Global Information Assurance Certification Paper

Malware Analysis: Environment Design and Artitecture

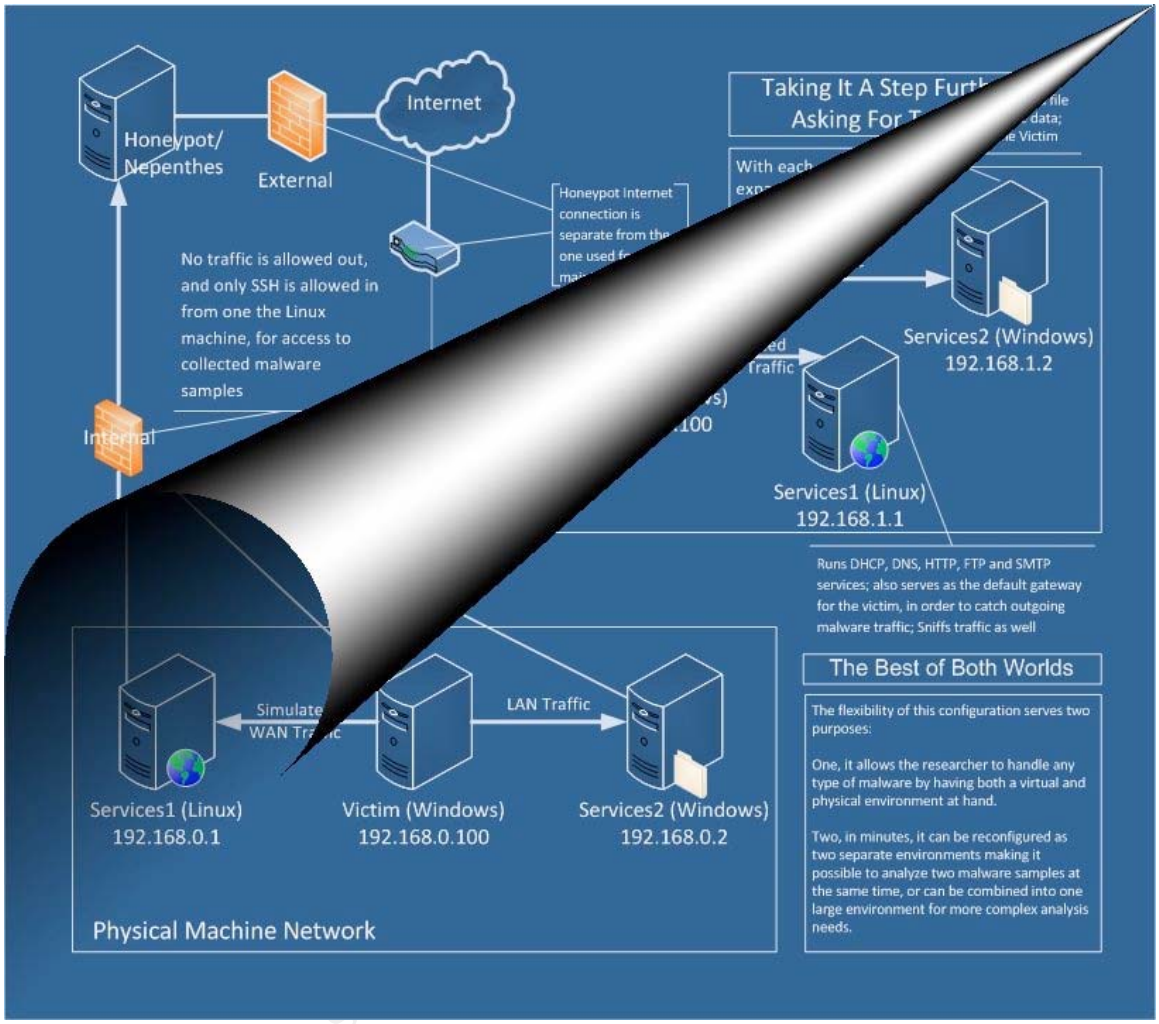
GCIH Gold Certification

Author: Adrian Sanabria, adrian.sanabria@gmail.com

Adviser: Rick Wanner

Accepted: January 18th 2007

Blueprints for a new age in Malware Analysis



Abstract

At the software level, tools and methods for analyzing, detecting, and disabling malware have been documented and employed for several years now. However, the design and architecture of malware analysis environments does not often get publicly discussed. To be sure, commercial anti-virus vendors and high-profile researchers most likely employ the use of highly customized and specialized environments to explore the goals and inner workings of malware quickly and efficiently. The average security researcher/analyst however, rarely experiments beyond the use of an isolated virtual machine to quarantine the malicious intent of a virus or trojan.

The goal of this paper is to discuss the architecture and design necessary to create an effective malware analysis lab environment, and to explore possibilities beyond the traditional two or three system VM-based lab.

Table of Contents

1. Introduction

The malware landscape is changing. As the potential for profit in computer crime has increased, so have trends towards more complex, sophisticated, and higher quality malware. In order to be able to analyze these new generations of malware, security researchers' lab environments have needed to become more complex as well. This paper focuses on providing the new and experienced security researcher with the information necessary to build a capable and effective lab. This paper also focuses more on the system architecture, rather than analysis tools, as that aspect has been documented more thoroughly already [1] [9].

1.1 Background

In the olden days, analysis had to be done with shell commands, built-in system utilities, and a text editor. Of course, back then, the attack surface was small, and there weren't many places for malware to hide.

As malware really began to hit its stride, VMWare's virtual machine technology started to gain in popularity among security analysts. Researchers could make a snapshot or backup of a virtual machine, and proceed to hack it, infect it, and trash it to their heart's content. Afterwards, they could restore the good copy in just a few short minutes. This process could be repeated over and over, and streamlined analysis in a big way.

However, virtual machine detection appears to be[1] trivial nowadays [2] [3] [4] - a fact of which some malware authors are well aware, and take advantage of [5]. Knowing that researchers use virtual environments to analyze their code, some malware authors now instruct their creations not to run, or to run differently within these environments. The goal is to make it more difficult for researchers that employ the use of virtualized environments to analyze samples of malware. This creates a dilemma that, in large part, inspired this paper. What kind of lab does a security researcher now need to handle malware analysis in the present and future?

1.2 Scope

This paper will not go into the specifics of disassembling and reverse-engineering malware. Its purpose is to define and explore the components, design, and architecture necessary to assemble malware analysis labs of varied sizes.

1.3 Assumptions

This paper assumes the reader is at least familiar with the following principles (not necessarily possessing expert hands-on experience with them):

- Computers
- Malicious software
- Software analysis
- Reverse engineering

Network security (mainly firewalls and traffic sniffing)

[1] Note subtle use of foreshadowing…

2. Goals and the Nature of Malware

Most malware aim to achieve one of three things: to steal something, destroy something, or compromise a system to achieve some higher goal. Table 1 shows how each of these properties affects the design and/or architecture of a lab.

2.1 Malware Defined

There are many different ways to classify malware. Most, such as Antivirus vendors, tend to classify by intent (Trojan, worm, mailer, etc…) and several aspects of severity (damage potential, potential of outbreak, and actual outbreak reports). These metrics are usually averaged to create an overall risk rating. To the end user, malware is usually just software they didn't want or ask for, doing nasty things to them or their computers. However, a great deal of modern malware doesn't want to announce its presence to the average end user.

Enter the none-too-surprising evolution of malware and spyware – malicious software silently installed, and secretly tucked away for maximum profit and reusability. Organized crime's presence in the computing and Internet age has guaranteed that malware would get the same treatment as other industries favored by the Mafioso. Obviously, credit isn't entirely due to organized crime, but the result for the good guys was the same: the need for a new method of identifying and classifying malware according to how difficult it is to detect. Joanna Rutkowska has proposed such a method, which she calls *Stealth Malware Taxonomy* [6]. The intent is not to say this categorization should replace currently used categories, but that there is another set of criteria to consider when analyzing malware.

Table 1: Brief Overview of Joanna's Stealth Malware Taxonomy

Malware Type	**Stealth Characteristics**	**Analysis Considerations**
Type 0 Malware	Does not use undocumented methods to hide	Most standard malware falls under this category; Use traditional tools to analyze
Type 1 Malware	Modifies constant resources to hide itself (by patching executables, inserting into BIOS, etc…)	Compare hashes of running memory with equivalent values on disk; Digitally sign code
Type 2 Malware	Modifies dynamic resources to hide itself (using sections of data within memory, for example)	Currently no good solution; Can't compare hashes of application data, as it is constantly changing.
Type 3 Malware	Hides itself where the operating system cannot see it at all, like a hypervisor.	Currently no good solution; Being nearly (if not totally) undetectable from within the Operating System, detection, prevention and analysis would have to be done at the hypervisor level – outside of the OS. The

		only hope for analysis is to compare the timing of instructions executed before and after type 3 malware is introduced.

This is all relevant to malware analysis and lab architecture, because there always exists the opportunity to specialize a lab or PC environment for the analysis of a specific type or class of malware. One of the most common recent examples is malware that refuses to run in virtualized environments, as such environments are often equated with malware analysis. A researcher hoping to analyze such a class of malware must take this behavior into account and make the necessary changes to their lab design.

The result is several ways to categorize malware, and several opportunities to specialize an analysis lab. This is explored in table two.

Table 2: A few examples of how goals and specialization can drive lab design

Malware Type	Lab Specialty	Goal
Mass Mailer/Spambot	Tracking sources and destination of spam	Identifying bot owner; and means of infection
Rootkits/Spyware	Discovery of malware using different levels of stealth	Prevention and/or Detection
All kinds	Comprehensive malware analysis	Analyze whatever pops up
Worm	Network traffic capture and analysis	Identifying means of propagation
Destructive	Local resource monitoring	Preventing infection
Spyware/PhoneHome	Network traffic capture and analysis	Identifying data recipient, means of transmission, and type(s) of data targeted
Bots	Static/Behavioral Analysis; Network traffic capture and analysis	Identifying bot owner and means of infection

Combining two methods of categorization in table three, shows some of the combinations possible. For example, Type 2 malware could be spread and infect hosts as a worm, or Type 3 malware, the mother of all rootkits could be used as spyware.

Table 3: Traditional and Stealth Categorization Combined

Traditional Types	Type 0	Type 1	Type 2	Type 3
Mass Mailer/Spambot	X	X	X	X
Rootkits/Spyware	X	X	X	X
Worm	X	X	X	X
Destructive	X	X	X	X
Spyware/PhoneHome	X	X	X	X
Bots	X	X	X	X

2.2 Terms and Technologies

2.2.1 Malware Source

The source of infection by malware, whether intentional or otherwise

2.2.2 Honeypot

In the world of information security, a system intentionally set up to attract both malicious software and/or attackers in order to study them.

2.2.3 Entrance Point

The point of infection by a particular example of malware. The entrance point doesn't matter for most kinds of malware, as most don't care, as long as they get the chance to infect a system *somehow*.

2.2.4 Virtual Machine Environment (VME)

Virtualized environments that run as an application, which allow for one or more operating systems to be run as if they were installed on dedicated hardware.

The “operating system within an operating system” concept also creates the necessity for two more terms:

Host OS – The operating system that runs the VME, also sometimes referred to as the VME host.

Guest OS(s) – The operating system(s) that run within the VME host.

2.2.5 Virtual Machine Monitor (VMM)

The VMM is the component that controls the VME, and serves as the point of interaction between guest OSs and the host OS.

2.2.6 Dedicated Virtualization Server

A server functionally or physically dedicated to the task of running Guest OSs. VMWare ESX is an example of the latter, as the host software is custom-designed only to run Guest OSs as quickly and efficiently as possible. ESX benefits from the performance advantages of functioning as both the VMM (Virtual Machine Monitor) and the host operating system. However, a plain server running Windows 2000 Server that only has base necessities and VMWare Server installed could also be considered a dedicated virtualization server, although it won’t enjoy the benefits of an OS customized and dedicated to the task of virtualization.

2.3 Goals

The goal of this paper is to get a reader that may be interested in creating or improving a lab, thinking about problems, options and solutions relevant to such a project.

2.3.1 Guidelines

Following are some of the most important things to keep in mind when designing and implementing a malware analysis environment:

- Simplicity – Each added bit of complexity can make it more difficult to maintain.
- Containment is paramount when designing an environment that may test the digital equivalents of plagues and super flues. Maintaining control is preferred as well, but cannot be guaranteed when dealing with new malware specimens. Containment is the safety net when control is lost.
- A flexible environment is essential. One that is too fragile, or has too much downtime is of little use to a malware researcher.

2.3.2 Desired Results

Several questions must first be considered and answered to determine what one hopes to gain from their lab.

- How far to take the analysis - analyze network behavior, local behavior, or all of the above?
- Will the analysis be:
    - Informal (not shared with other people)
    - Partially formal (share w/ public and community)
    - Formal (write up for public on entity's behalf, submission to company, government, 3rd party, virus bulletin, AV Vendor)

- Will you perform only static (code) analysis, or behavioral analysis as well?
- Do you plan to partially or fully reverse engineer malware?
- Will the lab be specialized in a specific area of malware analysis (see Table Two)?
- Learn only enough to better protect yourself or your organization?
- Will the results be used in a legal case?

2.3.3 Requirements

Physical and/or Financial Constraints

Will the size of the lab include several machines, or just a laptop? A researcher may need to do analysis on the road, could do all of it in a fully funded data center, or could employ a combination of both. A researcher without a large (if any) budget may be restricted to a single machine for analysis.

Malware Analysis

To be able to analyze any malware that comes along should be the goal of any analysis lab, even more modest ones. Until recently, this was a simple affair that could be accomplished by a laptop with a large amount of RAM and available disk space. See section 3 and 4 for more details on how and why this has become more complicated.

Swift Recovery

No one likes their lab to be down for too long. Traditionally, recovering a computer system to an earlier state would be a tedious, time intensive operation. In the

past five years, however, VMEs have become popular in malware analysis due in part to the ease and speed of recovery possible with these environments. VMEs are not the only option, however. Table 4 shows possible recovery options and the pros and cons related to them.

Table 4: Recovery Options

Method	**Pros**	**Cons**	**TTR[2]**	**Complexity**	**Action**
System Backup Software (e.g. Ghost)	Relatively Quick	Costs money, not quick	30 minutes	Medium	**Restore from Backup**
System Backup (to Tape)	Works	Slow, costs money	~2 hours	Medium	Restore from Backup
System Backup (using RAID)	Works	Slow, costs money	~2 hours	Medium	
Virtual Machine and Emulation Environments	Very Quick	Can be detected	Minutes	Easy	Restore from Snapshot

[2] Time to Recover (TTR)

Windows System Restore	Quick	May not remove malware	Minutes	Easy	Restore from Snapshot
Windows "Live" CD	Very quick and easy	Untested - may not be feasible	Boot time of the system	Very Easy	Reboot
Multiple Isolated Labs	No wait	Expensive	Instant	Very Easy	Switch environments

As an example, say a researcher's job is to perform malware analysis in response to outbreaks. In a scenario such as this, the only viable option may be to have multiple isolated labs, and a staff that restores them as soon as a researcher completes analysis.

Most of the options in Table 4 represent more modest methods, however. Many are currently in use, but none is used or preferred more than the VME methods. Some VME products have a feature that allows the user to create a snapshot of their current environment, and revert to that snapshot at any time. This can also be done manually in most (if not all) VME products by simply creating a backup or copy of the VME files at a particular point. To restore, the researcher only has to shut down the VME, delete the infected copy of the files, copy the clean snapshot over, and start the VME back up. In most cases, this can be done in less than 10 minutes.

Other methods listed in Table 4 represent untested theories, such as the Windows “Live” CD concept. Although it is technically feasible, it could face detection problems, as with VMEs. With this Windows “Live CD”, resetting everything to defaults would be as simple as rebooting, as with Linux Live CDs. Research [7] has shown that a Windows Live CD is possible [8], although it too has its problems. It may still be detectable, and the legality of hacking Windows to run as a Live bootable OS is unclear.

3. Design: Lab Components and Considerations

3.1 Lab Components and Their Roles

3.1.1 "The Victim"

Whether a physical host or a virtual one, very careful attention must be paid to setting up the computer that will be infected. The level of scrutiny is at the researcher's discretion, depending on their goals. A pre-built image is recommended, and should be as stripped-down and basic as possible, only adding software and/or services after a baseline has been completed with a bare system. *The primary purpose of "The Victim" is to study malware behavior while it is running.*

3.1.2 Primary Lab Services (Server1)

Unless only very basic static or behavioral analysis is going to be performed, at least one other system is necessary to run support services for the victim. Some of these are necessary for mundane reasons, as with a DHCP server, but others will have more clandestine roles. Most of the necessary support services can be provided by a server running Linux, BSD, or other UNIX-like operating system. Most, if not all malware that will pass through the average researcher's hands will not infect these operating systems, hence their desirability. However, it is still advisable to keep an image or backup of any support systems just in case. *The primary purpose of "Server1" is to provide support services to supplement behavioral analysis on "The Victim". A secondary, or alternate primary purpose is to perform manual or automated static (code) analysis.*

See Table 5 for a list of services that may be useful in a malware analysis lab.

3.1.3 Secondary Lab Services (Server2)

In all examples, the system providing secondary services is Windows-based, and provides network shares, but these roles are completely at the discretion of the researcher. See table 5 for a list of services that may be useful in a malware analysis lab.

Table 5: Support Services

Service	Useful for…	Run on…
DNS	Redirecting malware attempts to access sites on the Internet	Server1
DHCP	Necessary for lab systems to get an IP, and to set Server1 as a gateway, so that malware attempts are redirected and trapped	VMHost or Server1
HTTP	Capture redirected malware traffic	Server1
FTP	Capture redirected malware traffic	Server1
SMTP	Capture redirected malware traffic or spam attempts	Server1
SMB (Windows File Shares)	Bait malware with sensitive files, seemingly left unguarded on “corporate” network shares.	Server2

3.1.4 VME Host and Dedicated Virtualization Server

In all the examples, the VME Host is isolated from the Virtual Machine Network, as described in Richard Wanner’s work on reverse engineering malware [9]. As a precaution,

the VME Host should also be isolated from any other networks as well. The idea is to avoid infecting and having to rebuild more systems than necessary, as this could be an enormous time constraint for a busy or full-time researcher. The Dedicated Virtualization Server is only different from the VME Host in that:

1. It is more robust, and can support more virtual machines
2. It is a dedicated resource, whereas the VME Host could be the researcher's workstation, personal laptop, etc…
3. It could also be completely dedicated to the task of hosting virtual machines, like the VMWare ESX product.

3.1.5 Network Hub

The purpose of using a network hub is necessary for the simple reason that it greatly simplifies the task of sniffing and recording as much network traffic as possible. This is a crucial step in analyzing and reverse-engineering malware.

3.1.6 Honeypot

Different researchers have different sources for obtaining malware, and their own reasons for analyzing it, but Honeypots can be a novel (and safer) way to collect malware for research and analysis. Automating the mundane and repetitive tasks in malware analysis is the next logical step.

There are a few ways to deploy a honeypot as a malware source for the lab. In a *manual* scenario, the Honeypot is isolated from all other internal networks, connected to the Internet solely by a firewall and a router. Malware samples are collected manually using local storage, which will then

be used to manually inject the malware into the lab, with the specific entrance point being the “The Victim”. In light of recent (to this writing) malware incidents that have taken place [10], this is an ironically appropriate entrance point for malware.

In an *automated* scenario, one of the service machines in the lab could check the Honeypot on a regular basis, and begin analysis as soon as a piece of malware is discovered. SSH would be the ideal method for transferring the malware files, as it is simple to implement through a firewall, secure, and easy to deploy on nearly any operating system. A very simple *deny all* rule would be put in place on the firewall separating the honeypot from the lab, with just one additional rule, allowing only SSH (TCP 22) in. Do not allow traffic to be initiated out (from the honeypot into the lab network).

3.2 Virtual Machine Environments

Will the ability to detect VMEs from within ruin a good thing for malware researchers?

3.2.1 Introduction: Why Virtualization is Important

Virtualization is an important tool for malware researchers and as such, is a large focus in this paper. The fact that some samples of malware are now refusing to run in researchers’ labs is an important issue, and one without a simple solution. The aim of this section is to dissect the problem and clarify the solutions available.

Apologies, if the Matrix analogy is getting old, but it really is a perfect example, and a very effective way to explain the relationships between hosts and guests in the

world of VMEs. Most important to VM detection is the difference between different types of VMEs, specifically between *native virtualization/ paravirtualization* and *emulation*. This section aims to explain these differences, and why they matter.

3.2.2 Problems and Considerations

Detection - VMEs are fantastically useful tools for malware researchers. This is no secret, and most malware authors are likely aware of this fact. At least a few are aware of this, as there several examples of malware in the wild today that will refuse to run in a VME.

At first, it may seem strange that all current malware doesn't have VME-avoidance code built in now, but there are potential reasons.

1. The target could be a VME, or could include a VME. With thousands of companies using dedicated VMEs to consolidate their data centers, instructing malware not to run in these environments would exclude a large number of tempting targets.
2. The detection code is not a good fit for the malware payload.
3. Script kiddies have not yet found a place to copy the detection code from.

Cost – Overall, it is usually much less expensive to deploy a virtualized lab, than deploying one-to-one physical systems for all the same reasons large organizations have been doing so in their data centers. Additionally, many of the VMEs explored in this paper are free to download and use.

Flexibility – Probably the most important reason researchers employ the use of VME's the ability to restore virtual machines to their original form in mere minutes is essential.

Network Isolation – VMWare is specifically well suited to allow for network isolation, as described in Richard Wanner's work on reverse engineering malware [9]. Isolation can also be achieved using VLANs or firewalling techniques also.

3.2.3 VME Technologies

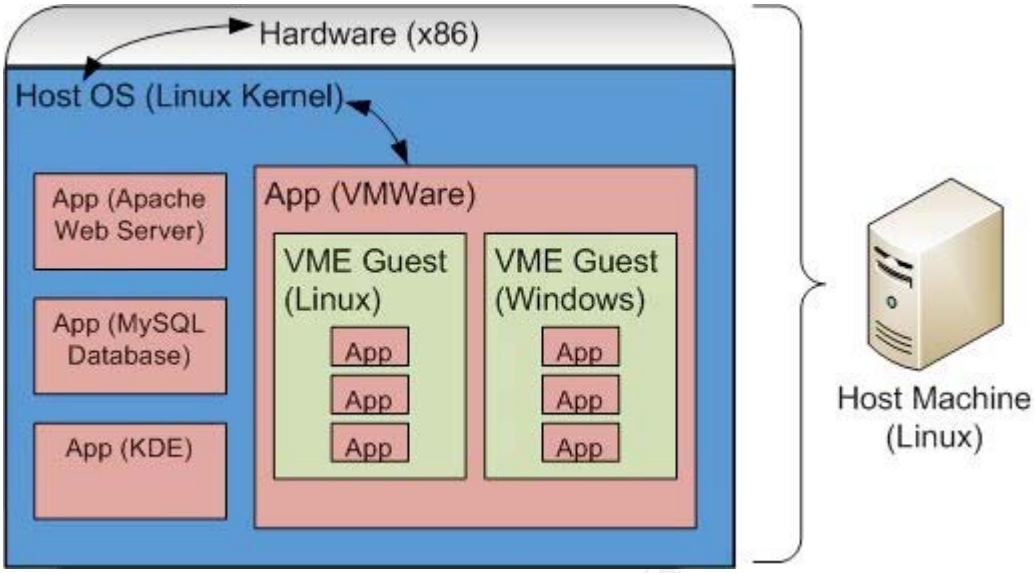
3.2.3.1 Native Virtualization

In Native Virtualization, the VMM executes guest code on the underlying hardware. Because the host and guest operating systems are sharing the same hardware, certain resources must be relocated by the VMM to prevent conflicts. One of these resources is the interrupt descriptor table register (IDTR). When this resource is relocated by the VMM, the address of the table changes. Using the SIDT instruction, one could write some simple code that will return the location of this table, and thus show whether code is being executed inside the matrix (inside a VME guest), or in "the world of the real" (within the host OS).

Pros: Fast; Easy; Flexible; Convenient

Cons: Easy for malware to detect; VME host software is limited to running on x86 architectures.

Figure 1: Native Virtualization



"Apps are red, hosts are blue.
Malware goes into a green guest,
And hopefully does not eschew…"

3.2.3.2 Paravirtualization

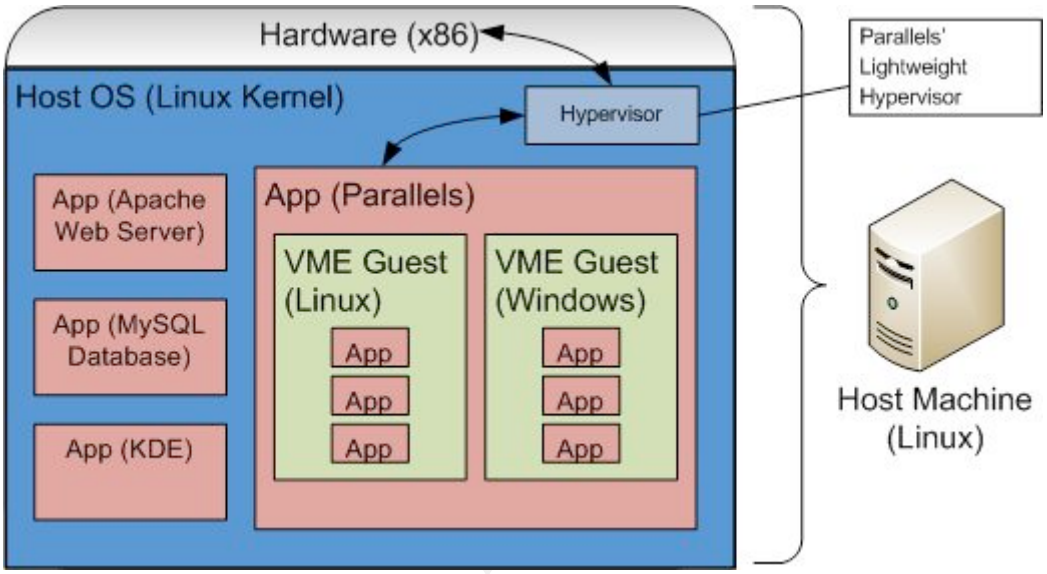
Paravirtualization is similar to Native Virtualization, except that there is a unique relationship between the host and the guest. The host presents an interface, similar to a software API, to the guest. This interface is called an ABI [11] (Application Binary Interface), is used by the guest to speak indirectly to the hardware.

Pros: Is claimed to be potentially even faster [12] than Native Virtualization, due to the unique "shortcut" paravirtualization provides for the guest.

Cons: The guest must be modified to work with the host's specific ABI. This generally means that paravirtualization is an approach that generally won't work with commercial operating systems, such as Microsoft Windows. The fact that

other virtualization approaches require no changes to the guest OS make it unlikely that commercial OS vendors will support paravirtualization products in the future.

Figure 2: Paravirtualization



3.2.3.3 Native Virtualization and Paravirtualization Detection

The IDTR Detection Technique

Tools and code demonstrating VM detection techniques are freely available. Joanna Rutkowska's *Red Pill* [2] is probably the most well known of these, though Tobias Klein's *Scoopy* [3] tool is a bit more informative. When Red Pill.exe is executed within an OS running directly on hardware, Red Pill informs us that we are "Not inside the Matrix". When executed within an OS running in a VME like VMWare, Red Pill informs us that we are, indeed, "Inside the Matrix" (see figure 3 and 4). Malware authors have taken advantage of the fact that VM detection can be done

with a line, or just a few lines of code. It is increasingly common to find malware that will refuse to run in virtualized environments, as their authors know that VMEs commonly used by malware researchers.

Figure 3: Red Pill running on a Windows 2000 VMWare guest

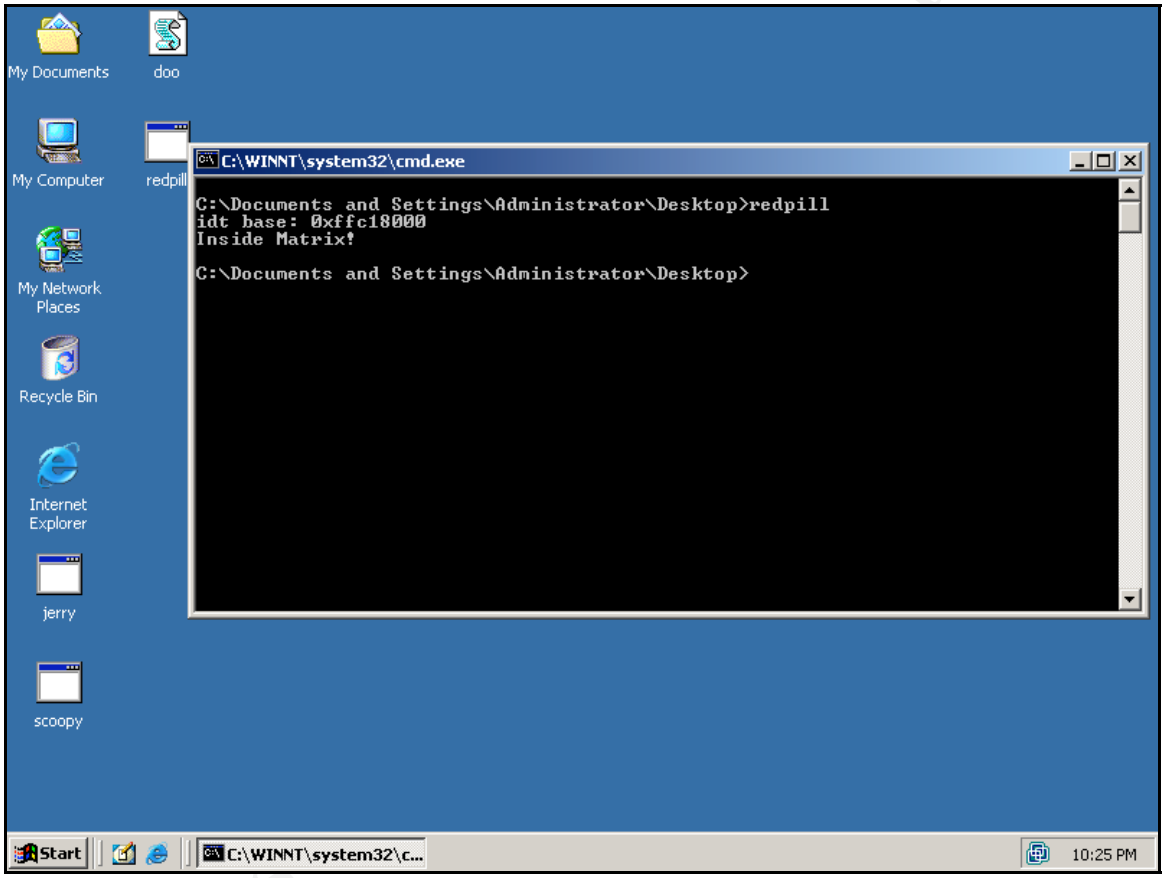
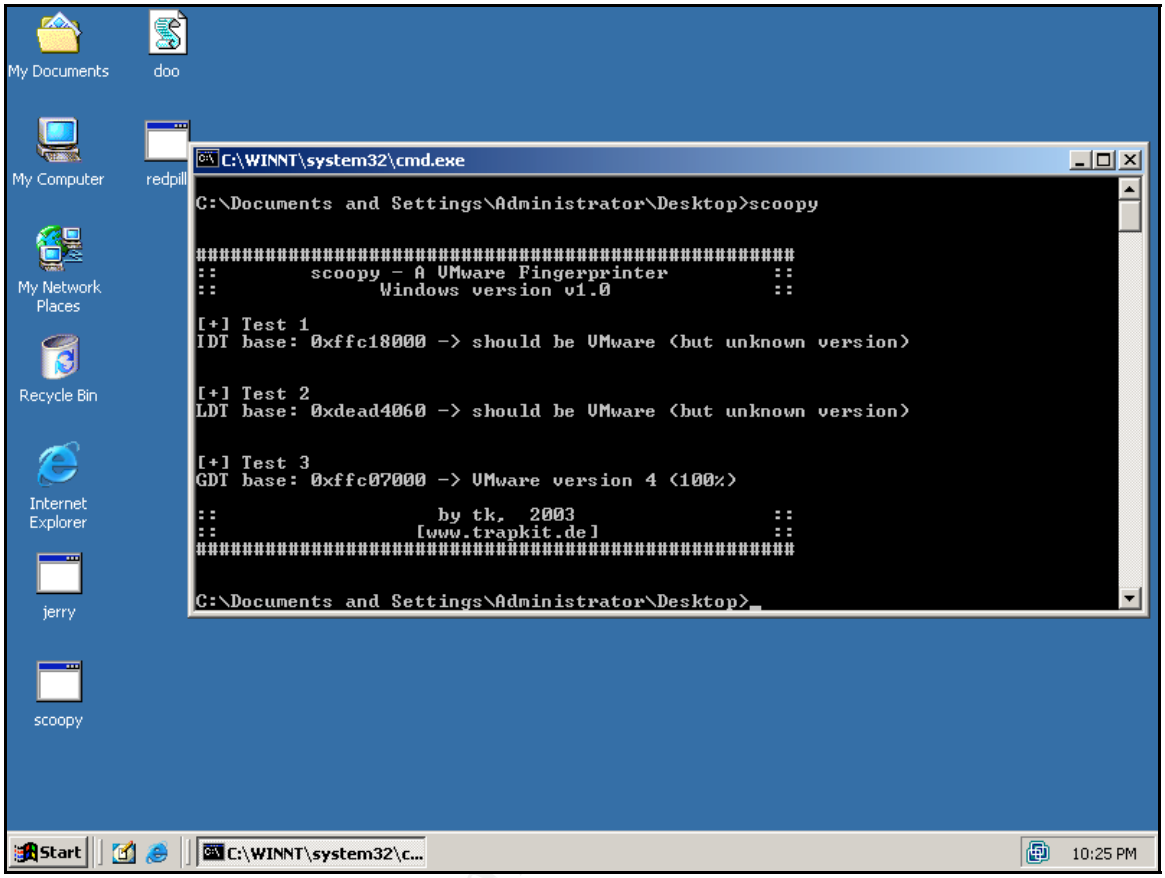


Figure 4: Scoopy running on a Windows 2000 VMWare guest



To counter this, it is possible that a VME could fake the results of a query for IDT values, but it is unlikely that commercial vendors would take much interest in making these changes. It is also not clear whether such changes would cause detrimental effects on operating systems running within the modified VME.

Other Detection Techniques

Most commercial VMEs create many artifacts that allow for easy VM detection. One such example is Tobias Klein's *Doo* vbscript, included in the *Scoopy Doo* release. This vbscript simply looks for VME artifacts in the Windows registry. These are extremely easy to find if a VME toolset, such as

VMWare Tools, or Parallels Tools have been installed on the Guest OS. Even if VME toolsets have not been installed, artifacts can still be found, as *Doo* shows. *Doo* specifically looks for the names of hardware components, which usually contain the word “virtual” or the name of the VME vendor. Tobias Klein calls this a “very stupid” method for VM detection, but it is effective all the same when one can just look at the video card, and see that it is a “VMWare SVGA” device.

Further detection techniques are more thoroughly explored in *On the Cutting Edge: Thwarting Virtual Machine Detection*, a paper by Ed Skoudis and Tom Liston. However, keep in mind that most of these alternative detection techniques are based on commercial VME vendors’ tendencies to advertise themselves whenever and wherever possible.

3.2.3.3 Emulation

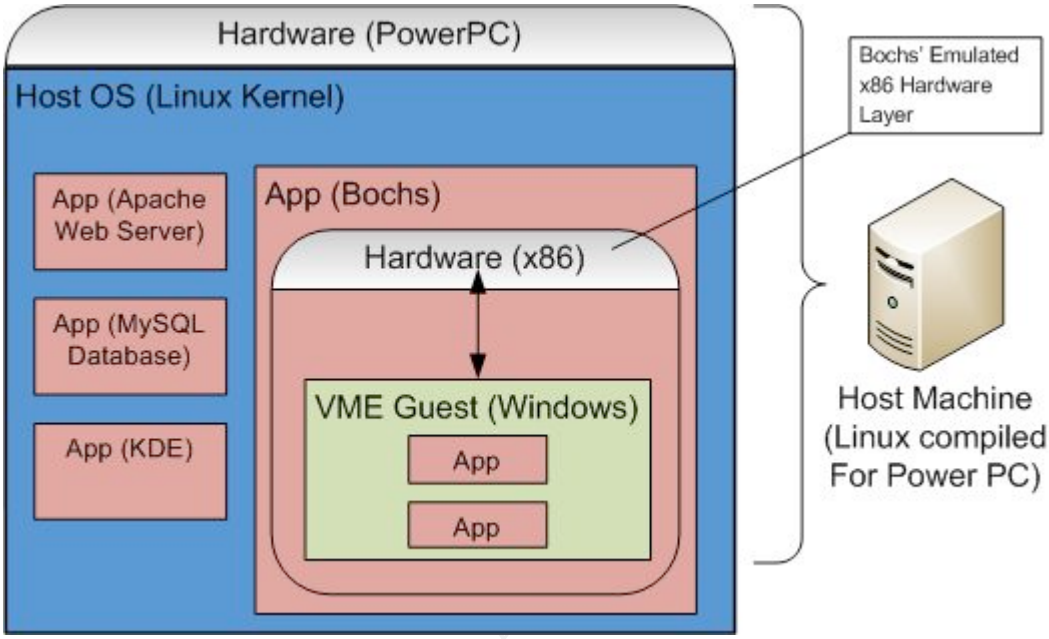
Emulation is a different matter altogether. Computer emulators emulate the underlying hardware using *code*, rather than by sharing the actual physical hardware. As a result, SIDT/IDTR detection techniques *do not work within emulated VMEs*. Another advantage of emulation is that the emulated hardware can potentially run on top of any other hardware architecture. For example, Bochs running on MacOS X could run x86 versions of Windows XP.

Pros: x86 emulators such as QEMU and Bochs can run on any architecture the code is ported to; Can evade current detection techniques

Cons: Emulation is generally much s-l-o-w-e-r than native virtualization or paravirtualization. If planning to use an

emulator such as Bochs, be sure to test it briefly before choosing it, to ensure that performance is within acceptable levels.

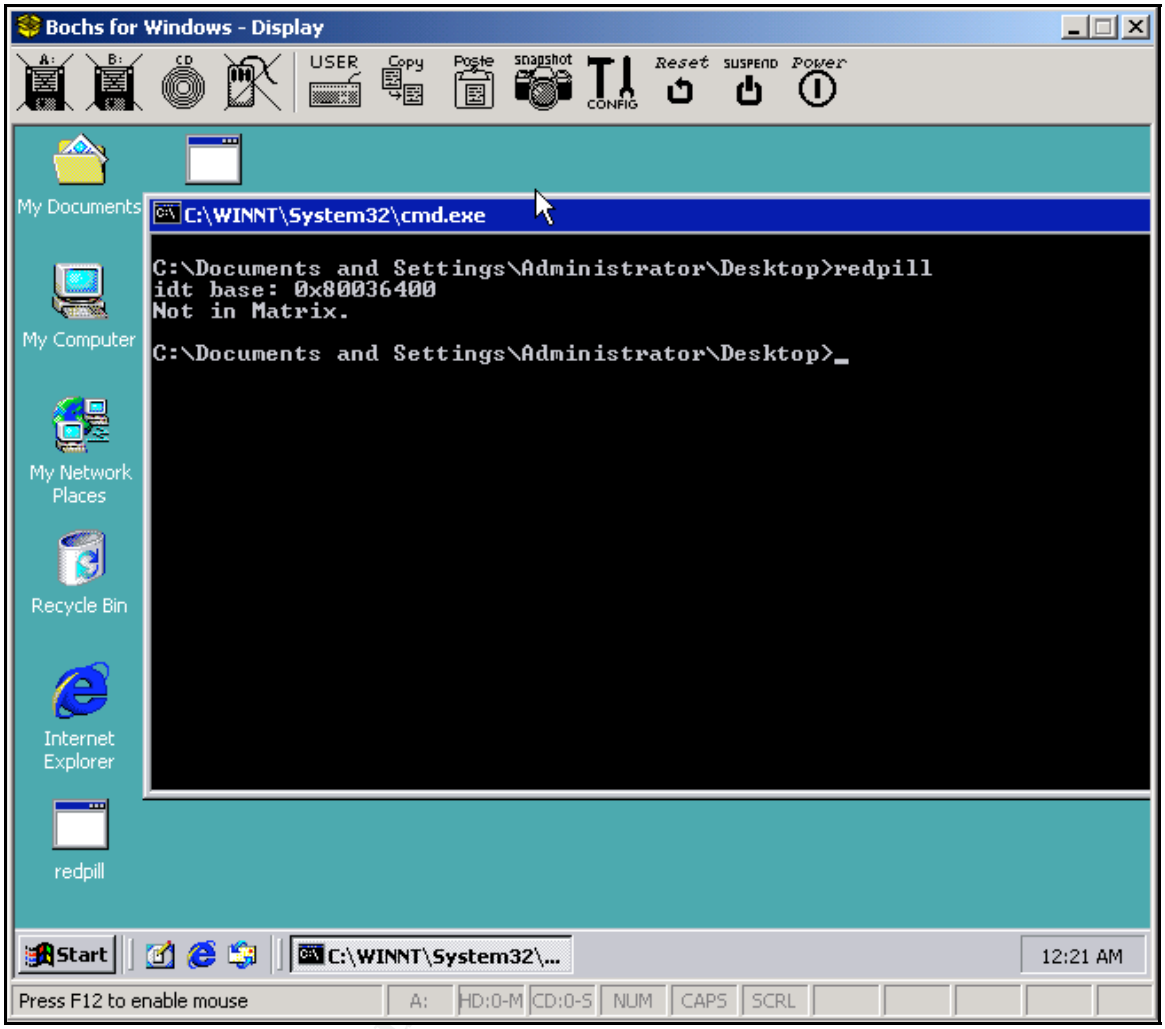
Figure 5: Emulation



3.2.3.4 Emulation Detection

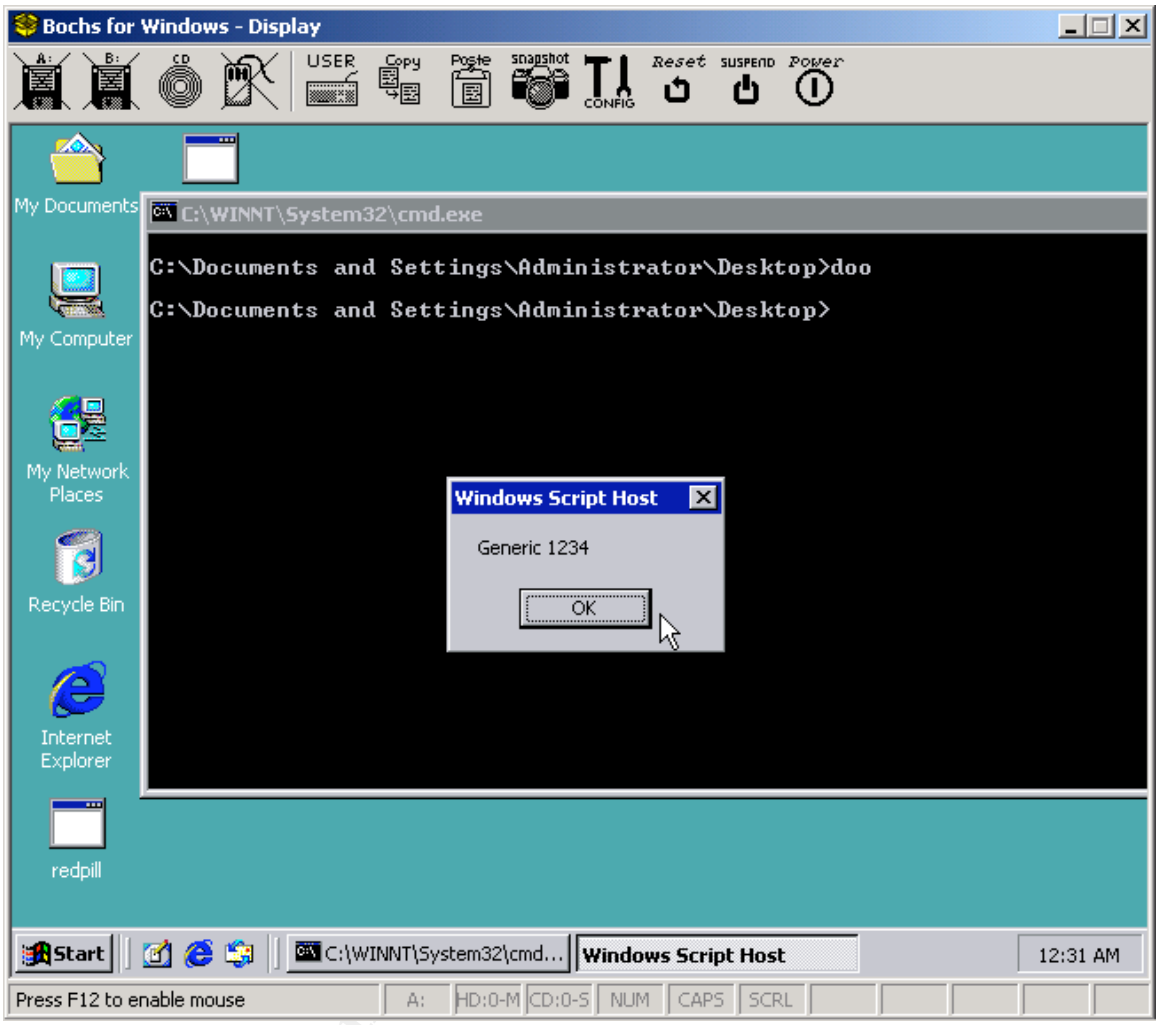
Execute Red Pill.exe within an OS running in an emulated VME like Bochs [13] or QEMU [14], and the result returned is “Not inside the Matrix”.

Figure 6: Red Pill running on a Windows 2000 Bochs guest



So, the IDTR technique doesn't work, what about other techniques? Tobias Klein's *Jerry* tool doesn't work, as it tells us our OS under Emulation is a "Native System". When we run *Doo* however:

Figure 7: Tobias Klein's "Doo" script running on a Windows 2000 Bochs Guest



Uh-oh, it seems our cover is blown. However, remember this is a non-commercial open-source emulator, so there may be hope. A quick look at the Bochs configuration file reveals an elegant solution. A simple config change allows the user to control the hard drive label:

```
ata0-master: type=disk, path="hd10meg.img", cylinders=306,
heads=4, spt=17, Model="FUJITSU MHK2060AT"
```

Note this change must be in place before Windows is installed. Windows appears to cache this information in the registry, so a change to the Bochs config after Windows is already installed won't be enough.

So far, it looks like all current VM detection techniques used by malware fail. Also, all the proof-of-concept detection tools tested failed (after minor changes). See Appendix A for a full results listing of VME detection tools within different environments.

At this point, it looked like Emulators were nearly undetectable. Then Peter Ferrie released a paper detailing several effective methods of detecting an emulator from within [15]. Although Ferrie points out several ways to perform detection, most, if not all, appear to be fixable with somewhat modest code changes. There exists the possibility that, with a few bug fixes to the main branches, or with an Information Security-specific fork, the emulators reviewed in this paper could be made undetectable.

3.2.3.5 Dynamic Recompilation

This method is similar to emulation, in that it can be used to run guests that require different processor architecture than what the host is using. In other words, MacOS running on a PowerPC chip could run 32-bit x86-based Windows XP. Rather than recreating hardware with code, however, in dynamic recompilation, the code is executed in its native format, captured, recompiled to the host format, and executed on the host processor.

Pros: Similar to the advantages of using emulation, but generally with faster results.

Cons: No products that use dynamic recompilation were tested, as the test environment was comprised only of x86-based machines, which had no need for such products. Performance and ability to evade detection are not known.

3.2.3.6 Dedicated Virtualization Server

A server, functionally, or physically dedicated to the task of running Guest OSs. VMWare ESX is an example of the latter, as the host software is custom-designed only to run Guest OSs as quickly and efficiently as possible. ESX functions as both the VMM (Virtual Machine Monitor) and the host operating system. However, a plain server running Windows 2000 Server that only has base necessities and VMWare Server installed could also be considered a dedicated virtualization server.

3.2.4 Exploring Available VMEs

3.2.4.1 Overview

Seven VMEs were reviewed in an effort to find the product best suited for use in malware analysis labs. All VMEs were installed on a Compaq Proliant DL380 (first generation) running Windows 2000 as the host OS. The hardware is modest by today's standards, having twin 750Mhz Pentium III processors, One gigabyte of RAM, and 75 gigabytes of RAID 5 storage.

Table 6: Notable Emulators and VMEs (more complete list on Wikipedia [16]):

Product	Type	Pros	Cons
VMWare Server 1.0	Native	**Can be remotely controlled and configured; Easy setup; Free**	Easily detected by malware
Virtual PC 2004	Native	Fast, easy setup	Commercial; Costs money; Easily detected by malware
Parallels Workstation 2.2	Para	Easy setup and configuration	Commercial; Costs money; Easily detected by malware
Bochs 2.3	Emu	**Free; Open Source; Cannot be easily detected by malware**	Operating systems run much more slowly in emulation; Need a fast processor (2Ghz+) for emulated Windows OSs to be usable
QEMU 0.8.2	Emu	Free; Fast; Open Source; Can evade detection	Can be confusing to configure and get running

3.2.4.2 VMWare Server 1.0

VMWare Workstation enjoyed a long reign as the premier VME for malware analysis. It allowed the user to take snapshots, begin testing, and revert to the snapshot when testing was completed. When VMWare Server was released for free, it quickly became one of the most valuable free tools in many computing areas – especially information security.

The ability to detect VMWare from within a guest, however, has hurt its effectiveness in malware analysis. Even without installing VMWare Tools on a Windows guest,

detecting the virtualized environment is trivial with tools made available by Joanna Rutkowska [2] and Tobias Klein [3].

VMWare could still be a fantastic tool for malware analysis, provided the malware being analyzed doesn't use VM detection. However, analysis is necessary because the malware's behavior is not known. Even if analysis on VMWare is partially effective, who is to say that the unknown sample being analyzed isn't using VM detection to hide at least part of its behavior?

Pros: Fast, convenient, enterprise features, ability to create an isolated guest network

Cons: Pointless to use as long as it can be easily detected without feasible means to evade detection

Bottom Line: Can still be used to run support servers, but should no longer be used for malware analysis.

VMWare Server can be downloaded here (registration is necessary to receive license key): http://www.vmware.com/products/server/

3.2.4.3 Virtual PC 2004

Virtual PC, like VMWare, is fast, flexible, and easy to use. However, as it uses Native Virtualization, it is also easy for malware to detect. Also like VMWare, its popularity also makes it a likely target for malware VM detection.

Pros: Fast, easy to use, now offered for free

Cons: Same as VMWare

Bottom Line: Same as VMWare

Microsoft's Virtual PC 2004 can be downloaded from here: http://www.microsoft.com/Windows/virtualpc/default.mspx

3.2.4.4 Parallels Workstation 2.2

Although Parallels approaches virtualization differently, the result is the same. Malware will be able to employ the same detection techniques that work on VMWare and VirtualPC.

Pros: Fast, easy to use

Cons: Same as VMWare; Is the only product tested that is not free

Bottom Line: Same as VMWare

Parallels is available for Windows and MacOS hosts here: http://www.parallels.com/

3.2.4.5 Bochs 2.3

With Bochs, the first emulator tested, things began to get interesting. First and foremost, this emulator's biggest drawback is lack of speed. Installing a Windows 2000 guest took several days of clicking and waiting. Once it was finally installed and booted, it was fast enough to run some VM detection tools, but the slowness of the mouse cursor was reminiscent of using VNC to remotely control a desktop over a 28.8k dial-up modem link. Bochs is likely usable on faster processors (2Ghz+), but on an older 750Mhz Pentium, it could not be used for malware analysis on a day-to-day basis.

Speed aside, the major breakthrough with Bochs is that, due to the fact that the processor is fully emulated, all

current VM detection techniques but one fail. Red Pill (see Figure 6), Jerry, and Scoopy all reported that the Windows 2000 Bochs guest was running directly on hardware.

The one exception was Tobias Klein's "Doo" tool, which is a small VBScript that looks for a few key hardware descriptions within the Windows registry. On Bochs, Doo revealed that the hard disk was named "Generic 1234" (see Figure 7), which is obviously not a physical drive. This problem was quickly eliminated after discovering that all drive names are user configurable in the Bochs configuration script.

Pros: Evades most popular detection techniques, relatively simple (for a non-commercial product) to download and configure, solid, well-tested code

Cons: Very slow; Can still be detected, due to "bugs" in the code

Bottom Line: Feasible for malware analysis on faster hosts. Could potentially completely evade detection with some code changes.

The main Bochs website is here:
http://bochs.sourceforge.net

3.2.4.6 QEMU 0.8.2

The best was unintentionally saved for last. QEMU also uses emulation, is free and is open-sourced. After using a menu-based console tool to create a disk image, it was somewhat difficult to determine how to use the main qemu.exe executable to launch the VME (this task is perfectly straightforward in Linux, though). After some brief research, the following command launched QEMU set to boot

off of a Windows 2000 cd. The difficulty was figuring out how to reference the physical cdrom drive using the Windows drive letter.

qemu –L ./bios –boot d –cdrom //./d: –hda win2000.img

Once the Windows 2000 guest was installed (in much less time than in Bochs) and working in QEMU, a speed difference was immediately apparent. QEMU ran almost as quickly as VMWare or Virtual PC on the same 750Mhz Pentium. QEMU also evaded all but one of the VM Detection techniques, just as Bochs did.

As with Bochs, however, Doo revealed the presence of QEMU HARDDISK and QEMU CD-ROM in the registry. These values are hard-coded in the QEMU binary, but since QEMU is open source, changing two strings in the code and recompiling should be a trivial task. A simple find/replace on “QEMU HARDDISK” and “QEMU CD-ROM” to more realistic sounding disk labels should be sufficient.

Note that a “QEMU Accelerator” is available, but it should not be used in a malware analysis environment, as it will use Native Virtualization in place of Emulation within QEMU, which will make VM detection again possible.

The Windows version of QEMU is reportedly in alpha stages of development, but testing revealed no bugs or issues that would prevent its daily use in a malware analysis lab.

Pros: Fast, easy to use, relatively simple to set up, and, most importantly, evades detection techniques known to be in use.

Cons: Windows version is still considered to be in alpha stages; Source code must be modified to completely evade detection.

Bottom Line: QEMU is, without a doubt, now the ideal VME for malware analysis, and with some code changes, has the potential to completely evade detection.

The Windows version of QEMU is available here: http://www.h7.dion.ne.jp/~qemu-win/

And the main website is here: http://fabrice.bellard.free.fr/qemu/

3.3 Malware Behavior

3.3.1 Dynamic Nature

Although more sophisticated attacks are currently on the decline [17], they have by no means disappeared, and less sophisticated attacks are increasing in variety. Increasing variety of malware will likely require a larger variety of test machines in an analysis lab. Scenario two and three in section three should scale nicely to accommodate expanded labs.

3.3.2 Emulating Malware Needs

Early malware was spread for fun, out of anger, to prove points, and for various other reasons, but, as mentioned in this paper's introduction, the landscape is changing. The time for commissioned malware, malware seeking sensitive personal data, and malware-enabled corporate espionage is upon us. In the old days, simply providing a juicy, unpatched system was all that was necessary to observe malware execute all their carefully crafted behavior and functionality. Nowadays, many varieties, especially malware

customized to exploit a specific target, must be baited. This must be taken into account when creating and configuring test lab components.

- Is the malware looking for sensitive data locally or on file shares?
- What kind of files is it looking for? Powerpoint, Excel, Word, address books, password files/databases, or something else?
- What kind of data is it looking for, credit cards, social security numbers, or driver's license numbers? Finding the reason why it is collecting this data, and how it is being used can uncover other hidden aspects of its purpose and mission. Understanding its purpose and mission is a large step towards defeating it.
- Do any of these files or data need to be staged[3] in order for the malware to "show its hand"?
- How does the malware communicate to its owner, if at all?
- What if malware uses authentication and encryption to hide and protect the data it is sending or receiving?

Table 7: Files and/or Conditions Malware is Interested In

Interest	**Reasoning**
Outlook/Outlook Express	Address Books for Spamming or propagation
Firefox	Password manager vulnerabilities
LAN	Most malware assumes LAN connectivity, but older examples refused to run otherwise.

[3] False sensitive data is sometimes available for testing use. Although it is not real or functional, it will pass basic tests as the real deal, and *may* fool automated attempts by malware to steal the real thing.

WAN	Same as LAN, except refuses to run without Internet connectivity
Sensitive Data	Sell for profit

Table 8: Data Malware May be Interested In

Data	**Black-Market Value**
Credit Card Information	Worth $20 to $75 USD per #, depending on source
Social Security Numbers[4]	Identity Theft, individually worth substantially more money than credit card information.
TaxIDs	Like with identity theft, a TaxID, when obtained along with other important pieces of information can be used for many kinds of fraud
Banking Information	The amount available in the accounts, should they be drained.
Local Passwords	Passwords could be used to protect any of a number of things.
Website Passwords	Many people use websites to interact with financial services.
Corporate/Proprietary Data	Corporate espionage can be quite lucrative, especially when the right data is offered to the right company at the right time.
Confidential Data	Some malware could be used to gather confidential data, which could then be ransomed to the data's owner,

[4] SSNs are US-specific, but any international personal identification numbers that contribute to identity theft can be substituted.

	should it be an individual or a company.
LAN Configuration/Maps	May aid in more serious, directed attacks; Some custom targeted malware may be used to gather information for a focused attack or theft.

3.4 Other Considerations

3.4.1 Where is malware stored prior to release into the lab?

Generally, the malware should be where could do the least damage, should it be mishandled, or an accident occur. A linux system is generally a good place for malware, as most is written to run only on Windows operating systems. Even then, malware should be compiled into an archive or compressed format, along with a checksum of the malware itself. Tar and Zip are available on most UNIX-like systems, and Zip is available natively on Windows XP, 2003 and Vista.

For extra security, consider password protecting and even encrypting the archive to protect it from misuse. TrueCrypt [18] and PGP/GPG [19] are good tools for protecting files in this manner.

3.4.2 How is malware released into the lab?

Automatic Entrance Points - The primary entrance point for automatic release of malware into the lab would be via honeypot. There are other potential automated entry points, however:

- Via a dedicated email that malware can be forwarded to for analysis, such as with the Internet Storm Center handlers [20].
- Upload form on a website dedicated to analyzing malware, such as with the Offensive Computing website [21].
- Custom NAP or NAC-based controls that, in the process of quarantining a system on a corporate or private network, forward a copy of any discovered malware to a predetermined drop-off point in the analysis lab.
- Custom scripting, either standalone, or working with antivirus software to send samples to the lab, also in a corporate or private network.
- Commercial or open source software (most likely antivirus or anti-spyware related) configured to send samples of newly discovered malware to their labs.
    - This could be particularly useful if antivirus/malware software techniques continue to focus more on heuristic scanning as opposed to signature-based scanning.
- Once in, malware samples should be restricted as much as possible to prevent leakage back out into the wild.

Manual Entrance Points – Manually introducing malware to a test environment is likely to be a simple file copy from a quarantine server, or some sort of portable media that can be stored off the network. The “defense-in-depth” principle is highly recommended in this situation to prevent accidental execution of malicious software in environments that may not be segregated or quarantined.

Store on:

- Read-only media (CD or DVD)
- Portable disks (USB drives)
- File Server

Protect with:

- Access permissions
- Password protected archive
- Encrypted file store

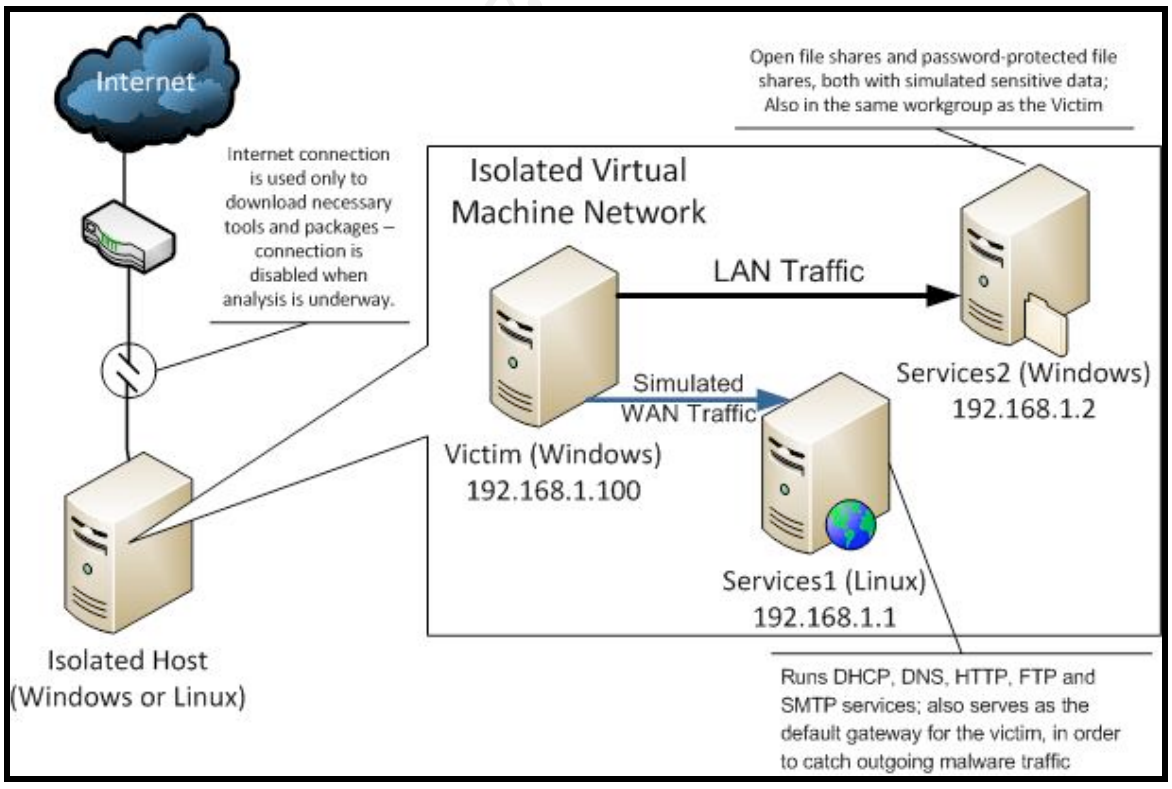
4. Lab Architecture (High Level)

4.1 Suggested Setups

4.1.1 Scenario One: Single PC Lab

The single PC lab is one of the most commonly used environments, especially for researchers. It is easy to deploy on a single workstation, and travels easily if deployed on a laptop. If using emulators, such as Bochs or QEMU, rather than VMWare, keep in mind that it will be more difficult to isolate guest networks. The easiest way to do this is using the VLAN features of QEMU [22], and then blocking all incoming traffic from that VLAN using a host-based firewall on the host itself.

Figure 8: Single PC Lab

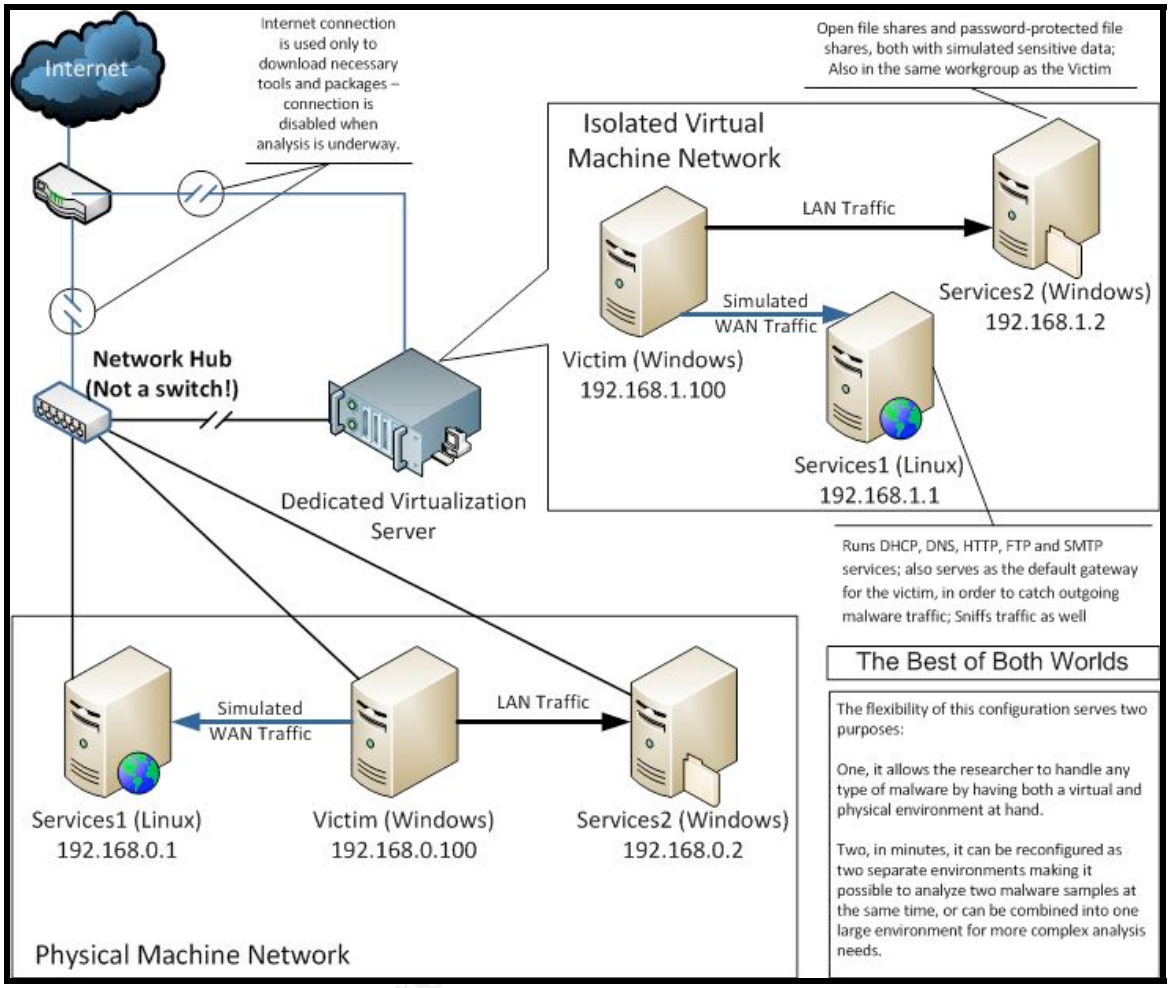


4.1.2 Scenario Two: The Expandable Multi-PC Test Lab

At first glance, the Multi-PC test lab appears to just add a physical environment to the existing virtual one in Scenario one, but it opens up several new possibilities.

- The environment can be combined to make additional resources available for malware testing.
- Split apart, each part of the environment (physical and virtual) can each be dedicated to its own separate analysis.
- One can serve as an immediate backup for the other.
- This lab can be capable of processing all known malware, including those that detect and refuse to run in VMEs.
- This model of several small segregated environments can be replicated easily and inexpensively, to create a larger analysis lab that could scale to support a team of researchers working full time on analysis.

Figure 9: Multi-PC Lab



4.1.3 Scenario Three: Dabbling in Malware Analysis Automation

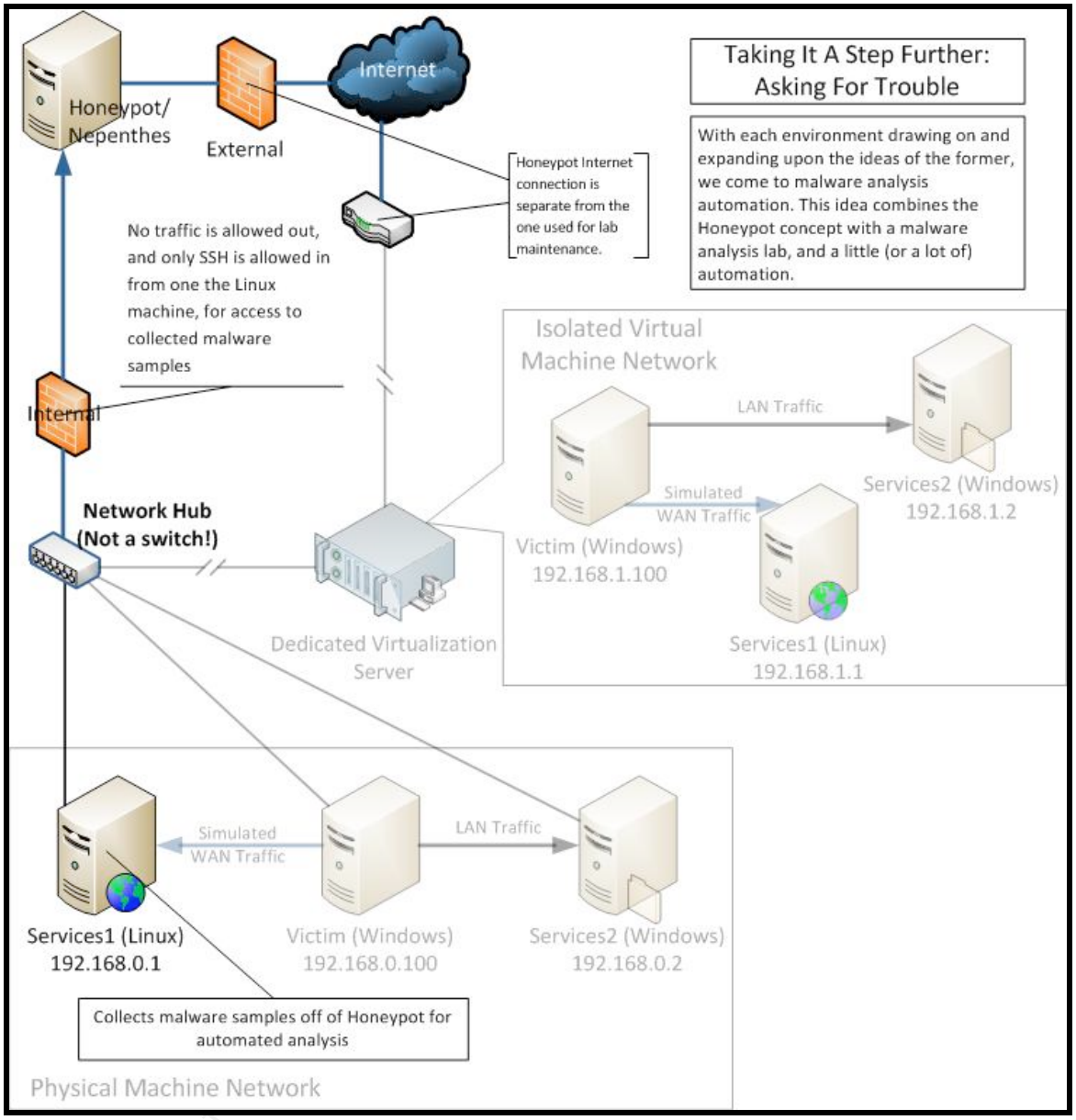
This environment is modeled after scenario two, except that a honeypot and some automation has been added. As displayed here, malware analysis automation could be integrated into any carefully designed malware analysis lab model.

Highlights include:

- Type of Honeypot is not important, as long as it collects malware samples

- Honeypot uses a different link to the outside world (read: Internet) than what the lab uses for maintenance, research and software updates
- Honeypot is contained in a DMZ, positioned between the Internet and the malware analysis lab, and contained by two firewalls.
- External firewall has a normal ruleset, as would befit a Honeypot.
- Internal firewall has a strict “deny all” ruleset, only allowing SSH (TCP 22) into the Honeypot from the lab.
- Linux Server within the Lab (Services1) can be scheduled to check the Honeypot for newly collected malware on a regular basis. Each malware sample is transferred from the honeypot to the linux server and is then analyzed and stored.

Figure 10: A Partially Automated Lab



4.2 Procedural Examples

Note that more work needs to be done on the software side in these examples, such as baselining the environments, and setting up the necessary analysis tools. This is out of scope for this paper, and although may be mentioned, will not be discussed in detail here.

4.2.1 The Malware Researcher as a Road Warrior

1. Researcher is on the road, and learns of a need for quick analysis on some malware.
2. The sample is obtained in a secure manner (malware contained within a ZIP, RAR or TAR file where it cannot execute, and then emailed to the researcher, or posted on a website).
3. The researcher creates two new VMEs on his/her laptop. The researcher has several pre-built systems for various purposes, zipped up and ready for deployment. A Windows XP-based “Victim” and a Linux-based support system are unzipped into place.
4. Both systems are powered on, the malware samples are copied into the VMEs using a USB thumbdrive or harddrive, and the researcher completes the necessary short term analysis.
5. After analysis is complete, the researcher either saves the VMEs for further analysis in a more complete lab, or if they have served their purpose, they are permanently deleted.

4.2.2 Automated Submission and Analysis for Antivirus Vendors

Such a process could yield quick and efficient turnaround on protection and outbreak prevention for many antivirus users.

1. Commercial or open-source antivirus software asks the user whether or not they would like to participate in an automatic virus/malware submission program. The user will be warned that any host-based or network firewalls may need to be configured to allow antivirus products to upload files over the Internet.

2. The antivirus software detects, heuristically, a piece of new software with potentially malicious properties and behavior.
3. Checksums of the related files are created, and checked against vendor and/or community malware databases.
4. If no matches are found, the software is considered POTENTIAL NEW, and is uploaded securely, either to a staging point, or directly to the analysis lab.
5. All subsequent submissions of software with the same checksums are not uploaded. Rather, a count is kept of how many subsequent copies of the same potential malware are discovered.
6. If the count increases significantly over a short period of time, the software is then considered LIKELY NEW, and is analyzed by automated tools.
7. Upon completion of automated analysis, a report is emailed to appropriate researchers for review and approval of corrective actions (most likely, a new virus definition update with removal instructions).

Also Note:

- Currently, ClamAV requests new malware be submitted via a web-based form.
- Symantec has used a “digital immune” system [23] in enterprise products in the past. Developed by IBM with Symantec’s assistance, these systems only performed analysis and provided solutions for the environment they were deployed in, rather than for the entire community of users using the product. One would imagine, however, that a customized version of this system in Symantec’s lab assists in the creation of

solutions for users of their personal antivirus product as well.

- The design of an “Automated Virus Analysis Center” is described in a now ancient (by computer industry standards) paper from an IBM Research Team, *Anatomy of a Commercial-Grade Immune System*[24].

5. References and Appendices

5.1 Appendix A: VM Detection Results

	Redpill	Jerry	Scoopy	Doo	Speed
Win2000	0x8782f7e8 Not in Matrix.	You are on a native system.	IDTBase: 0x80036400 Win2000	No virtual devices	Native
			LDTBase: 0xdead0000 Win2000/XP		
			GDTBase: 0x80036000 Win2000		
VMWare Server 1.0	0xffc18000 Inside Matrix!	You are inside of VMWare. Product Version: GSX	IDTBase: 0xffc18000 Unknown VMWare	NECVMWar VMWare IDE CDR10	Near Native
			LDTBase: 0xdead4060 Unknown VMWare	VMWare, VMWare Virtual S1.0	
			GDTBase: 0xffc07000 VMWare Version 4: 100%	VMWare SVGA II	
				VMWare, Inc.	
Parallels	0xeb110000 Inside Matrix!	You are on a native system	IDT Base: 0xeb110000 Unknown VMWare	Virtual HDD [0] (Target ID0)	Near Native
			LDT Base: 0xdeadff5b Unknown VMWare	Script Error	
			GDT Base: 0xeb153000 Unknown VMWare	Script Error	
				Script Error	
				Found also: HKLM\HARDWARE\DESCRIPTION\System VideoBiosVersion = 2Parallels(2) VGA-Compatible BIOS Version 1.05...	
				And: PRL Virtual CD-ROM (Target ID1)	
Bochs	0x80036400 Not in	You are on a	IDTBase: 0x80036400	Generic 1234	Slow

	Matrix.	native system.	Win2000		
			LDTBase: 0xdead0000 Win2000/XP	Script Error	
			GDTBase: 0x80036000 Win2000	Script Error	
				Script Error	
QEMU	0x80036400 Not in Matrix.	You are on a native system.	IDTBase: 0x80036400 Win2000	QEMU HARDDISK	Near Native
			LDTBase: 0xdead0000 Win2000/XP	QEMU QEMU CD-ROM	
			GDTBase: 0x80036000 Win2000	Cirrus Logic 5446 Compatible Graphics Adapter	
				Microsoft	

5.2 General References

- Skoudis, E., & Liston, T. (2006) *Thwarting Virtual Machine Detection*, from http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf

- Skoudis, E., & Liston, T. (2005, December). *Counter Hack Reloaded*. Prentice Hall.

5.3 Referenced Notes

[1] Skoudis, E., & Liston, T. (2003). *Malware: Fighting Malicious Code*. Prentice Hall.

[2] Rutkowska, J. (2004, November). *Red Pill... or how to detect VMM using (almost) one CPU instruction*, from http://www.invisiblethings.org/papers/redpill.html

[3] Klein, T. (2003). Scoopy Doo - VMware Fingerprint Suite. from http://www.trapkit.de/research/vmm/scoopydoo/index.html

[4] Klein, T. (2003). Jerry – A(nother) Vmware Fingerprinter http://www.trapkit.de/research/vmm/jerry/index.html

[5] Zeltser, L. (2006, November 11). Virtual Machine Detection in Malware via Commercial Tools. Retrieved January 18, 2007, from SANS Internet Storm Center Web site: http://isc.sans.org/diary.html?storyid=1871&rss

[6] Rutkowska, J. (2006, November). *Stealth Malware Taxonomy*, from http://invisiblethings.org/papers/malware-taxonomy.pdf

[7] Lagerweij, B. (2006, February). *Bart's Preinstalled Environment (BartPE) bootable live windows CD/DVD*, from http://www.nu2.nu/pebuilder/

[8] Lagerweij, B. (2003, January). *Run Winxp from Cd, Run bare Windows XP from CD*, from http://www.911cd.net/forums//index.php?showtopic=3&hl=

[9] Wanner, R. (2004, July). Reverse Engineering msrll.exe, from Web site: http://www.giac.com/certified_professionals/practicals/grem/32.php

[10] Caton, M. (2006, December). eWEEK Labs' Stupid Technology Tricks of 2006, from Slide 3 Web site: http://www.eweek.com/slideshow_viewer/0,1205,l=&s=700&a=196323&po=3,00.asp

[11] Web site: http://en.wikipedia.org/wiki/Application_Binary_Interface

[12] Barham, P., Dragovic, B., Fraser K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., & Warfield, A. (2003). *Xen and the Art of Virtualization*, from http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf.

[13] Web site: http://bochs.sourceforge.net/

[14] Web site: http://fabrice.bellard.free.fr/qemu/

[15] Ferris, P. (December, 2006). *Attacks on Virtual Machine Emulators*, from http://pferrie.tripod.com/papers/attacks.pdf

[16] Web site: http://en.wikipedia.org/wiki/Comparison_of_virtual_machines

[17] Kirk, J. Malware quality falls but hassle rises. Retrieved January 18, 2007, from Techworld.com Security News Web site: http://www.techworld.com/security/news/index.cfm?newsID=7636&pagtype=samechan

[18] Web site: http://www.truecrypt.org/

[19] Web site: http://www.pgpi.org/

[20] Web site: http://isc.sans.org/contact.php

[21] Web site: http://www.offensivecomputing.net

[22] Web site: http://fabrice.bellard.free.fr/qemu/qemu-doc.html#SEC24

[23] Web site: http://www.symantec.com/press/1999/n990511.html

[24] White, S.R., Swimmer, M., Pring, E.J., Arnold, W.C., Chess, D.M, & Morar, J.F. (1999). *Anatomy of a Commercial-Grade Immune System*, from http://www.research.ibm.com/antivirus/SciPapers/White/Anatomy/Anatomy.PDF.