Introduction

This article will demonstrate how an application can detect if it is being run from inside a virtual machine software.

The code in this article will detect two well known machine virtualization software:

- Microsoft's Virtual PC (formally from Connectix).
- VMWare from VMWare.com

Other virtual machine software such as Bochs or Plex86 are not covered in this article.

It is best that the readers have a general idea about the Intel x86 assembly language to better understand how the code works, however I will do my best to explain the techniques in layman's terms.

Please note that whenever I use the term "Virtual Machine Software", this means I am referring to a software such as Virtual PC or VMWare. When the term "Virtual Machine" is used, this means the emulated machine, usually running an operating system.

A little about virtual machine software

Virtual machine software are software that emulate a given machine's architecture using software (code) instead of relying on hardware, thus allowing a code to be executed in that virtual machine as if it is being run from a real machine.

Till today, these software are far from being perfect, and emulating a given real machine still poses many challenges due to complexities involved when trying to emulate every component of a given machine.

Both Virtual PC and VMWare allow you to install "add-in"s to accelerate emulation, allow drag-n-drop from your real desktop to your virtual desktop, and allow file sharing between your real machine and the virtual machine.

In order to accomplish this task, a communication mechanism between the virtual machine software and the virtual machine itself must exist.

This sort of interfacing is called a "backdoor interfacing", since, using a special/undocumented mechanism, certain commands can be carried and interpreted in a different manner (by the virtual machine software) unlike having them interpreted by the real machine.

Next, I'll be covering how you can tell whether your software is being executed using a real machine or a virtual machine software (covering both Virtual PC and VMWare).

How to detect Virtual PC

As you may already know, every machine has a defined set of instructions commonly referred to as Instruction Set Architecture (ISA).

When an invalid instruction (that is not present in the ISA) is encountered, the machine raises an exception of the type "Invalid Opcode". The software can either handle the exception (using the usual `try`/`catch`mechanism), let the operating system handle the exception, or crash the machine in worst cases.

Virtual PC uses a bunch of invalid instructions to allow the interfacing between the virtual machine and the Virtual PC software.

Here's what happens when Virtual PC's virtual machine wants to talk with Virtual PC:

1. The program sets exception handlers (`try`/`catch` blocks).
2. Set needed parameters before calling the VM software.
3. Issue a special "Invalid Opcode" instruction.
4. VM software will recognize this invalid opcode and act accordingly, causing no exception if VPC was present, and an exception if VPC isn't present.
5. The program's "`catch`" block will handle the exception and examine the returned parameters for the presence/absence of VM software.

In short, Virtual PC uses the "Invalid Opcode" mechanism as a backdoor.

The following code shows how to detect Virtual PC's presence:

Hide Shrink ▲ Copy Code

```
// IsInsideVPC's exception filter
DWORD __forceinline IsInsideVPC_exceptionFilter(LPEXCEPTION_POINTERS ep)
{
  PCONTEXT ctx = ep->ContextRecord;

  ctx->Ebx = -1; // Not running VPC
  ctx->Eip += 4; // skip past the "call VPC" opcodes
  return EXCEPTION_CONTINUE_EXECUTION;
  // we can safely resume execution since we skipped faulty instruction
}

// High level language friendly version of IsInsideVPC()
bool IsInsideVPC()
{
  bool rc = false;

  __try
  {
    _asm push ebx
    _asm mov  ebx, 0 // It will stay ZERO if VPC is running
    _asm mov  eax, 1 // VPC function number
    // call VPC
    _asm __emit 0Fh
    _asm __emit 3Fh
    _asm __emit 07h
    _asm __emit 0Bh

    _asm test ebx, ebx
    _asm setz [rc]
    _asm pop ebx
  }
  // The except block shouldn't get triggered if VPC is running!!
  __except(IsInsideVPC_exceptionFilter(GetExceptionInformation()))
  {
  }

  return rc;
}
```

More details on the code:

1. Install exception handlers.
2. Prepare input registers "`eax`" and "`ebx`".
3. Issue invalid instruction 0x0F 0x3F 0x07 0x0B. This invalid instruction is like a function designator, it tells Virtual PC what to do exactly. For other functionality, Virtual PC uses another invalid instruction.
4. Inside the exception handler -> modify registers so to mark VPC's absence (`EBX` is set to -1 if exception is triggered -> VPC is absent).
5. Return from exception and resume execution (only if VPC was absent).
6. Inspect returned registers accordingly.

How to detect VMWare

The Intel x86 provides two instructions to allow you to carry I/O operations, these instructions are the "`IN`" and "`OUT`" instructions. These two instructions are privileged instructions and cannot be used in a user-mode (while in protected mode) process unless the necessary privileges are enabled, so using them in normal cases will cause an exception of the type: "`EXCEPTION_PRIV_INSTRUCTION`".

VMWare uses the "`IN`" instruction to read from a special port. This port does not effectively exist, however when VMWare is present, that port will be the interface between the virtual machine and VMWare.

Here's the code:

Hide Shrink ▲ Copy Code

```
bool IsInsideVMWare()
{
  bool rc = true;

  __try

  {
    __asm
    {
      push   edx
      push   ecx
      push   ebx

      mov    eax, 'VMXh'
      mov    ebx, 0 // any value but not the MAGIC VALUE
      mov    ecx, 10 // get VMWare version
      mov    edx, 'VX' // port number

      in     eax, dx // read port
                     // on return EAX returns the VERSION
      cmp    ebx, 'VMXh' // is it a reply from VMWare?
      setz   [rc] // set return value

      pop    ebx
      pop    ecx
      pop    edx
    }
  }
  __except(EXCEPTION_EXECUTE_HANDLER)
  {
    rc = false;
  }

  return rc;
}
```

1. The program sets exception handlers (in case VMWare isn't present, we just discard its presence).
2. Set up into the EAX register the magic number 0x564D5868 (or 'VMXh').
3. Set EBX register to any value but the magic number.
4. Set ECX register to the "function number" value. The value 10 means Get VMWare version, other codes means other functionality.
5. Set into DX the magic port number 0x5658 (or 'VX'), this special port number allows interfacing with VMWare when it is present.
6. Read from that port into EAX.
    1. When VMWare is not present, an exception will occur and we discard VMWare's presence.
    2. Otherwise, the code flow continues.
7. EBX should now read the magic number value.
8. If so, then VMWare is present.

Using the code

This article comes with a sample GUI written in C#/VB.NET and VC++, allowing you to detect the presence of either VMWare or Virtual PC.

The C++ code uses "*DetectVM.cpp/.h*" which exports the following functions:

Hide Copy Code

```
bool IsInsideVPC();
bool IsInsideVMWare();
```

The C#/VB.NET code uses the following read-only properties:

Hide Copy Code

```
System::Boolean IsInsideVPC
System::Boolean IsInsideVMWare
```

Closing words

Hope you enjoyed and learned from reading this article and using the code. Thanks to CodeProject members for their continuous support and quality articles/code.

Special thanks goes to Mr. Ken Kato, for his work that enabled me to learn about VMWare's backdoor interface.