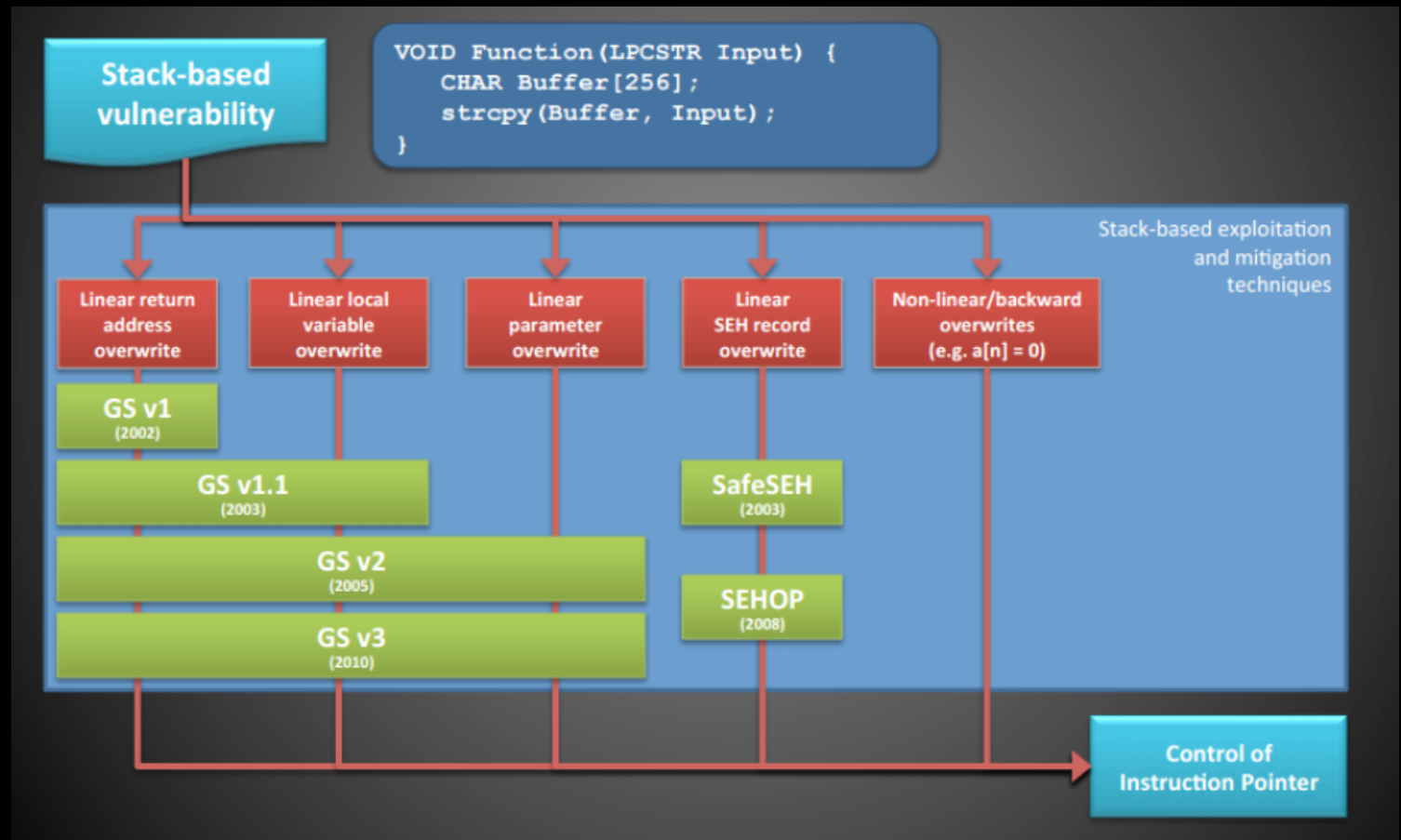
- All indirect call are checked by predefined read-only bitmap
- Attack Vtable is history now.
- Bypassed by
 - Overwrite CFG unprotected value (return address, SEH handler, etc.).
 - Overwrite CFG disabled module
 - COOP++



Windows Security Mitigations

Stack Based Vulnerability Mitigations

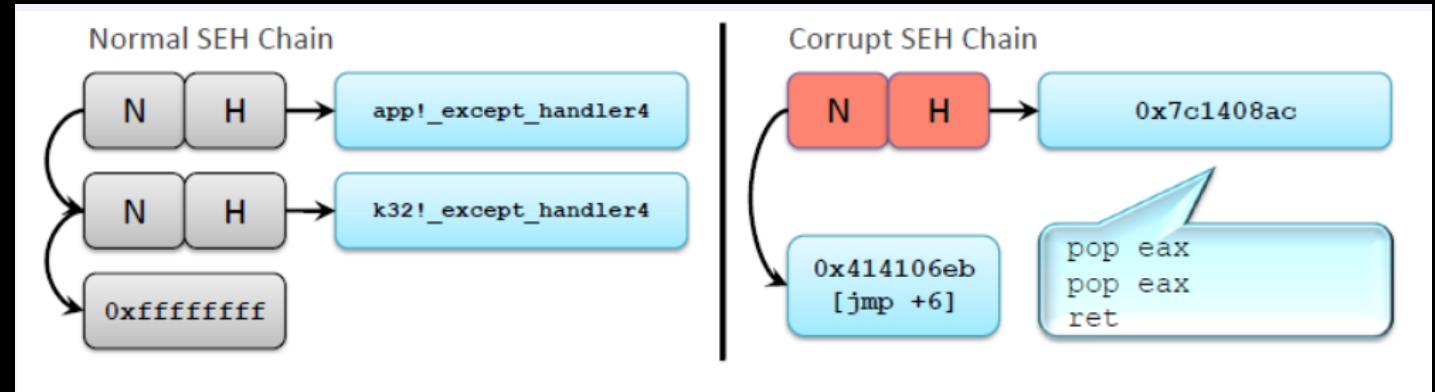
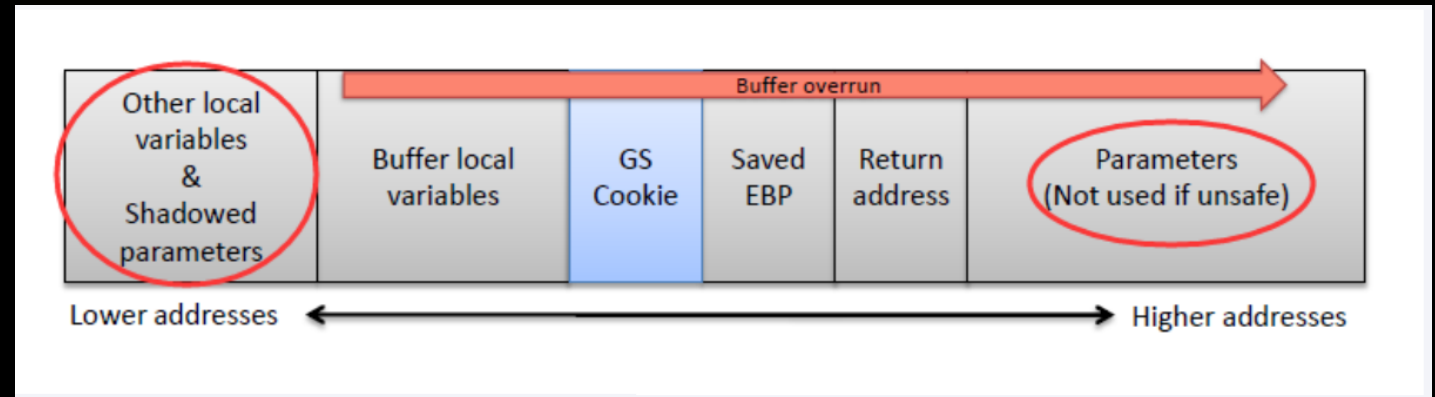
- GS
- SafeSEH
- SEHOP



Windows Security Mitigations

GS

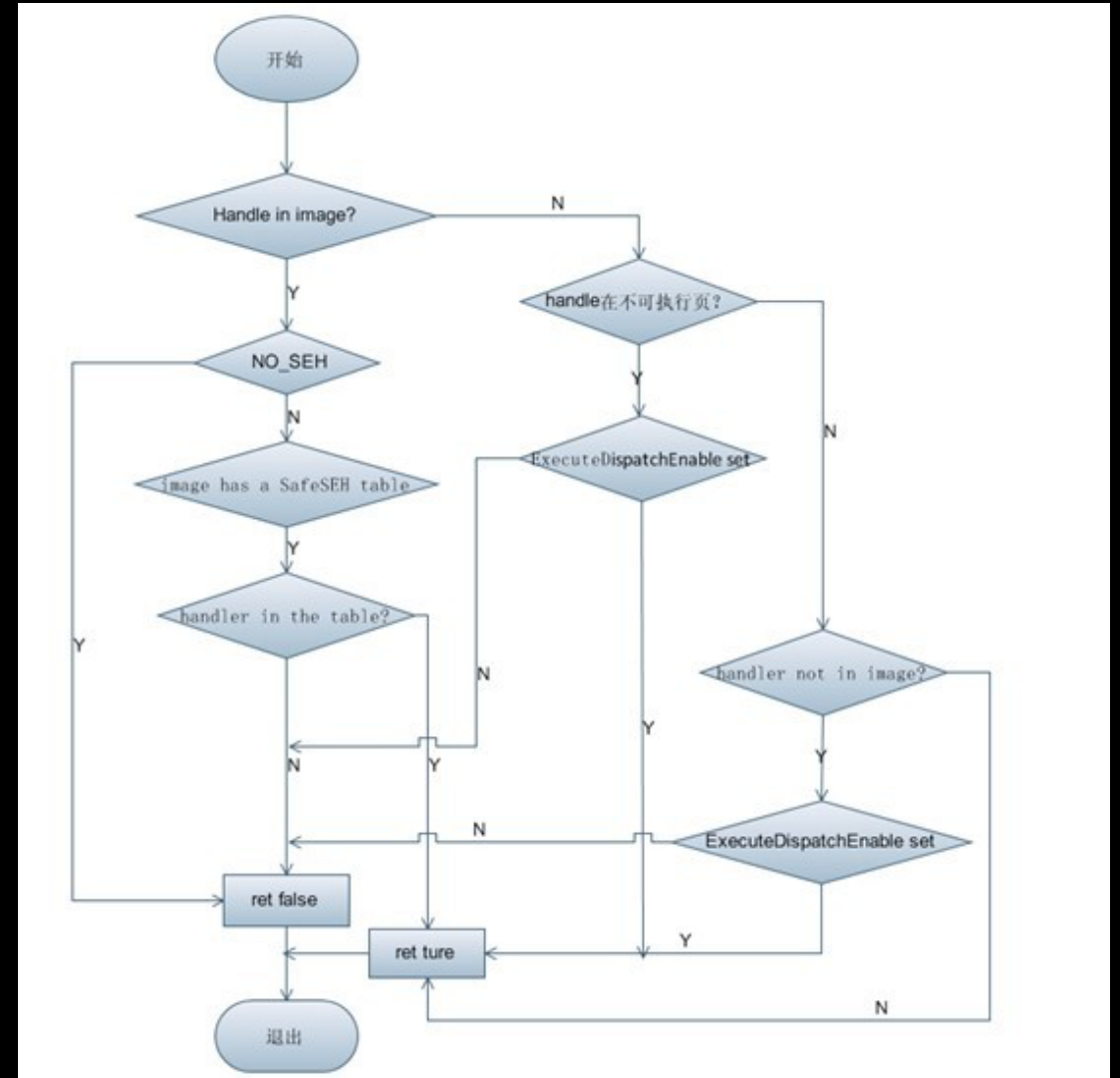
- Similar to stack canary
- Bypassed by
 - corrupt SEH(x86)
 - Stack underflow
 - nonlinear write



Windows Security Mitigations

SafeSEH(x86)

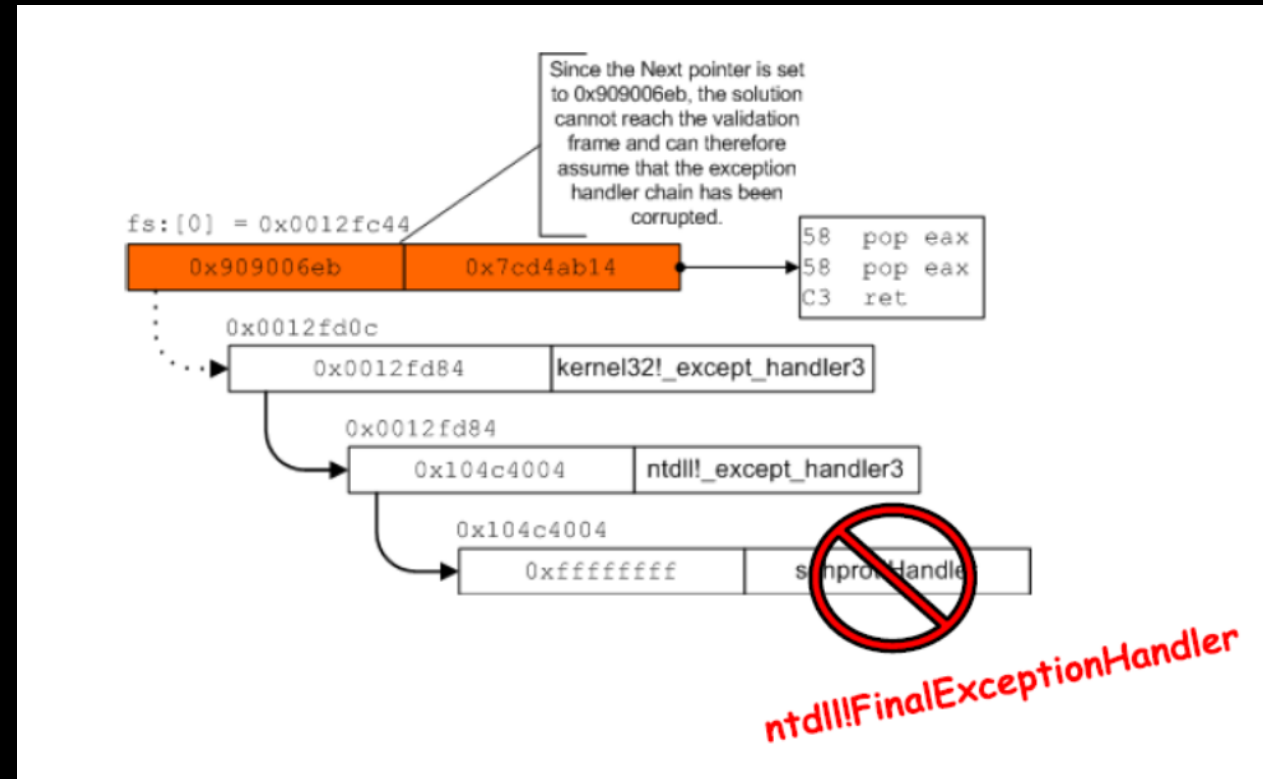
- Check whether handler is valid before calling the exception handler
- Bypassed by:
 - corrupt handler to an image with seh but without safeseh



Windows Security Mitigations

SEHOP(x86)

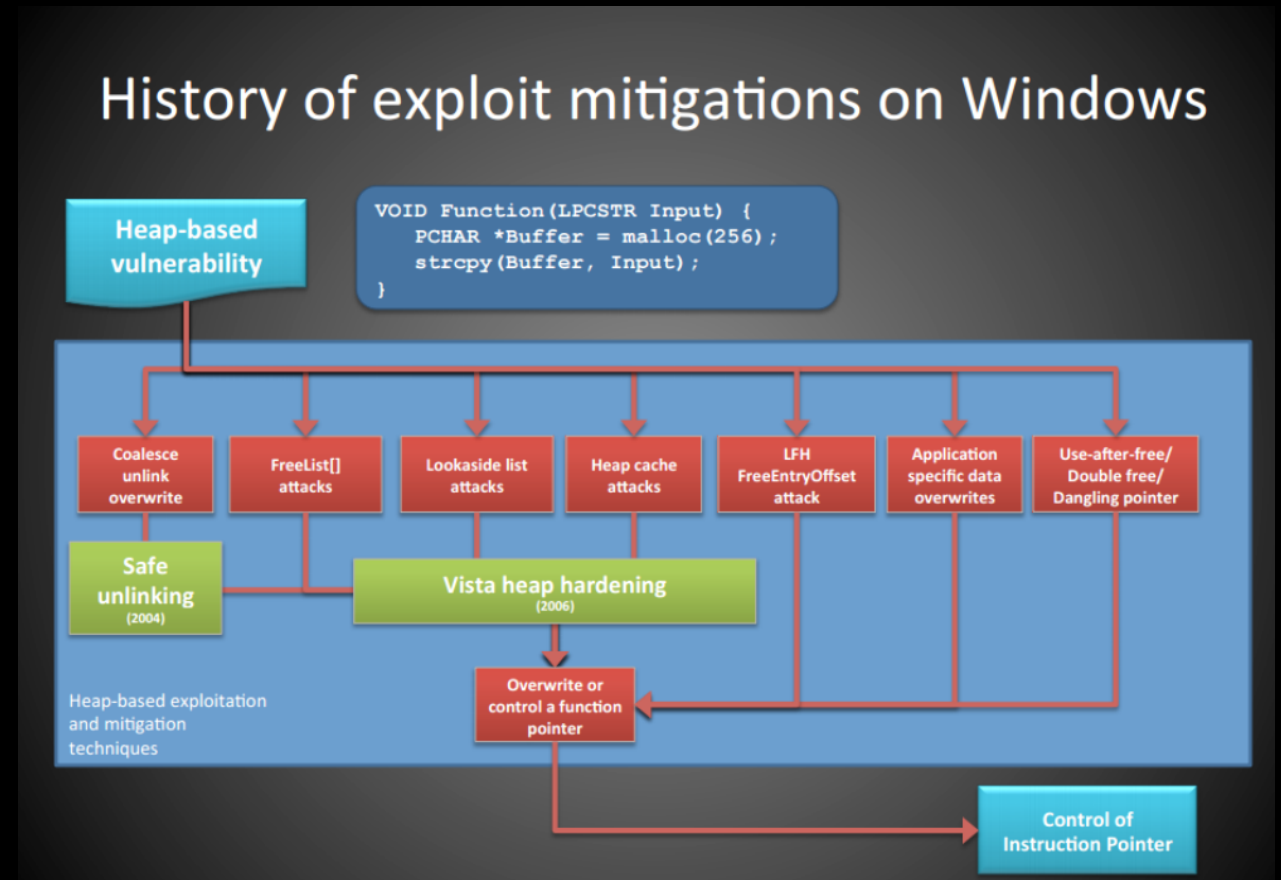
- Check whether SEH chain ends with ntdll!FinalExceptionHandler
- Bypassed by
 - Leak stack address and recover the SEH chain



Windows Security Mitigations

Heap-based vulnerability mitigations

- Metadata check & hardening
- LFH allocation randomization
- VirtualAlloc randomization



Windows Security Mitigations

Metadata check & hardening

- Almost impossible to attack heap meta-data
 - Safe unlink
 - Replace lookaside lists with LFH
 - Heap cookies & Guard pages
 - Heap cookies are checked in some places such as entry free
 - Zero Permission Guard pages after VirtualAlloc memory
 - Metadata encoding
 - Pointer encoding
 - Almost all function pointer are encoded such as VEH, UEF, CommitRoutine, etc.
- Bypassed by
 - Overflow User data

Windows Security Mitigations

Metadata check & hardening

Change in Windows 8	Impact
LFH is now a bitmap-based allocator	LinkOffset corruption no longer possible [8]
Multiple catch-all EH blocks removed	Exceptions are no longer swallowed
HEAP handle can no longer be freed	Prevents attacks that try to corrupt HEAP handle state [7]
HEAP CommitRoutine encoded with global key	Prevents attacks that enable reliable control of the CommitRoutine pointer [7]
Validation of extended block header	Prevents unintended free of in-use heap blocks [7]
Busy blocks cannot be allocated	Prevents various attacks that reallocate an in-use block [8,11]
Heap encoding is now enabled in kernel mode	Better protection of heap entry headers [19]

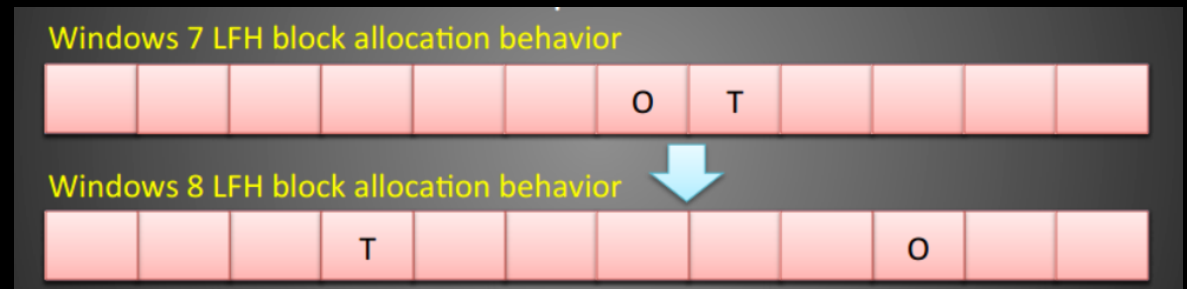
Windows Security Mitigations

VirtualAlloc randomization

- `Ptr=VirtualAlloc(size+random)`, return `ptr+random`

LFH allocation randomization

- `GetNextFreedLFHblock(random_start_index)`
- Bypassed by
 - allocate LFH unhandled size (larger than 0x4000)
 - allocate LFH disabled size (specific-sized LFH will enable only if allocation times exceeded some threshold)
 - heap spray
 - brute-force



Linux Exploit Techniques on Windows

General Exploit Techniques

- Heap metadata attack(off-by-one, house of xxx, xxxbin attack)
 - Hard, heap metadata are well protected on windows
- Heap manipulation (heap fengshui or something)
 - A little hard due to LFH allocation randomization
- Stack canary overwrite
 - OK, Stack cookie on .data section and writeable

Linux Exploit Techniques on Windows

General Exploit Techniques

- Return oriented programming
 - Sometime hard, indirect calls are protected by CFG
- Disable DEP via mprotect like function
 - OK, VirtualProtect on windows
- System call style shellcode
 - Hard, Windows system calls are not well-documented and vary by version

Linux Exploit Techniques on Windows

Information Leak Techniques

- Non-PIE Binary Fixed binary base
 - Hard, Binary Base are changed every time system reboot
- Leak share object base via GOT/GOT_PLT
 - OK, IAT is still readable
- Dynamic link related techniques such as DYNELF, ret2dlresolve
 - No lazy binding, Ret2dlresolve related techniques are unavailable
 - IAT EAT are readable, DYNELF-like things are still available

Linux Exploit Techniques on Windows

Information Leak Techniques

- Leak stack address via non-stack address(such as libc environ)
 - OK, Stack address can be leaked via heap, TEB, etc.
- Leak Address via Format String Bug
 - No \$ sign, a little hard
- Leak Stack/SO/Binary Base Address via uninitialized stack buffer (OK)
- Leak memory via stack_check_failed
 - Impossible, report_gs_failed don't print anything

Linux Exploit Techniques on Windows

Control Flow hijack Techniques

- GOT overwrite
 - Impossible, IAT on windows are read-only
- Internal function pointers overwrite (IO_FILE_JUMP, free hook etc.)
 - Hard, some function pointers are encoded or removed
 - UEF VEH encoded, PEB RtlEnterCriticalSection, RtlLeaveCriticalSection Removed.
 - Some function pointer such as SEH handler are still available to write

Linux Exploit Techniques on Windows

Control Flow hijack Techniques

- Vtable overwrite
 - Hard, CFG limited the overwrite value to function start
- Return address non-linear overwrite (OK)
- User function pointer overwrite (OK)

Windows-only Exploit Techniques

Bypass GS via SEH(x86)

- What is SEH
 - For function contains try..except block, a VC_EXCEPTION_REGISTRATION struct will be pushed into stack
 - Overwrite handler and trigger a exception to hijack control flow

```
struct VC_EXCEPTION_REGISTRATION
{
    VC_EXCEPTION_REGISTRATION* prev;
    FARPROC handler;
    scopetable_entry* scopetable; //指向scopetable 数组指针
    int _index; //在scopetable_entry 中索引
    DWORD _ebp; //当前EBP 值
}
```

寄存器和局部变量	
ebp ^ cookie	-1c
esp	-18
xxxx	-14
fs:[0]	-10
handler	-c
scopetable ^ cookie	-8
trylevel	-4
original ebp	ebp
Ret addr	+4

Windows-only Exploit Techniques

Bypass GS by overwriting SEH

- Bypass SafeSEH
 - Corrupt handler to an image with SHE but without safeSEH. (only way, see ntdll.dll!RtlIsValidHandler)
- Bypass SEHOP
 - Leak stack address, recover SEH chains
- A little hard

```
bool RtlIsValidHandler(handler)
{
    if (handler image has a SafeSEH table) {
        if (handler found in the table)
            return TRUE;
        else
            return FALSE;
    }
    if (ExecuteDispatchEnable|ImageDispatchEnable bits set in the process flags)
        return TRUE;
    if (handler is on a executable page){
        if (handler is in an image) {
            if (image has the IMAGE_DLLCHARACTERISTICS_NO_SEH flag set)
                return FALSE;
            if (image is a .NET assembly with the ILOnly flag set)
                return FALSE;
            return TRUE;
        }
        if (handler is not in an image) {
            if (ImageDispatchEnable bit set in the process flags)
                return TRUE;
            else
                return FALSE;
        }
    }
    if (handler is on a non-executable page) {
        if (ExecuteDispatchEnable bit set in the process flags)
            return TRUE;
        else
            raise ACCESS_VIOLATION;
    }
}
```

Windows-only Exploit Techniques

- X86 address brute-force
 - Only 8 bits randomization entropy for x86 image&dll base
- Cross Binary Leak
 - Some kernel related dlls (such as ntdll.dll kernel32.dll) share base among all processes
- Cross Same-Binary Process Leak
 - Image randomization base changed every time system booted

References

- <http://blog.talosintelligence.com/2014/06/exceptional-behavior-windows-81-x64-seh.html>
- http://media.blackhat.com/bh-us-12/Briefings/Valasek/BH_US_12_Valasek_Windows_8_Heap_Internals_Slides.pdf
- <https://www.blackhat.com/docs/us-16/materials/us-16-Yason-Windows-10-Segment-Heap-Internals.pdf>
- <http://www.blogs8.cn/posts/WSGsfd>

Thank You

Atum