that would require the use of an external program, and then the virus would no longer be ‘pure batch’), so a file will not be infected if it has the read-only attribute set.

LYME DISEASE
The virus has a fatal bug when run under Windows XP: the line ‘set _out1=strftime(‘%-%‘, which is supposed to append ‘%‘ (the double ‘%‘ is required in order to emit a single ‘%‘), does not append anything. It is not known why this happens, but it appears that a line cannot end with that sequence of special characters. The bug appears to be in Windows, not in the virus. If an additional character is added to the line, then all of the characters are appended correctly. If the virus had added that additional character, and then removed it after the characters were appended, then the virus would work on Windows XP, too. The bug causes the virus to fail to parse anything, and then to delete itself, because there is no new representation.

The virus has an ‘even more’ fatal bug when run under Windows 2000 (the bug that exists in the Windows XP command processor is present here, too). The line ‘set \n_val1 += “_ind”‘, which is supposed to select the case of the randomly selected letter, does not make use of the ‘_ind’ variable. Instead, the value is always treated as a zero. This might be considered to be a bug in Windows, rather than in the virus, however the behaviour is undefined because the documentation regarding the use of quotes is ambiguous regarding this situation. The virus contains another line in the same style, but without the quotes, so we can assume that this is a bug in the virus. If the quotes were removed, and if the fix were applied as for the Windows XP case, then the virus would work on Windows 2000, too. The bug causes the virus to emit strings that are composed solely of the letter ‘A’.

The virus works correctly on Windows 7 without modification. This is especially interesting, because the virus writer is known for producing very compatible code. For example, most of his binary viruses still support Windows 95. His more recent viruses ‘merely’ require Windows NT. It is clear that he did not test this virus on anything other than a relatively recent platform such as Windows Vista (assuming that it works there – I did not try it) or Windows 7. Perhaps he finally upgraded his machine.

CONCLUSION
It’s clear that some people have too much time on their hands, to have found a way around all of the limitations and quirks of the batch language, and produced a virus like this. However, if we can’t stop them from writing viruses at all, then we can at least be thankful that they’re not writing something much worse than this.

TECHNICAL FEATURE 1
MALWARE DESIGN STRATEGIES FOR CIRCUMVENTING DETECTION AND PREVENTION CONTROLS – PART ONE
Aditya K. Sood and Richard J. Enbody
Michigan State University, USA

In this paper, we discuss some of the different techniques that are used by present-day malware to circumvent protection mechanisms.

1. DETECTING WINDOWS X86 EMULATOR
With the advent of Windows x64 systems, the x86 emulator has been added to provide backward compatibility. WOW64 is an x86 emulator that allows 32-bit Windows applications to run on 64-bit Windows. Malware writers use an x86 emulator detection routine to get detailed information about the environment in which the malware is going to be executed. This is a critical step from the attacker’s perspective because in order to trigger successful DLL injection, a 32-bit process has to load a 32-bit DLL, thereby avoiding collisions with 64-bit DLLs. Malware writers harness the power of inbuilt

![Image](image-url)

Figure 1: x86 emulator detection using ‘IsWow64Process’ in ICE bot.
2. ANTI-VIRTUAL-MACHINE CODE

This technique has been used widely by malware writers to detect the presence of virtual machines. The primary aim is to make analysis of the malware harder by shutting down some of its functionality if a virtual machine is detected. There are several techniques that can be used to detect the presence of a Virtual Machine Environment (VME), as follows:

- Memory-specific techniques include Red Pill, which is a proof of concept that utilizes the Store Interrupt Descriptor Table (SIDT) to collect information about the Interrupt Descriptor Table Register (IDTR). The IDTR points directly to the Interrupt Descriptor Table (IDT) and, based on the memory address, Red Pill can detect the presence of a virtual machine. ScoopyNG [1] is another proof of concept that scrutinizes the location of the Local Descriptor Table (LDT), Global Descriptor Table (GDT), Interrupt Descriptor Table (IDT) and Store Task Register (STR) to determine the presence of a virtual machine. It also runs additional checks using VMware commands such as ‘get version’, ‘get memory size’ and ‘emulation check’. Any of these techniques can easily be deployed by malware to detect whether the code is inside a virtual machine. Listing 1 shows the output of ScoopyNG.

VMDetect [2] uses an invalid opcode mechanism that acts as a backdoor code to detect a virtual machine. It uses the privileged ‘IN’ (reading from communication ports) instruction to check if an exception occurs as ‘EXCEPTION_PRIV_INSTRUCTION’, and uses this information to verify whether the code is executing under VMware. However, these protections can easily be subverted by disabling all the protection flags in the VM configuration files, as shown in Figure 2.

Several samples of malware have been found using one of these memory-based techniques to design an anti-virtual-machine routine to subvert detection. (More details about virtual machine detection and analysis can be found at [3].)

- Virtual machines make a number of adjustments in the Windows registry and create certain specific processes that can be utilized to detect the presence of a virtual machine environment. We have come across several registry-based settings that can be used to harness information about virtual machines. One of these is very critical as it is very hard for analysts to work around, as tampering with this key information could

Listing 1: ScoopyNG in action.

```
C:\ScoopyNG\ScoopyNG.exe

#**********************************************************************
# ScoopyNG : The VMware Detection Tool
# Windows version v1.0
#
[# Test 1: IDT
IDT base: 0x8003f400
Result : Native OS

[# Test 2: LDT
LDT base: 0xdead0000
Result : Native OS

[# Test 3: GDT
GDT base: 0x8003f000
Result : Native OS

[# Test 4: STR
STR base: 0x28000000
Result : Native OS

[# Test 5: VMware "get version" command
Result : VMware detected
Version : Workstation

[# Test 6: VMware "get memory size" command
Result : VMware detected

[# Test 7: VMware emulation mode
Result : Native OS or VMware without emulation mode
(enabled acceleration)

#: tk, 2008
#: [ www.trimkit.de ]
#**********************************************************************
```

Figure 2: Memory bypassing configuration parameters.
interfere with the booting state of the virtual machine. Figure 3 shows the VMware detection check based on SCSI/Disk info.

- **VMware** can easily be detected based on the Media Access Control (MAC) address. This is not a widely used technique because it is not difficult to tweak the MAC address of a system. **VMware** can be detected in this way because the first 24 bits of the MAC address
define the manufacturer of the machine. Generally, MAC addresses for VMware machines always start with '00-05-69-xx-xx-xx' or '00-0c-29-xx-xx-xx'. If the MAC address matches any of the 24 bits discussed above then it is a VMware machine. Figure 4 shows VMware detection using this method.

- The Virtual Machine Communication Interface (VMCI) [4] is another target that can provide details about the running state of a virtual machine. VMCI provides an effective communication interface between the virtual machine and the host operating system. To detect whether a guest is running inside a virtual machine, malware writers can trace the installed VMCI device on the system. Simply, the malware can open a handle to the VMCI device(s) present on the system to verify the presence of a virtual machine. Table 1 presents the information that is required to query the VMCI interfaces on Linux and Windows operating systems.

![Figure 3: SCSI/Disk-based VM detection.](image)

![Figure 4: VMware detection based on MAC address.](image)

<table>
<thead>
<tr>
<th>Operating system</th>
<th>VMware VMCI details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>· Host machine: /dev/vmmmon</td>
</tr>
<tr>
<td></td>
<td>· guest machine: /dev/vmci</td>
</tr>
<tr>
<td>Windows</td>
<td>· Host machine: \vm\x66</td>
</tr>
<tr>
<td></td>
<td>· guest machine: \VMCL</td>
</tr>
</tbody>
</table>

**Table 1: VMCI details of VMware.**

Malware writers typically look for ‘\\\BoxGuest’ to determine if a virtual box is present on the system.

### 3. INJECTIONS USING APC

DLL injection has been around for several years and is used very effectively by malware writers. This technique is used to inject an authorized DLL into the target process at runtime to hook specific functions so that execution flow can be redirected. Until now, malware writers have explicitly used three standard techniques for performing DLL injection: CreateRemoteThread, SetWindowsHook and Appinit_dlls. However, recently APC-based DLL injection has been seen in the wild. Both user- and kernel-mode Asynchronous Procedure Calls (APCs) [5, 6] are used to build robust malware. All the APC-based routines require the _KAPC structure, which is called using the ‘nt!KeInitializeApc’ call. The details are shown in Listing 2.

The kernel-mode and user-mode functions executed through the APC procedure are termed kernel-mode and user-mode routines, respectively. APC-based DLL injection can be used by both user-land and kernel-land rootkits, as discussed in the following sections.
3.2 Kernel-mode APC injection

Kernel-mode APC injection is categorized into two types: regular kernel-mode APC and special kernel-mode APC. In regular kernel-mode APC, the target kernel-mode routine is executed at passive interrupt request level (IRQL), whereas special kernel-mode APC triggers the target kernel-mode routine at APC IRQL. Both special and regular kernel-mode APCs are asynchronous events that have the ability to direct the flow of execution in threads from normal state to the target kernel routine by taking them out of their waiting states. The only difference is that regular kernel-mode APC is executed in more restricted conditions.

The complete details of kernel-mode and user-mode APC can be found in [7]. ZeroAccess [8] (and see p.4) is an example of malware that has shown the usage of code execution through APC. Listing 3 shows a simple prototype of APC injection in action.

4. MUTEX-BASED DETECTION

Many malware writers use mutex-based detection techniques to determine whether an operating system has any security programs installed on it. A mutex [9] is typically a mutual exclusion lock and is used to protect the different resources and data from being accessed concurrently. Malware writers define the mutex routine in the main entry point of the malware. The primary aim is to detect whether any other installed program is using that mutex. Generally, malware writers have knowledge of the mutexes (unique mutex names) that are used by different protection programs or anti-virus software that may be installed on the system. In Windows-based malware, the CreateMutex() API is used extensively to detect the presence of any type of mutex in the system.

The entry routine defined in the malware code triggers this API to scrutinize whether the mutex is already present in the system. If the mutex exists, the API returns an error message – which shows that protection programs have already been installed on the running machine. Based on this information, the malware stops its execution and becomes dormant. Zeus, SpyEye, ICHX and several other bots use this technique.

Mutexes are also used in bot wars. Based on mutex information, one bot can kill another to increase its kingdom of infections. In this case, the OpenMutex() API is used to access the running mutex in the system. The kind of API used for collecting mutex information from the system depends on the malware writer’s choice. This functionality has been seen in earlier versions of SpyEye, which had an inbuilt Zeus-killing routine that used named pipes and designated commands to kill the Zeus bot in the system.
#define _WIN32_WINNT 0x0500
#include <windows.h>
#include <ntdef.h>

DWORD Trigger_APCInject (PCHAR szProcName, PCHAR szDllName){
    DWORD dwRet=0;

    Step 1: Define the NtMapViewOfSection by calling GetProcAddress and GetModuleHandle to load the NtMapViewOfSection by importing ntdll.dll.

    Step 2: Allocate buffer by calling CreateFileMapping and defining the view of the file by calling MapViewOfFile.

    Step 3: Define the PROCESS_INFORMATION and STARTUPINFO structure using ZeroMemory.

    Step 4: At this point, create the suspended process by using CreateProcess and then call NtMapViewOfSection, LoadLibrary, GetProcAddress and QueueUserAPC.

    Step 5: Trigger the UnmapViewOfFile to release the address space in the process that is occupied during mapped view of the file.
}

int main(void){
    DWORD dwHandle= Trigger_APCInject (Target_Process_for_Injection, DLL_To_Be_Injected);
    if(!dwHandle)
        puts("[-] APC Injection Successful!");
    else
        printf("[-] APC Injection Failed. Fix it!", dwHandle);
    return 0;
}

Listing 3: Prototype of APC injection.

5. EXPLICIT RUNTIME LINKING

To detect the presence of security programs in the Windows operating system, malware writers use the de facto standard of runtime dynamic linking of system DLLs.

This technique allows malware writers to design a generic routine that calls the LoadLibrary() API to dynamically load the target library into the address space of the calling process. The GetProcAddress() API is used afterwards to resolve the address of the loaded library in the system. The detection routine is very simple. Since the malware writers have information about the specific set of DLLs used in sandbox programs, anti-virus software and many others, if the required DLL is loaded through the LoadLibrary() API, it means the system is equipped with the protection software and the malware stops its execution and does not interact with the system. If the required DLL is not found in the system, then the malware starts the infection process.

Figure 5 shows the idea behind this detection technique.

Figure 5: Detection-based explicit runtime linking.

In the first part of this article, we have presented some of the tactics used by malware writers to design code that is resistant to the detection routines used by malware analysts. We will continue the discussion in part two of the article, in which we will look at advanced anti-debugging, polymorphism, tactical encryption routines, subverting client-side protection software, bypassing anti-virus solutions, etc.

REFERENCES


