Detecting the Presence of Virtual Machines Using the Local Data Table

Danny Quist {chamuco@gmail.com}
Val Smith {mvalsmith@metasploit.com}

Offensive Computing
http://www.offensivecomputing.net/

Abstract

In this paper we describe a method for determining the presence of virtual machine emulation in a non-privileged operating environment. This attack is useful for triggering anti-virtualization attacks and evading analysis. We then discuss methods for mitigating this risk for malware analysts. This method was demonstrated using the Windows series of operating systems.

Introduction

The SIDT mechanism as implemented by Tobias Klein [1] and separately by Joanna Rutkowska [2] is a method for detecting the presence of a virtual machine environment. While the test is by no means thorough, it is an effective test for the presence of an emulated CPU environment on a single-processor machine. There are various problems with the implementation, however. If a multi-core CPU is used, the interrupt descriptor table can change significantly when the process is run on different cores. Furthermore if two or more physical processors are present the same implementation issues apply.

The Interrupt Descriptor Table (IDT) is an internal data structure used by the operating system in processing interrupts. Devices use the IDT to process events in the operating system. The IDT is a data structure often exploited by rootkits. [4] By subverting the IDT, the attacker can point critical items such as the keyboard interrupt to a different function. Using this method an attacker can then insert malicious code to be executed when certain interrupts are run.

The Redpill and scoopy_doo mechanisms use the SIDT assembly operation to retrieve the interrupt descriptor table from the CPU. This data is available at unprivileged operating levels. By providing this key information a non-privileged (non-OS level) process can then query this information. This is bad for a number of reasons. First this exposes a small level of detail regarding the operating state of the underlying OS. Second, this information can be used to ascertain the operating environment of the OS. Malicious software can then determine the presence of a virtual machine. This can allow the program to terminate itself, or implement specific exploits to escape from the virtual machine.

IDT Usage Issues and Workarounds

The value of the IDT is specific to the running processor. In a single-processor environment the value of the IDT is constant, and can be effectively used to determine the presence of a multi-processor machine. When multiple cores are added, each processor has it's own IDT. This is the source of the problems given by the scoopy_doo and Redpill methods.

There are a couple of ways to get around the IDT problem. First using the Redpill method, one can run the tasks repeatedly in a loop on the system. The inherent problem is that the IDT value will be different for each of the processors. By running multiple times one can build a statistical map of the changes present on the system. This may not be optimal due to the added loading of the processor.

Another possible work around to the issue is to use the SetThreadAffinityMask() Windows API call to limit thread execution to one processor. When running this test it is possible to accurately limit the threads execution environment on a native processor only. It is not possible, however to limit that execution on the VM system. Since the VM can be scheduled to run on various processors, this value will change as the VM thread is executed on different processors. Since the problem space is centered around detecting virtualization or not, this check is useless.

LDT Process Determination

Our method is a variant on the SIDT process used by Redpill and scoopy_doo. We use the Local Descriptor Table (LDT) as a signature for virtualization. The LDT provides segmentation for operating privilege changes. It provides the base addresses, access rights, type, length, and usage information for each segment.

The value of the LDT, like the IDT and Global Descriptor Table (GDT) are readable by unprivileged memory. The problem for the VM arises when these memory addresses are used. [3] Since the VM is running under an unprivileged process itself, it cannot load or unload the values of the registers.

Furthermore the LDT is not used by all operating systems. Notably Windows does not use it, however Linux does. Since the VM must account for any discrepancies in the GDT and LDT, Windows' LDT running in virtualization must be a separate value than that which is present on the operating system. The SIDT, SLDT, and SGDT assembly operations must be further virtualized to maintain the virtualization. Since these cannot be the same, the VM provides a separate copy of each of these values.

Experimentation

The IDT (*Redpill*) and LDT (*Nopill*) methods were both tested extensively on multiple pieces of hardware and across multiple virtual machines.

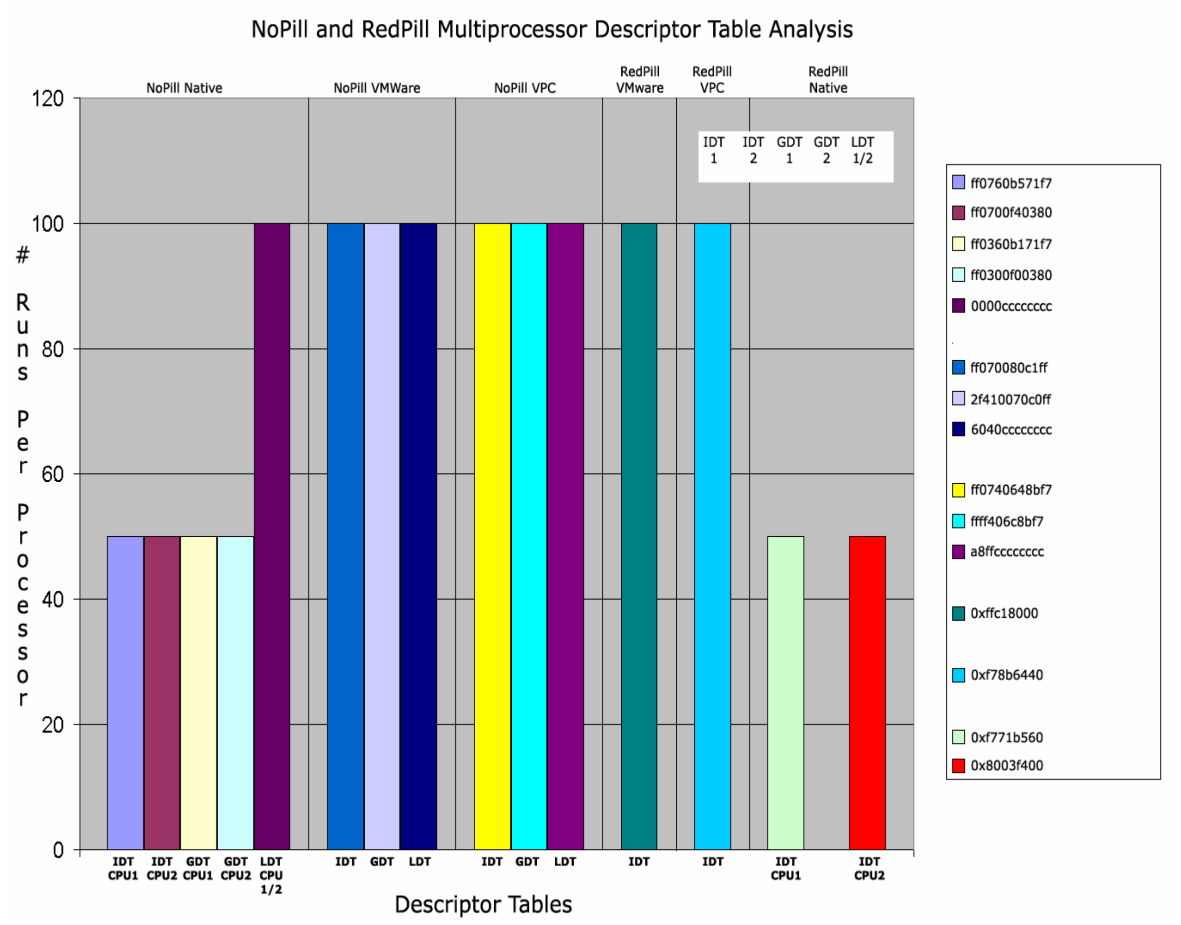
Specs:

- Dual Intel Xeon CPU 3.06GHz
- Single AMD Athalon 2.0 GHz
- Microsoft Virtual PC 5.4.582.27
- VMWare Workstation 5.5.1 build-19175
- Windows XP Sp 2
- Windows 2000 SP 4

Redpill was found to correctly determine its virtualized state 100% of the time when run inside a VM, as was Nopill. However, when run on native, non-virtualized, hardware Redpill was only %50.36 accurate while Nopill was 100%. Out of 10,000 runs on native Dual Processor Intel hardware Redpill generated 4964 false positives. This means that 49.64% of the time a piece of software implementing the SIDT VM signature technique will incorrectly detect that it is running inside of a virtual machine.

The IDT base address flipped ~50% of the time on a dual processor. Assuming even distribution across all processors the IDT VM signature method will be incorrect proportionately to the number of processors in the system.

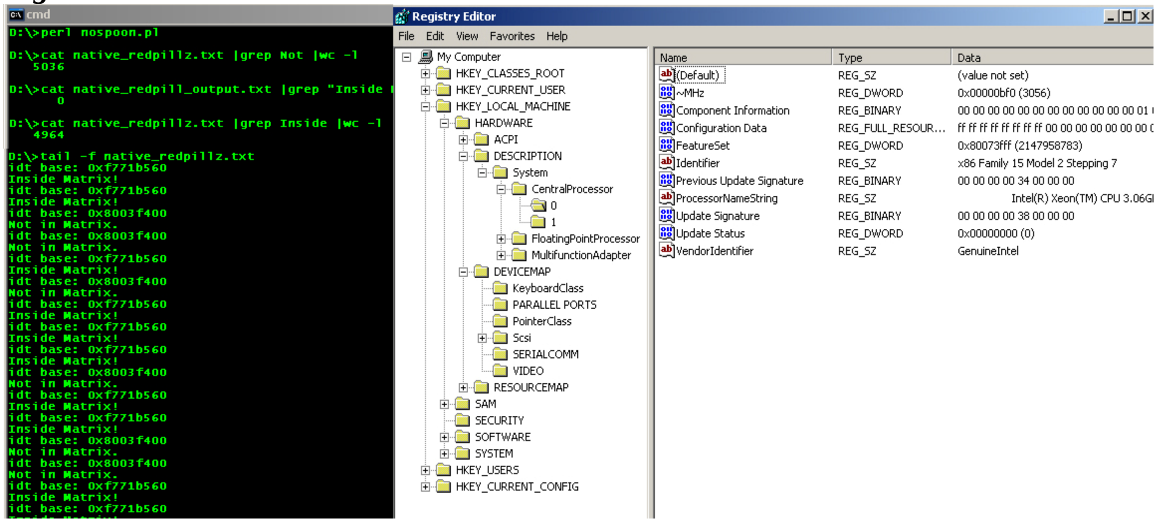
Ex. On a quad processor system Redpill will be correct ~25% of the time and incorrect ~75%.



The above graph shows the results of 100 runs of Nopill and Redpill and is broken down by descriptor table values. For example Running Nopill on native Intel dual processor hardware the IDT was one value 50% `ff0760b571f7` of the time and a second value `ff0700f40380` the other 50%. The same was true for the GDT. However LDT had the same value 100% of the time. Since Redpill is an IDT only technique its accuracy proportion is dictated by the number of processors.

Screenshots

***Fig. 1** Native Hardware Runs*



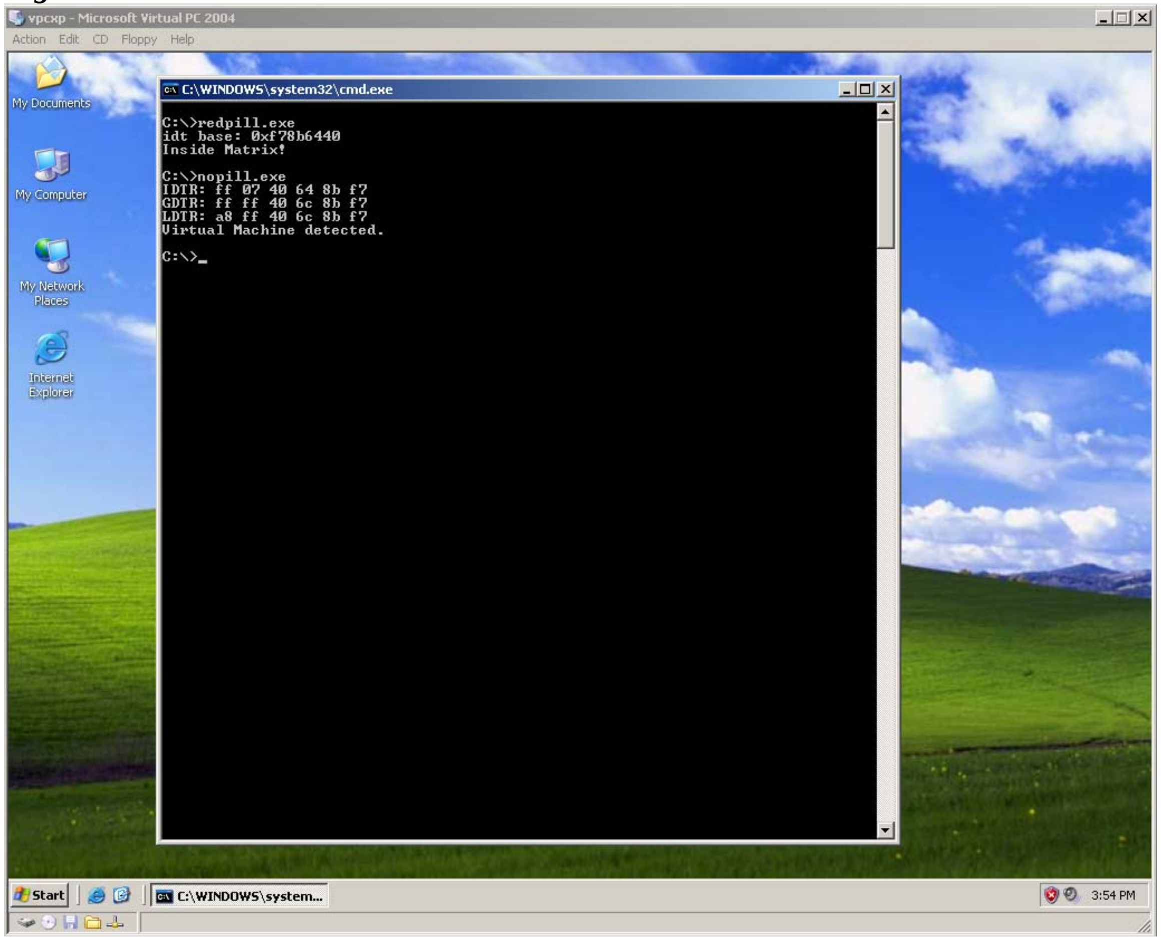
```
D:\>perl nospoon.pl

D:\>cat native_redpillz.txt |grep Not |wc -l
   5036

D:\>cat native_redpill_output.txt |grep "Inside 
      0

D:\>cat native_redpillz.txt |grep Inside |wc -l
   4964

D:\>tail -f native_redpillz.txt
idt base: 0xf771b560
Inside Matrix!
idt base: 0xf771b560
Inside Matrix!
idt base: 0x8003f400
Not in Matrix.
idt base: 0x8003f400
Not in Matrix.
idt base: 0xf771b560
Inside Matrix!
idt base: 0x8003f400
Not in Matrix.
idt base: 0xf771b560
Inside Matrix!
idt base: 0xf771b560
Inside Matrix!
idt base: 0xf771b560
Inside Matrix!
idt base: 0x8003f400
Not in Matrix.
idt base: 0xf771b560
Inside Matrix!
idt base: 0xf771b560
Inside Matrix!
idt base: 0x8003f400
Not in Matrix.
idt base: 0x8003f400
Not in Matrix.
idt base: 0xf771b560
Inside Matrix!
idt base: 0xf771b560
```

```
My Computer
  HKEY_CLASSES_ROOT
  HKEY_CURRENT_USER
  HKEY_LOCAL_MACHINE
    HARDWARE
      ACPI
      DESCRIPTION
        System
          CentralProcessor
            0
            1
          FloatingPointProcessor
          MultifunctionAdapter
      DEVICEMAP
        KeyboardClass
        PARALLEL PORTS
        PointerClass
        Scsi
        SERIALCOMM
        VIDEO
      RESOURCEMAP
    SAM
    SECURITY
    SOFTWARE
    SYSTEM
  HKEY_USERS
  HKEY_CURRENT_CONFIG
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
~MHz	REG_DWORD	0x00000bf0 (3056)
Component Information	REG_BINARY	00 00 00 00 00 00 00 00 00 00 00 00 01
Configuration Data	REG_FULL_RESOUR...	ff ff ff ff ff ff ff ff 00 00 00 00 00 00 00
FeatureSet	REG_DWORD	0x80073fff (2147958783)
Identifier	REG_SZ	x86 Family 15 Model 2 Stepping 7
Previous Update Signature	REG_BINARY	00 00 00 00 34 00 00 00
ProcessorNameString	REG_SZ	Intel(R) Xeon(TM) CPU 3.06G
Update Signature	REG_BINARY	00 00 00 00 38 00 00 00
Update Status	REG_DWORD	0x00000000 (0)
VendorIdentifier	REG_SZ	GenuineIntel

The Figure 1 graphic shows various runs of Redpill and the false positive rate. To the right are the registry settings showing the dual processor configuration.

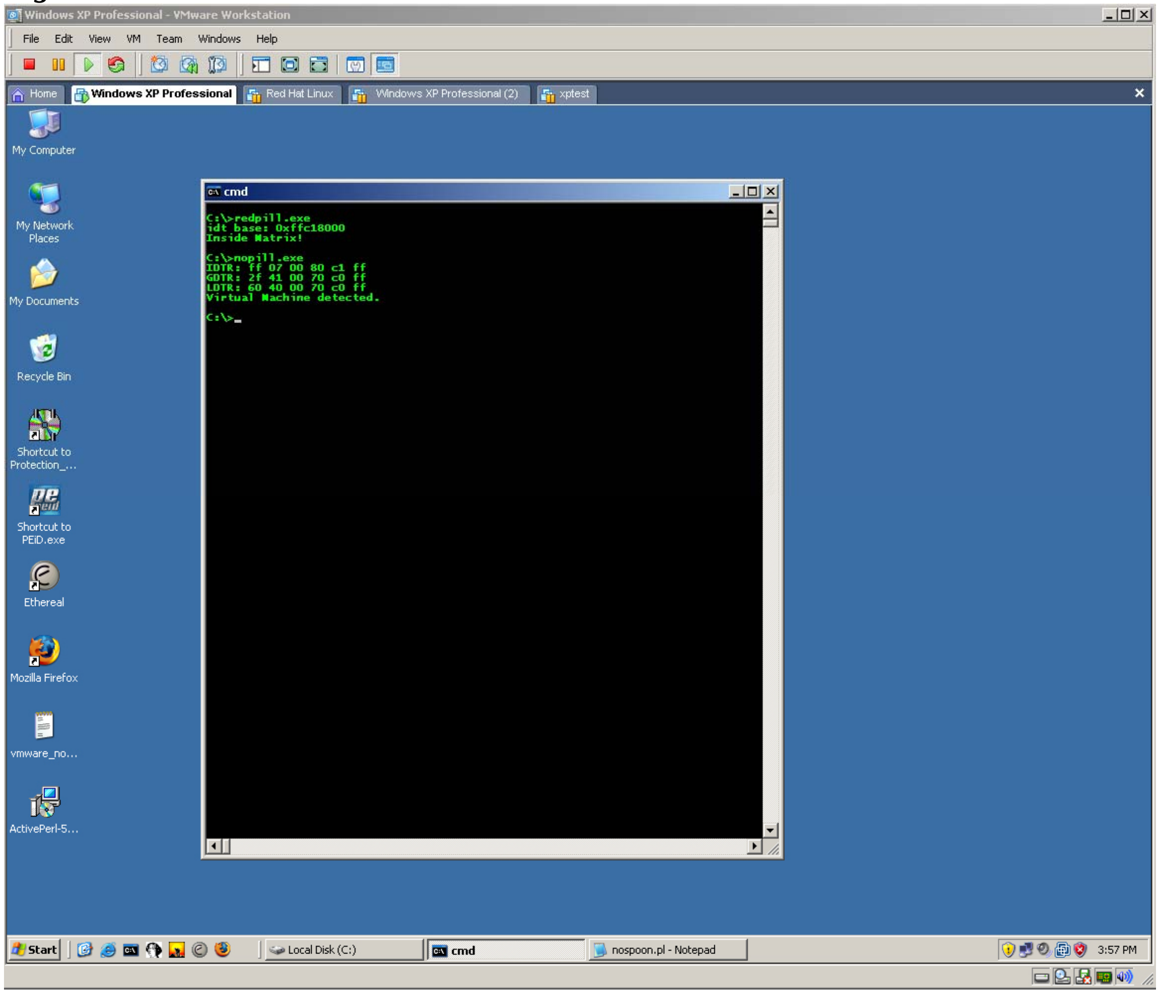
Fig. 2 *Virtual PC Runs*



```
C:\>redpill.exe
idt base: 0xf78b6440
Inside Matrix!

C:\>nopill.exe
IDTR: ff 07 40 64 8b f7
GDTR: ff ff 40 6c 8b f7
LDTR: a8 ff 40 6c 8b f7
Virtual Machine detected.

C:\>_
```

Figure 2 show runs of Redpill and Nopill inside a Microsoft Virtual PC emulator.

Fig. 3 *VMWare Runs*



```
C:\>redpill.exe
idt base: 0xffc18000
Inside Matrix!

C:\>nopill.exe
IDTR: ff 07 00 80 c1 ff
GDTR: 2f 41 00 70 c0 ff
LDTR: 60 40 00 70 c0 ff
Virtual Machine detected.

C:\>_
```

Figure 3 shows runs of both Redpill and Nopill on a VMWare emulator.

LDT Problems

The LDT method is not without problems, however. There are some issues when using these on a fully emulated VM. Specifically we've found that the Virtual PC implementation running on the Power PC architecture will not yield any useful results. The virtual PC environment exhibits extremely poor performance on the PowerPC architecture. This can cause analysis of malicious binaries to take much more time than other VM instantiations.

Mitigation Techniques

Since this attack relies on fundamental implementations of the VM architecture, it is difficult to mitigate these issues. Some methods that can be done are to perform a completely non-executed analysis of the binary. Also binary patching can be implemented to jump over or skip the checks. With further binary obfuscation it could be possible to completely bypass any analysts efforts to dynamically analyze the binary.

References

[1] http://www.trapkit.de/research/vmm/scoopydoo/scoopy_doo.htm
[2] http://www.invisiblethings.org/papers/redpill.html
[3] J. S., Robin; C. E., Irvine: "Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor", Proceedings of the 9th USENIX Security Symposium, August 14-17, 2000.
[4] G. Hoglund; J. Butler: “Rootkits”
[5] http://www.embedded.com/showArticle.jhtml?articleID=55301875

Appendix A: Code

```c
#include <stdio.h>

inline int idtCheck ()
{
	unsigned char m[6];

	__asm sidt m;
	printf("IDTR: %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n", m[0], m[1],
m[2], m[3], m[4], m[5]);
	return (m[5]>0xd0) ? 1 : 0;
}

int gdtCheck()
{
	unsigned char m[6];

	__asm sgdt m;
	printf("GDTR: %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n", m[0], m[1],
m[2], m[3], m[4], m[5]);
	return (m[5]>0xd0) ? 1 : 0;

}

int ldtCheck()
{
	unsigned char m[6];

	__asm sldt m;
	printf("LDTR: %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n", m[0], m[1],
m[2], m[3], m[4], m[5]);
	return (m[0] != 0x00 && m[1] != 0x00) ? 1 : 0;

}

int main(int argc, char * argv[])
{
	idtCheck();
	gdtCheck();
	if (ldtCheck())
		printf("Virtual Machine detected.\n");
	else
		printf("Native machine detected.\n");

	return 0;
}
```