Intelguardians

On the Cutting Edge: Thwarting Virtual Machine Detection

Tom Liston, Senior Security Consultant – Intelguardians
Handler – SANS Internet Storm Center
tom@intelguardians.com

Ed Skoudis, Founder / Senior Security Consultant – Intelguardians
Handler – SANS Internet Storm Center
ed@intelguardians.com

http://www.intelguardians.com

 1

Hello, and welcome to our SANS@Night presentation on virtual machine detection, and some possible methods for thwarting the types of detection currently in use by malware in the wild.

We'll start things off with an overview of some of the methods being used to detect the use of virtual machine environments – how they work and what exactly they are detecting. Finally, we'll pass along some tips for making use of a virtualized environment more difficult for the bad guys to detect.

So, please sit back, relax, and follow along. If you have any questions, please feel free to ask!

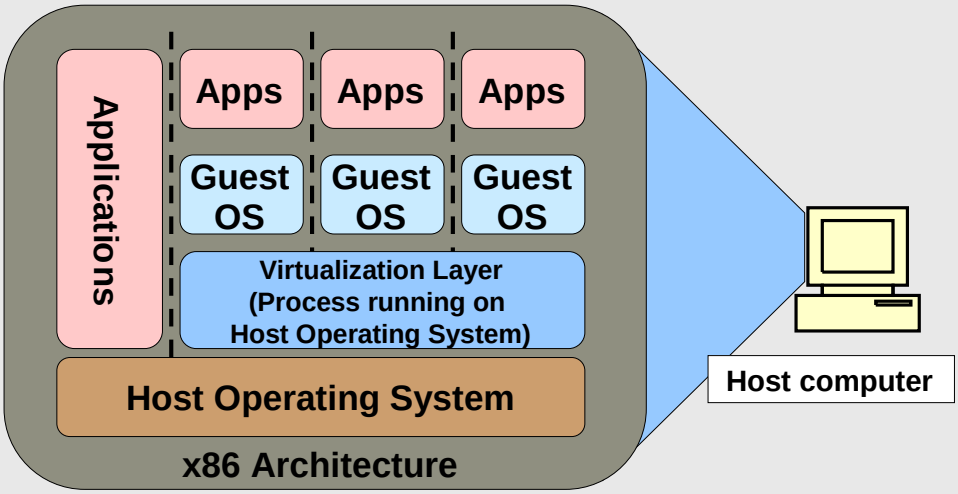
Tom Liston

tom@intelguardians.com

Ed Skoudis

ed@intelguardians.com

Virtual Machine Environment



 2

Virtual machine environments (VMEs), such as VMware, VirtualPC, Xen, BOCHS, and User-Mode Linux, let a user or administrator run one or more "guest" operating systems on top of another "host" operating system. Each of the guest operating systems "run" in an emulated environment and are provided by the VME with mediated access to both virtual and real hardware. In theory, the environment provided by the VME is self contained, isolated, and indistinguishable from a "real" machine.

Virtualization Benefits

- By consolidating multiple servers onto a single hardware platform
  - Decrease hardware costs
  - Simplify maintenance
  - Improve reliability
- These benefits are driving a boom in virtualization use
  - Both Intel and AMD have announced processor extensions to support virtualization

Virtualization of both clients and servers has several very tangible benefits that are driving a boom in the use of VMEs. Obviously, there are cost benefits anytime you can decrease the number of physical machines required within your environment, but some of the ease-of-use benefits of VMEs have an impact on the bottom line as well. The ability to "roll-back" any changes to a virtual server makes testing and maintenance far easier, reducing support costs. By focusing limited support dollars on a smaller number of machines, reliability is increased.

Given the rising use of VMEs, computer attackers are very interested in detecting the presence of VMEs, both locally on a potential VME and across the network. Beyond simply their increased use, however, there are some specific uses of VME technology that are driving the computer underground toward deploying techniques for virtual machine detection. We'll explore some of these uses in-depth in the next two slides.

Virtual Machine Detection – Why?

1) Malicious code researchers increasingly use virtual machine technology to analyze samples
   - Virtualization offers many benefits:
     - Multiple operating systems
     - Ability to reset to a previous "snapshot" undoing any changes made by the malware
     - Easily monitored
     - Isolation
       - Hmmmm…. We'll take another look at this one later

Because so many security researchers rely on VMEs to analyze malicious code, malware developers are actively trying to foil such analysis by detecting VMEs. If malicious code detects a VME, it can shut off some of its more powerful malicious functionality so that researchers cannot observe it and devise defenses. Given the malicious code's altered functionality in light of a VME, some researchers may not notice its deeper and more insidious functionality.

We are seeing an increasing number of malicious programs carrying code to detect the presence of virtual environments.

Virtual Machine Detection – Why?

(…continued)

2) Virtual machines are often used to create honeypot or honeynet environments
   - This is done for the same reasons as for malicious code research
3) "Questionable usage patterns"
   - "Bridging" deployment
     - Using a single host machine with multiple guests, each accessing networks with different security levels
   - "Firewall" deployment
     - Deploying an insecure OS or application on a guest, and relying on the VME for isolation or reset-ability

Individuals and organizations that deploy honeypots or honeynets as research tools are attracted to virtualization technology for many of the same reasons as malicious code researchers. Given the possibility that the machine that he just 0wnz3r3d might be part of a virtual honeynet and that his every move may now be monitored, Joe-Hacker has a very strong motivation to discover that fact. However, as virtualization technology is increasingly deployed in the mainstream, this becomes less and less of a "sure" indicator that the system is of questionable value.

Finally, there are what we will call "questionable usage patterns" of virtualization technology: deployments that rely more than they probably should on the "isolation" aspects of virtualization. In these instances, virtual machine detection schemes are seen as a precursor to other types of attacks aimed at compromising that "isolation": "backdoor" virtual machine-to-virtual machine communication and, the Holy Grail of VME attacks, virtual machine escape.

Local Virtual Machine Detection

- Goal: Detect if you are inside a virtual machine (guest operating system) from within the machine
  - Am I running inside the Matrix, or in the Real World?

- There are currently four categories of methods for locally detecting the presence of a virtual machine:
  1. Look for VME artifacts in processes, file system, and/or registry
  2. Look for VME artifacts in memory
  3. Look for VME-specific virtual hardware
  4. Look for VME-specific processor instructions and capabilities
- Covers nearly all of the elements of the virtual machine

 6

The attacker's goal is to detect if their code is running in a virtual machine, or running on a real system. Are they in The Matrix, or The Real World?

To detect a virtual machine, an attacker has numerous options. There are four categories of local VME detection used today, including:

–Looking for VME artifacts in processes, file system, and/or registry

–Looking for VME artifacts in memory

–Looking for VME-specific virtual hardware

–Looking for VME-specific processor instructions and capabilities

If you think about it, you'll realize that most modern computer systems consist of a file system, memory, various hardware components, and the processor itself. The local VME detection mechanisms in this list cover each of these elements, analyzing each component as a generalized category of methods for VME detection. Thus, given that these four categories entail all the elements of a modern computer, any additional methods for local VME detection will likely still fit into the framework of these four categories.

1) VME Artifacts in Processes, File System, and/or Registry

- Some VMEs insert elements into the Guest that can be easily found
  - Running processes or services
  - Files and/or directories
  - Specific registry keys
- Some Phatbot malware specimens use this technique
- In a VMware Workstation WinXP Guest:
  - Running "VMtools" service
  - Over 50 different references in the file system to "VMware" and vmx
  - Over 300 references in the Registry to "VMware"
- This method is of limited utility, easily fooled
  - Rootkit-like techniques tweak the operating system to hide artifacts from users
  - Similar techniques could be applied to hide VME

Some virtual machine tools insert an enormous amount of tell-tale signs in the guest operating system, including running processes and services, files and directories, and registry keys. Some instances of the phatbot malicious code look for these items from VMware.

In a VMware Workstation Windows XP Guest, there is the running Vmtools service, over 50 references in the file system to VMware and vmx, and over 300 references in the registry to "VMware". That's a lot of areas to choose from for the attacker to detect the virtual machine.

However, while the pickings are plentiful, they aren't too reliable. A researcher could hide these items by employing various techniques borrowed from malicious rootkit technologies. Thus, the bad guys' detection mechanism can be readily fooled.

2) Look for VME Artifacts in Memory

- The Guest system memory map has some differences from the Host memory map
  - Strings found in memory
    - In a VMware Workstation WinXP Guest, we dumped RAM using dd
    - We found over 1,500 references to "VMware" in memory
    - Rather a heavy-weight approach, but could be refined to focus on specific regions
  - Some critical operating system structures located in different places
    - Much quicker, easier, and hard to fool without redesign of VME
- One particular memory difference is the location of the Interrupt Descriptor Table (IDT)
  - On Host machines, it is typically low in memory
  - On Guest machines, it is typically higher in memory
  - Cannot be the same, because the processor has a register pointing to it (IDTR)
- This technique is usable across different VMEs (VirtualPC and VMware) and more difficult to fool

A second area of VME detection involves looking for anomalies in memory introduced by the virtualization process. First off, the attacker could search through system memory for references to the virtual machine. To test this, we dumped the memory from a running WindowsXP VMware guest, and found over 1,500 references to "VMware" in memory. Dumping and searching memory is a performance hog, so the technique is not trivial to deploy. But, targeting specific areas of memory, where an attacker reliably knows to look, might make the technique more effective.

A better memory-related technique involves looking at pointers to critical operating system tables that are typically relocated on a virtual machine. One such table is the Interrupt Descriptor Table (IDT), which tells the system where various operating system interrupt handlers are located in memory. On real machines, the IDT is located lower in memory than it is on guest (i.e., virtual) machines.

This IDT-related technique is nice because it can be applied on both VirtualPC and VMware and between Linux and Windows. Also, because it is part of the underlying architecture of the VME, it is much harder to fool.

The Red Pill

- In November 2004, Joanna Rutkowska released a simple tool called "The Red Pill" that reliably detects virtual machine usage without looking for file system artifacts
  - http://invisiblethings.org/index.html#redpill
- Instead, this tool runs a single machine language instruction, SIDT "Store Interrupt Descriptor Table"
- This instruction, which can be run in user mode, takes the location of the Interrupt Descriptor Table Register (IDTR) and stores it in memory

 9

In November 2004, a researcher named Joanna Rutkowska introduced code that implements this IDT-checking concept. Her code, called "The Red Pill", runs a single machine language instruction, called SIDT. This instruction stores the contents of the Interrupt Descriptor Table Register (the IDTR which points to the IDT) in a processor register, where it is analyzed.

How The Red Pill Works

- The Red Pill simply runs the SIDT instruction and then checks the results
- Rutkowska observed that:
  - On VMware guest machines, the IDT is typically located at 0xffXXXXXX
  - On VirtualPC guests, it is located at 0xe8XXXXXX
  - On host operating systems, it is located lower than that, typically around 0x80ffffff (Windows) and 0xc0ffffff (Linux)
- The Red Pill merely looks at the first byte returned by SIDT
  - If it's greater than 0xd0, you've got a virtual machine
  - If it is less than or equal to 0xd0, you are in a real machine

The Red Pill simply looks at the IDTR (via the SIDT instruction).

Rutkowska observed that on VMware guest machines, the IDT is typically located at 0xffXXXXXX, while on VirtualPC guests, it is located at 0xe8XXXXXX. For host operating systems, the IDT is located far lower in memory. To handle both conditions, the Red Pill checks to see if the IDTR is greater than 0xd0000000. If so, the Red Pill prints a message saying that it is running in a guest operating system. If the returned result is less than or equal to that value, the Red Pill says that it is running on a host machine. We have found that the results are highly accurate for both VirtualPC and VMware, running both in RedHat Linux 7.2, 8.0, 9.0, and Enterprise 3.0. It likewise is accurate in Windows 2000, XP, and 2003.

The Red Pill on Various VMs

- This program works like a charm in detecting various virtual machine types, including:
  - VMWare
  - VirtualPC
  - Perhaps others??? Needs further testing
- Works regardless of virtual and real operating system type
  - Tested on both Windows (2000/XP/2003) and Linux (RH 8/9/Fedora Core 2/FC4)
  - Although, some Linux security packages (grsecurity, PAX, etc.) prevent this call from Ring 3 to Ring 0

In fact, this tool functions quite nicely on VMWare and VirtualPC. We are currently in the process of testing other virtual environments.

Also, the technique works regardless of the operating system type, for the most part. The technique works on Windows almost all of the time. On some Linuxes, however, additional security fixes prevent it from working. These alterations prevent certain kinds of memory access from Ring 3. Therefore, on such machines, the Red Pill will seg fault. This is a consequence of how Rutkowska wrote her code, and can be somewhat easily fixed.

Expanding on the Red Pill

- Besides the IDT (measured by the SIDT instruction), there are other tables in memory that are shifted by VMEs…
- Which have pointers in the processor…
- That can be measured with single machine language instructions
  - The Global Descriptor Table (GDT), measured by the SGDT instruction
  - The Local Descriptor Table (LDT), measured by the SLDT instruction

So, the IDT analysis can be fruitful, but there are other critical operating system structures an attacker can look to for detecting VMEs, notably the Global Descriptor Table (GDT) and the Local Descriptor Table (LDT). These tables hold critical variables associated with the operating system and particular running processes, respectively.

Scoopy

- A free suite of VME detection tools by Tobias Klein at http://www.trapkit.de/
- Scoopy runs SIDT, SGDT, and SLDT
  - Checks to see of the IDT is located at an address that starts with 0xc0 (Linux) and 0x80 (Windows). If it does, Scoopy prints a message indicating that it is running in a host machine. Otherwise, it prints that it is in a guest operating system
  - Klein uses identical logic to compare the GDT's location with 0xc0XXXXXX to get a second opinion on the matter
  - And then, he looks at the LDT (only 2 bytes)
    - If LDT is located at 0x0000, it is a real machine, else VMware
- That's three tests for the price of one!

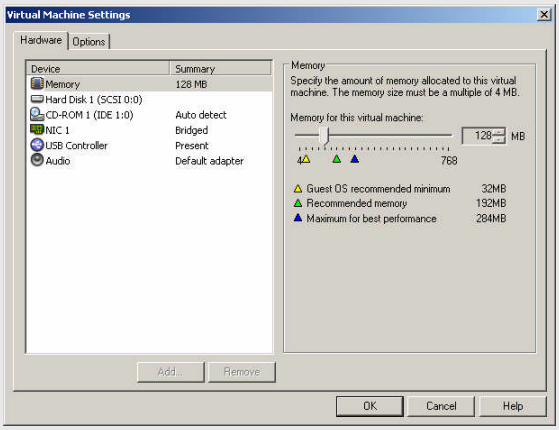
Tobias Klein wrote the Scoopy suite that looks at the location of the IDT (like the Red Pill), the GDT, and the LDT, using the IA32 SIDT, SGDT, and SLDT instructions respectively. Klein observed that host operating systems have an IDT located at memory location 0xc0XXXXXX (which is consistent with the Red Pill, in that it is lower than 0xd0XXXXXX). Scoopy checks to see of the IDT is located at an address that starts with 0xc0. If it does, Scoopy prints a message indicating that it is running in a host machine. Otherwise, it prints that it is in a guest operating system. Klein uses identical logic to compare the GDT's location with 0xc0XXXXXX to get a second opinion on the matter.

Finally, Klein observed that the LDT on a host machine is typically at location 0x0000. On guests, it has some other value.

With three different tests for virtual machines, Scoopy has quite accurate results.

3) Look for VME-Specific Virtual Hardware

- VME introduces virtualized hardware
  - Network
  - USB controller
  - Audio adapters
- Some of these have distinct fingerprints
  - MAC addresses on NICs
  - USB controller type
  - SCSI device type
- Also, anomalies in the way the Guest system clock is updated
- Easy-to-write code, but likely easily fooled
  - Again, using rootkits



 14

A third category of VME detection mechanisms involves looking for specific virtualized hardware, such as NIC cards (with VMware MAC addresses), USB controllers, and audio adapters. Also, some VMEs introduce specific SCSI devices that can be detected. Some VMEs also introduce anomalies on the system clock associated with host<->guest clock synchronization. Oftentimes, inside of a guest, one second doesn't always take one second. Once, when doing a lab for SANS Security 504, a student's clock in a guest operating system didn't change seconds for over 2 minutes! Such anomalies can be detected by attackers.

VME-Specific Hardware Checks with Doo

- Included with Scoopy, Tobias Klein has Doo
  - Linux version of Doo:
    - Simple shell script looks for "VMware" located in:
      - /proc/iomem
      - /proc/ioports
      - /proc/scsi/scsi
      - dmesg command (print kernel ring buffer; holds boot messages and related logs from kernel)
    - Also looks in dmesg output for "BusLogic BT-958" and "pcnet32" - These are known VMware devices
  - Windows version of Doo:
    - Uses Windows Scripting Host to read 2 registry keys associated with SCSI to look for "VMware"
    - Uses WSH to read 2 other registry keys associated with specific Class ID of VMware virtualized hardware

 15

The Doo tool by Tobias Klein (who was also the author of Scoopy) looks for virtualized hardware. The Linux version first combs through various places in the virtual /proc directory looking for the string "VMware" associated with IO, ports, and SCSI. It also looks for messages from the kernel (via the dmesg command) to see if any boot messages mention VMware-specific hardware.

The Windows version of Doo looks for registry keys associated with VMware SCSI adapters and Class IDs of VMware hardware, by checking for:

```
HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port
0\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier

HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port
1\Scsi Bus 0\Target Id 0\Logical Unit Id 0\Identifier

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\
{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\DriverDesc

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Class\
{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\ProviderName
```

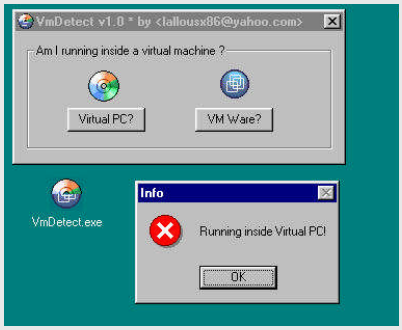
4) Look for VME-Specific Processor Instructions and Capabilities

- Some VMEs introduce “extra” machine-language instructions beyond the standard x86 instruction set to foster Guest-to-Host communication or for other virtualization issues
  - For detection, code could play a non-standard x86 instruction used by VMware, VirtualPC, or Xen into processor to see if it rejects it or handles it
    - A program called VMDetect does this
  - Alternatively, we could look for unusual processor behavior associated with “normal” machine language instructions
    - A program called Jerry does this

One of the most interesting areas of VME detection involves analyzing the processor to see if it has any behavioral characteristics of a virtual machine. Two VME processor anomalies are used today to identify VMEs: looking for support for non-standard VME machine language instructions, and identifying a guest-to-host communication channel.

VME Detection with VMDetect

- VMDetect uses different techniques for detecting VirtualPC and VMware
  - Written by lallous
  - Highly effective
  - Hard to dodge
  - Free



- To detect VirtualPC, VMDetect:
  - 1) Registers its own handler for invalid OpCodes
  - 2) Runs a VirtualPC-specific non-standard IA32 instruction
  - 3) If the processor runs the instruction, it is VirtualPC
  - 4) If the handler for invalid OpCodes is called, it it's a real machine
- VMDetect is available at:
  - http://www.codeproject.com/system/VmDetect.asp

 17

A tool called VMDetect determines if it is running within VirtualPC using a fascinating technique. It does this by attempting to use one of the non-standard x86 instructions that VirtualPC uses for guest-to-host communication. In a "real" (non-VME) computer, the use of a non-standard instruction would create a processor exception, or error, in the running application. The processor would then stop running the program and notify the operating system that an error had occurred in the application's execution. At that point, the operating system would attempt to gracefully shut down the errant program. However, it is possible for a programmer with sufficient foresight to know that there may be situations where a program will cause one of several different types of processor exceptions. Thus, a developer can write software to tell the operating system that it has built-in functions capable of dealing with specific types of errors. These functions are known as "error-handling code," and it is this mechanism that VMDetect exploits to allow it to detect VirtualPC.

By registering error-handling code with the operating system, VMDetect can cause execution to follow one of two possible paths, depending on whether or not the application is running on VirtualPC. If the program is running in a non-virtual environment, then immediately after attempting to use a non-standard instruction, execution will branch to the program's error handler. If the program is running on a VME, then when the program executes the non-standard instruction, execution will continue as if nothing had happened. No error-handling code will be invoked, and thus VMDetect can accurately infer that it is running inside a VirtualPC environment. Otherwise, if an exception is thrown and error-handling code is invoked to deal with the non-standard instruction, VMDetect determines that it is not running in a virtualized environment.

Detecting VMware's "ComChannel"

- Many ease-of-use features offered by VMware require a means for communication between host and guest
  - Shared clip-board
  - File sharing
  - Drag-and-drop
  - Time synchronization, etc.
- VMware accomplishes this by subverting the functionality of a certain x86 instructions
- Detecting the presence of this "backdoor" is the most popular VM detection method in use today
  - Included in VMDetect
  - Also included in Jerry.c by Tobias Klein
  - Also included in checkvm, formerly a part of vmtools
  - Also included in scads of really evil malware

If you're an attacker, and you're even mildly interested in virtual machine detection, you're going to target VMware, because it is, without any doubt, the most widely deployed virtualization environment. So, it should come as no surprise that the most popular VME detection mechanism of all, the most widely used in malicious code today, involves detecting the presence of the VMware by detecting the communications channel VMware uses to send data between guest and host.

This channel, implemented by adding some functionality to the IN machine language instruction on VMware guests, is used for shared clipboards, file sharing, and drag and drop features between guests and host systems in VMware.

VMDetect, Jerry, and checkvm are all tools that implement this technique. We have also located many examples of very nasty malware that uses this technique, described in detail on the next slide.

VMware Detection with VMDetect at the Machine Language Level

```
MOV EAX,564D5868 <-- "VMXh"
MOV EBX,0
MOV ECX,0A
MOV EDX,5658 <-- "VX"
IN EAX,DX <-- Check for VMWare
CMP EBX,564D5868
```

- First, the EAX register is loaded with the "magic value" required for the use of the communication channel ("VMXh")
- ECX is loaded with a command value (0x0A which is used to request VMware version information from the host)
- Any parameters needed for the command (in this case there are none) are loaded in EBX
- Finally, the IN instruction (used for port I/O) is used, which would normally attempt to load data from port 0x5658 ("VX")
- If we are outside VMware, a privilege error occurs. If we're inside VMware, the magic value (VMXh) is moved to register EBX; otherwise it is left at 0
- Based on the version values returned, we can even determine the specific VMware product (Workstation, ESX, GSX, etc.)

The machine language on this slide looks for the VMware guest-to-host channel, by checking for a strange processor property of VMware guests. This code attempts to invoke the VMware guest-to-host communication channel. This channel is actually created by the VME overloading the functionality of a specific x86 instruction, "IN." The IN instruction is normally used to read a byte, word, or dword of data from an I/O port (i.e., from a device, like a modem, connected to the computer). It requires two parameters: the first indicates what register is to be the destination for the data, and the second is a number indicating what port is to be accessed. For the instruction type used by VMware, the port number is placed in the processor register DX before the instruction is executed. VMware monitors any use of the IN instruction, and captures any I/O destined for a specific port number (0x5658) but only when the value of another processor register, EAX, is a very specific "magic" number. In the code listed above, the malicious software is attempting to detect VMware by first loading EAX with this magic value, "VMXh". A specific "command" for VMware to process (in this case 0x0A or decimal 10) is loaded in ECX and parameter data (in this case, 0) is loaded into register EBX. The special port number (0x5658 which also stands for the characters "VX") is loaded into register EDX. With those values in place, the code then calls the machine language instruction IN. On a non-virtual machine, calling this instruction will cause a processor exception (that is, an error) and will trigger specifically provided exception handling code elsewhere within the malicious software. On a guest machine inside of VMware, the IN instruction will be monitored and allowed to succeed without error by VMware which will then change the values in the processor's registers before returning execution to the code. The end result will be that the magic value, "VMXh" will be moved into register EBX. The code then compares EBX to this value and continues on, knowing for certain that it is running in a virtual environment.

Remote VME Detection

- Across the network, VME detection can be accomplished by:
  - Looking for timing anomalies in ICMP and TCP
    - We are experimenting with code to do this; nothing released yet
  - Looking for IP ID strangeness
    - NAT with Windows on a Linux host might have non-incremental IPID packets, interspersed with incremental IPID packets
  - Looking for unusual headers in packets
    - Timestamps and other optional parameters may have inherent patterns
    - Continuing area of research…

The four techniques we've discussed so far detect a VME locally. Other techniques are being analyzed to detect a VME remotely. One technique that has great promise is to look for timing anomalies in the timestamps associated with ICMP and TCP. We've developed an in-house tool for measuring the discrepancies introduced by virtual clocks, which give us an 80% success rate in identifying VMEs remotely.

Also, because different operating systems assign the IP ID values of packets they send differently, it might be possible to identify a VME using NAT networking based on the IP ID values. For example, Windows machines set their IP ID values incrementally, while Linux machines set IP ID values to a pseudo random number. If we have a VMware machine with NAT networking for a Windows guest and Linux host, the observed packets coming from a single IP address could have IP ID values of A, B, C, C+1, C+2, C+3, D, E, C+4, C+5, F, and so on. What we are witnessing is the Linux IP ID values (A, B, C, D, E, and F) intermixed with the incremental Windows IP ID values (C+1, C+2, C+3, etc.)

Another remote VME detection mechanism is to look for patterns in the packets that are set by virtual machines. We haven't yet identified any in our research, but are looking for them.

Thwarting Local VM Detection

- We're going to focus our efforts on the most widely deployed means of detecting virtual machines: detection of the VMware communication channel
- So…
- VMware's idea of a requiring a "magic value" to access the communication channel is great… except for one small problem:

IT'S ALWAYS THE SAME VALUE…

All we can say is "Aaaaaaarrrrrrgh!"

VMware started strong and then faded on the back stretch. They had a really good idea to require a specific value be used in order to "authenticate" to the command channel, but then they USED A CONSTANT VALUE. Essentially, once the bad guys know this value, the VMware communication channel requires no authentication to use, and so there is nothing to keep the bad guys from using it to detect that virtualization is in use.

Sigh.

So, what if we were able to change that value?

Thwarting Local VM Detection

(continued…)

- VMmutate is a small proof-of-concept executable that attempts to change the VMware "magic value" to a user-specified alternative
- Problems:
  - The value must be changed, not only in the VMware executable itself, but also in any programs or drivers WITHIN each guest that rely on the communication channel
  - This can be done by scanning through the guest disk image files, locating and changing instances of the "magic value"
  - But guests are BIG. In big files, simply by chance, you're going to find false positives. So VMmutate must be very careful about what it changes
- Limited success
  - We've gotten some machines to the point of booting, but the keyboard and mouse (and perhaps other things) don't work correctly
  - Networking (surprisingly!) does appear to work correctly
  - Perhaps these machine could just be controlled via SSH like God intended.

In an attempt to provide a means of changing this static value, we have developed a proof-of-concept executable called VMmutate. VMmutate is essentially a high speed search-and-replace tool that is designed to find the fixed "VMXh" magic value used to access the VMware communication channel and change it to a user-specified alternate value.

Because this value is used both within the VME software on the host as well as by various support programs and drivers within the guest, VMmutate must be able to alter this string in both portions of the VME. The disk image files used by VMware are ideally suited to this type of operation with one minor problem: VMware disk images are generally HUGE. When scanning through these files, simply by pure chance, we're going to come across many instances of the magic value that are not involved in the use of the communication channel. VMmutate does a great deal of "context checking" in order to be sure that it isn't changing a value that it shouldn't.

Still, the best we've been able to do is to coax a VM into booting (which was an accomplishment in and of itself… did you know that really messed up VMs actually make POST beeps?) but with severely limited functionality (i.e. no keyboard, no mouse). But there's always SSH!

Thwarting Local VM Detection

(continued…)

- Another way to skin this cat…
  - VMware Configuration Options – used in the guest's .vmx file
    - isolation.tools.getPtrLocation.disable = "TRUE"
    - isolation.tools.setPtrLocation.disable = "TRUE"
    - isolation.tools.setVersion.disable = "TRUE"
    - isolation.tools.getVersion.disable = "TRUE"
    - monitor_control.disable_directexec = "TRUE"
    - monitor_control.disable_chksimd = "TRUE"
    - monitor_control.disable_ntreloc = "TRUE"
    - monitor_control.disable_selfmod = "TRUE"
    - monitor_control.disable_reloc = "TRUE"
    - monitor_control.disable_btinout = "TRUE"
    - monitor_control.disable_btmemspace = "TRUE"
    - monitor_control.disable_btpriv = "TRUE"
    - monitor_control.disable_btseg = "TRUE"

It turns out that there are a number of undocumented "features" in VMware that can be used to mitigate the "detectability" of VMware by various means. VMware uses a text file with the extension .vmx to set various configuration options. These options control the behavior of various sub-systems and components of the guest VM: the CD-ROM, floppy disk, networking, etc…

By placing these configuration options in the .vmx file associated with a particular guest VM, the VM detection techniques used by Jerry.c (and most VM detecting malware), the RedPill, and Scoopy are nullified.

Thwarting Local VM Detection

(continued…)

- Problems!
  - VMtools and many other ease-of-use features are ***BROKEN***
  - These must be applied to a guest VM when it is OFF, not simply sitting at a snapshot
  - These are ***UNDOCUMENTED*** features, and we're not entirely sure what other side effects they may have

Essentially, these configuration options break the communication channel between guest and host not just for the program trying to detect the VM, but for ALL programs. That means that basically all of the ease-of-use features of VMware will no longer work: no drag-and-drop, no shared folders, no shared clipboard, no time synchronization.

Also, having played around with these options, there is one "gotcha" that we'll warn you about: you can only apply these options to a guest that is actually turned off. If you apply them to a machine sitting at a snapshot, firing that machine up will overwrite any changes that you've made.

Finally, because these are undocumented features, there may be other side-effects that we're unaware of. Use these settings at your own risk.

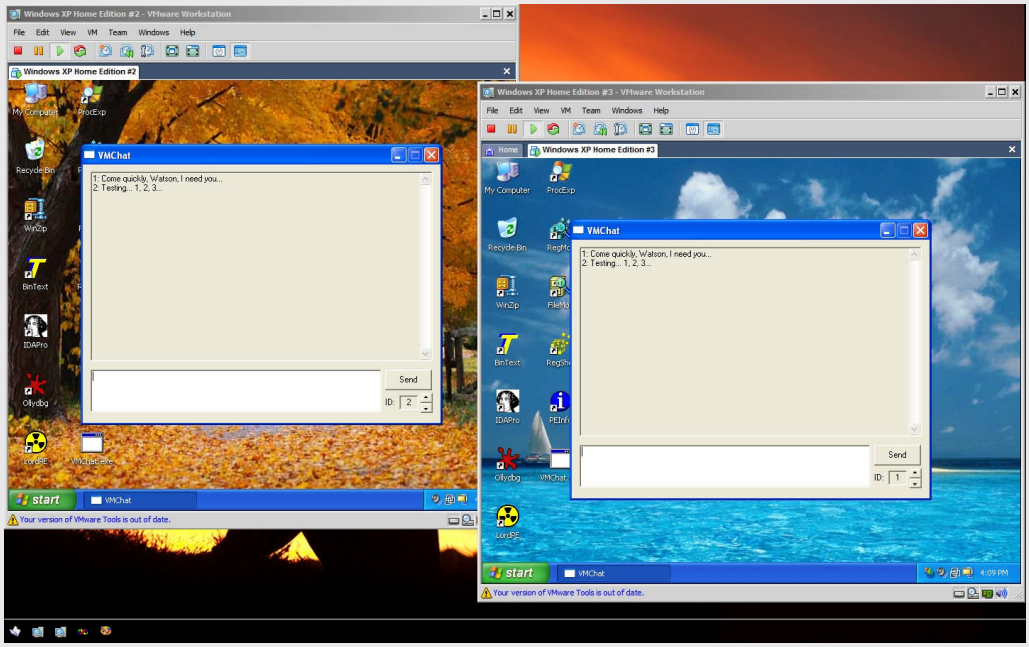
Finally… a teaser

- We talked a bit before about what we called “questionable usage patterns”
- These are VME deployments that, we believe, overly rely on the isolation of guest machines from one another
- Here’s why we feel that way…

When we introduced some of the forces behind the bad guy’s drive toward developing virtual machine detection techniques, we talked a little about what we called “questionable usage patterns,” or VME deployments that rely on virtualizations guest-to-guest isolation to provide security.

In many cases, this isolation isn’t all it’s cracked up to be… as the next slide will illustrate.

Isolation?



Just sayin'.... you know... thinking out loud. ;-)

Thank you!

- Thank you for stopping by and sharing your evening with us
- Questions, praise, cash donations:
  - tom@intelguardians.com
- Enraged hate mail, slanderous insults:
  - ed@intelguardians.com

Thank you for stopping by and we hope you found the presentation worthwhile.

If you didn't, it's all Ed's fault.