Where am I? Operating System And Virtualization Identification Without System Calls

Jason L. Wright (jason@thought.net)

Cyber Security Symposium

April 17-19, 2017

Coeur d’Alene, Idaho, USA

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

What can I see? Given:

- Code execution
  - Arbitrary code
  - Unprivileged (user-level not kernel level)
  - Gather info without violating system call policy
- It’s about knowing:
  - what information is available
  - … and whether it is possible to monitor its access (or lie)

What is visible to user processes?

- Focus on x86 (32 and 64 bit, protected mode)
- General purpose registers:
  - EAX, EBX, ECX, EDX, EBP, ESP, ESI, EDI
- Normal segment selectors:
  - CS, DS, ES, FS, GS, SS
- Special segment selectors:
  - LDTR, TR
- Descriptor table registers:
  - GDTR, IDTR
- Machine Status Word: MSW

Segments: movl $1, (%eax)

- AT&T syntax (destination on the right)
- This means: move the constant 1 into the address stored in EAX
- In C: int *p = 1; // assuming p is stored in EAX

- The full address is DS:EAX
  DS is a segment selector (a 16 bit index into the OS controlled “Global Descriptor Table (GDT)” or “Local Descriptor Table (LDT)”
- Descriptor Table provides BASE, so virtual address is BASE+EAX
- Page tables turn BASE+EAX into the physical page of memory

So:

MOV $1, (%EAX)

Virtual address = DS:EAX

BASE = GDT[DS].BASE

Virtual address: DS:EAX = BASE + EAX

Physical address = page table[virtual address]

Reading current segments

Easy: PUSH CS; POP EAX

Push the current code segment onto the stack, pop it back into EAX

Works for all 6 “normal” segment selectors (CS, DS, ES, FS, GS, SS)

SLDT and STR instructions for fetching LDT and TR segment selectors

Probing for segments

LAR (load access rights)
LSL (load segment limit)
VERR (verify segment for reading)
VERW (verify segment for writing)

LAR and LSL allow probing for visibility of arbitrary segments
Can be used to generate “segment map” of the current environment

A list of valid segments, their type and their permissions

Segment Example: FreeBSD 10.2 64bit

Segment				
2	3	6	7	8
FS	GS	CS	DS,ES,SS	
Process local	Thread local	32bit code	data	64bit code

Segment Example: Windows 10 64bit

Segment			
4	5	6	10
	DS, ES, GS, SS	CS	DS,ES,SS
32bit code	data	64bit code	Thread local

Machine Status Word (aka CR0)

“SMSW is only useful in operating-system software. However, it is not a privileged instruction and can be used in application programs. The [instruction] is provided for compatibility with the Intel 286 processor.”

Allows examination of Configuration Register 0 (CR0)
One bit of differentiation power: Alignment Mask bit

AM bit	OS
clear	Windows (XP, 8.1, 10 i386), MacOS 10.11, OpenBSD
set	Windows (10 amd64, Server 2016), Linux, FreeBSD

Descriptor Tables

“[SGDT, SIDT] [are] only useful in operating-system software. However, [they] can be used in application programs without causing an exception to be generated.”

Retrieve the base and limit of the GDT or IDT

(cannot access the <u>contents</u> of GDT/IDT, though)

GDT Identification

Base	Limit	OS
0x00020000	0xeff	Linux 3.13 domU
0x8003f000	0x03ff	Windows XP SP3
0x80DCC000	0x03ff	Windows 8.1
0x8129C000	0x03ff	Windows 10
0xc15b07a4	0x0097	FreeBSD 10.3
0xf4e1c000	0xffff	OpenBSD 5.8
0xf7beb000	0x00ff	Linux 2.6.13
0xf7be6000	0x00ff	Linux 3.13
0xf714f000	0x00ff	Linux 3.19

Base	Limit	OS
0x0000000008169a450	0x0067	FreeBSD 10.2
0xffff800000011068	0x003f	OpenBSD 5.8
0xffff820000000000	0xefff	Linux 3.16 dom0
0xffff820000020000	0xefff	Linux 3.16 domU
0xffff88002fc09000	0x007f	Linux 3.16
0xfffff80028dc3000	0x006f	Windows 10
0xfffff80089a54000	0x006f	Windows Server 2016
0xffffff8000001000	0x0097	MacOS 10.11.6

Other sources of OS/Virtualization fingerprints

- IDTR (like GDT identification)
  - Linux fixmap IDT
- CPUID
  - Reserved leaf for hypervisors (EAX=0x40000000-4fffffff)
  - Hypervisor idiosyncrasies
    - Virtualbox RDRAND
    - Virtualbox XSAVE
- FPU initialization
  - FreeBSD 32bit initializes precision and reserved bit differently than everyone else

Future work

- Are there other sources of fingerprinting?
    - Are there mitigations?
- Would similar techniques work on other instruction sets?

Availability

- Utility for fetching data exposed to user processes:
    - https://github.com/wrigjl/ucpuinfo
- Modified Linux, Xen, and FreeBSD patches:
    - https://github.com/CyberGrandChallenge