Analyzing Malware & Malicious Content

By: magikh0e - magikh0e@ihtb.org
Http://magikh0e.ihtb.org/

Some info may be incomplete, this paper is draft only.
Last Edit: 12-24-2009

reverse engineer

Table of Contents

Special Thanks too everyone I bugged in the process of writing this!
“Gr@ve_r0se, Neize aka Orly, Skiffy, Mannibal, Redsand... darpa hax0rs”

Overview

Malware, short for malicious software, is a piece of software that's sole purpose and design is to infiltrate or cause damage to a computer system without the owner's well informed consent. In the information security world we hear this term or expression all the time used by professionals to describe a variety of hostile, intrusive or other wise annoying code running on a system.

It is not uncommon to hear people still using the term "computer virus" as a catch-all sort of phrase to include all types of malware, without neglecting a real computer virus that would fall within. Having said that, Malware can be used to classify computer viruses, worms, Trojan horses, majority of root kits, spyware, suspicious adware, along with any other un-wanted software

Just because a piece of software may be buggy or defective will not be enough to be classified as malware. [1] Some findings by Symantec published in 2008 suggest that "the release rate of malicious code and other unwanted programs may be exceeding that of legitimate software applications." If you think this is something to worry about, according to F-Secure, [2] "As much malware [was] produced in 2007 alone, than has been created in the previous 20 years all together"

It is a safe bet to say, if you have ever browsed the internet, sent & received some emails or browsed to your favorite website. You have ultimately encountered a piece of malware once or twice.

Summary

Analyzing malware & malicious content aim's to be a paper covering the topics of analyzing malicious content and malware infested executables. This paper will take the reader through the very beginning steps of safely acquiring some malware specimen, and analyzing & disassembling the malware in a safe, controlled environment. Some basic knowledge of ASM, C will be assumed, as well as familiarity with tools such as OllyDBG, IDA Pro, Wire Shark and so on…

Why Analyze Malware anyways….?

In short, there is a multitude of different reasons we would analyze malware, see below to name just a few:

- Discovering signs of an intrusion or attempt there of.
- Assess the damages after an intrusion has been detected.
- Identifying vulnerabilities or ways to detect and identify new malware for developing techniques at preventing future infections.
- Answering questions…?

References

[1] http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_exec_summary_internet_security_threat_report_xiii_04-2008.en-us.pdf

[2] http://www.f-secure.com/f-secure/pressroom/news/fs_news_20071204_1_eng.html

Creating Safe Malware Analysis Environments

If you can recall in the story "The Three Little Pigs", they all had their own ideas of building a safe & secure shelter. When the first two failed, luckily the 3rd one had built an armored fortress house made of brick. When building our malware analysis environment, we want to make sure their will be no mistakes and we have a proper armored fortress the first time.

At the very least when building a malware analysis environment, we want to be 100% for sure that these system(s) do not have any access to any live production systems or the Internet. It is a good idea to always start out with a clean slate using a fresh install of the OS of your choice for analysis. I prefer to keep around several pristine installs of various operating systems and versions for analysis on a live system. When you start from a pristine state it becomes much easier to track and monitor changes done to the system after running a piece of malware for analyzing.

Physical VS Virtual Environments

You will have several options when creating a malware analysis environment. If you have the hardware lying around you can always build your lab using bare metal machines. I prefer to use virtualized Operating systems for several reasons that will be touched on below:

- Faster Restore times to a pristine state using non-persistent images

Non-persistent images support: VMware, Xen, Parallels

- No need for extra hardware lying around
- Switching between Operating Systems much faster
- The list could go on… which is going out of scope for this paper ;)

With anything good, there is always some down falls or faults that lie hidden within the details. I have always said if you wish to see the most creativity in computer security/programming, check and see what the malware authors are up too. At the end of the day their creations sole purpose is to run undetected and do its dirty deeds in the background. These guy's are savvy to debugging and disassembling as well, I have seen more than one piece of malware completely disable it's self when it is detected to be running within a virtual environment. This leads too a manual intervention using our good friends OllyDBG, IDA Pro along with others we will dive into shortly.

Building the Environment

If you are using real machines or virtual machines, the setup logically is pretty much the same. We will need at least one machine to serve the purpose of the victim. On this machine we will run the operating system of choice on which we wish to infect and perform a live analysis on a piece of malware.

The second machine will be the monitoring machine. Here we will only ever store and hash malware specimen. Be sure to NEVER run, load and or debug any malware on this machine. Everything must be READ-ONLY!

The only thing to really change is in the networking if you are using real machines or virtual machines. If you are using real machines and not familiar with sniffing on a switched network OR the setting up of SPAN ports on a switch. I suggest you use a simple hub, as all packets will be replicated to each port allowing the easy capture of traffic from the victim machine. Be sure to make sure your monitoring machine is fully patched, some malware is weaponized.

Meaning they have the ability to scan for and infect other host on the network. I have seen malware in the wild that would run their own DHCP server handing out valid addresses but a poisoned DNS server under an attacker's control.

There is a catch 22 involved here... It is a good idea to allow the malware to interact with its server out in the wild & scary internet for complete inspection, though doing so could very well could put you in the middle of a WW3 battle with the malware bot herders. Ever heard that saying about not meddling in the affairs of a system admin? Trying angering a bot herder…. ;D

Physical Machines

- Minimal 2 machines.
  (1 machine for running malware) (1 machine for running network analysis)
- Norton Ghost or some other imaging software
- Hub – Switch (networking) hub is probably the best choice, makes the sniffing of traffic much easier ;)

Pros	Cons
• Essentially every piece of malware/virus will operate* • Tighter control to ensure the malware does not escape the environment into the host.	• Can be costly on hardware. • Requires more physical space for hardware. • Takes longer to restore pristine image

Virtual Machines

- Minimal 1 Real system
- Virtualization software:
  VMware [Workstation - Player, Server (free)]
  XEN (free)
  QEMU (free)
  Microsoft Virtual PC (free)

Pros	Cons
• Faster setup times • Non-persistent image support	• Virtualization detection from malware

Organizing the Malware Collection – Keeping things tidy

once your collection starts to grow, if it has not already... Things will tend to get pretty messy. It is a good idea to keep your collection organized from the beginning, as this will greatly pay off in the end.

I have found a scheme online for managing viruses and modified it of sorts for keeping my collections clean and organized. Reverse engineering of malware can be a very time consuming process and require lots of notes and logging of details. Using the methods and naming convention below, this will allow for an easier to manage collection for management of the binaries as well as data and notes pertaining to each file in the collection.

I like to keep my collection organized by starting out with the family name of the malware. Under each family name I will then create sub directories for each variant that is classified to be within a specific family of malware.

Example:

```
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/notes.txt
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/file_exe
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/unpacked/file_exe
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/unpacked/file_exe.md5
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/unpacked/file_exe.sha1
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/unpacked/file_exe.strings
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/file_exe.md5
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/file_exe.sha1
Backdoor.Win32.Kbot/Backdoor.Win32.Kbot.al/file_exe.strings
```

`*always add some extra character on the end of malware extension to prevent accidental execution.`

Yes it does happen... Yes, I have done it before... =)

Analyzing Malware – The first steps...

Now that we have a safe environment built for analyzing some malware, and assuming that you have some malware collected already. Let's begin the steps of analyzing to see what we can learn about its mysterious ways. We first choose our initial method of attack at analyzing the malware.

Static VS Dynamic Analysis

When analyzing malware there is generally 2 approaches you can take. In most cases both may be required, where as in others one may be enough.

Static Analysis

Static analysis will always be your safest bet when analyzing malware.
This is mainly due to the reason that ALL of the analysis is done as read-only
per se. No actual executing of the malware is ever performed in a live environment.
Performing a static analysis you will not need to go to such great lengths as compared
to a dynamic analysis of malware on a live system.

Dynamic Analysis

Dynamic analysis can get a bit more interesting for you as a researcher as well as exposing yourself or others to further risk of attack. If you must allow your victim machine access to the net during analysis for capturing traffic to the mother ship, I would suggest making use of the TOR network. The bot herder may notice you tampering around and turn the BOT net around on you. This never equals good times...

Fingerprinting Malware Binaries

The very first step before analyzing any piece of malware is the renaming & hashing of the original binary. The process of renaming the malware is a good habit to get into this will prevent the accidental execution of the malware. Assuming we have a file *file.exe* we are attempting to analyze.

```
  Original: file.exe New: file_exe
```

For the purpose of hashing we will use tools like MD5, SHA1. Hashing a file using the utilities is fairly simple to do. Hashing of the file is pretty essential when you are analyzing Malware. This will ensure the malware file its self does not change during analysis.

To keep things organized, I store the file hash in a file with the following naming convention:
File_ext.hashType Example: *file_exe.md5*

We first hash the file with the following process.

```
Magikh0e@ihtb.org:~$ md5sum file.exe > file_exe.md5
d41d8cd98f00b204e9800998ecf8427e file.exe

Magikh0e@ihtb.org:~$ sha1sum file.exe > file_exe.sha1
da39a3ee5e6b4b0d3255bfef95601890afd80709 file.exe
```

Gathering Strings from binaries

After hashing the binary, I always tend to search for some low hanging fruit. Using the strings command is a perfect way to grab all of the human readable/printable characters from the file and store them away in a separate file for later analysis.

Gathering strings from a binary can be performed in several different ways. Linux systems with GNU tools come with a utility built in which can be accessed using the strings command.

Note: In some cases no strings will be returned, this usually means the file has been packed.

```
Magikh0e@ihtb.org:~$ strings file_exe > file_exe.strings
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used…..
```

Automating the process

I have hacked up a quick Perl script to auto-mated the process of collecting the MD5 and SHA1 hash of the binary, then gather printable characters found within the binary and log them to a file. The purpose of the script is to give you an idea at creating some of your own automated way's of managing/analyzing your collection. If you come up with anything you think would be useful, I would love to check it out…

```
#!/usr/bin/perl
# Script for collecting hashes and strings from a malware binary.

use strict;
use warnings;

my $MD5 = '/usr/bin/md5sum';
my $SHA1 = '/usr/bin/sha1sum';
my $STRINGS = '/usr/bin/strings';
my $mw_file;
$mw_file = $ARGV[0];

if (!$mw_file) { print "Malware File: ";
chomp($mw_file = <STDIN>); }

if (-e $mw_file) {
system("$MD5 $mw_file > $mw_file.md5");
system("$SHA1 $mw_file > $mw_file.sha1");
system("$STRINGS $mw_file > $mw_file.strings");
} else { die "File: $mw_file - does not exist: $0"; }
```

Identifying Known Malware

I've always hated to re-invent the wheel or see someone else suffer trying. Having said that, now we have renamed and hashed our malware specimen. It is time to pull out your favorite anti-virus scanner.

Start scanning the binary you are analyzing using different AV scanners. We do this for several purposes. One the file may have already been identified and documented, voiding the need for manual analysis.

There is a fair amount of good online resources for this purpose and help considerably when speeding things up. One of the best sites I know of for this purpose is http://virusscan.jotti.org/. Using this site you will have the ability to submit a file and then have it scanned using 21 of the leading anti virus vendors. I tend to live by the motto of security being a process, not a product. Make sure to use more than one scanner.

Online Virus scanning

http://virusscan.jotti.org
http://www.virustotal.com

Known Malware

If you are lucky, the AV will be able to detect the malware you are analyzing. Upon detection the chances are very high of someone on the internet documenting every detail about the malware will already be posted. Google is your friend.

Unknown Malware

Now you must have a binary that you have suspicions of, but none of the current anti-virus scanner definitions are picking up any malware infection within the binary it's self. Now is where things will get tricky, interesting, frustrating and mentally anguishing all at the same time. Have no fear; there is a vast variety of tools to help you accomplish the task of debugging, disassembling and analyzing binaries of unknown origins and or purposes.

Unknown Malware, Packers and Cryptors… oh my!

The first steps towards identifying an unknown piece of malware is too first analyze the binary with a fine tooth comb. Chances are the binary has been packed with some sort of packer. If luck is on your side the packer being used will already have been documented with details on the unpacking of the executable.

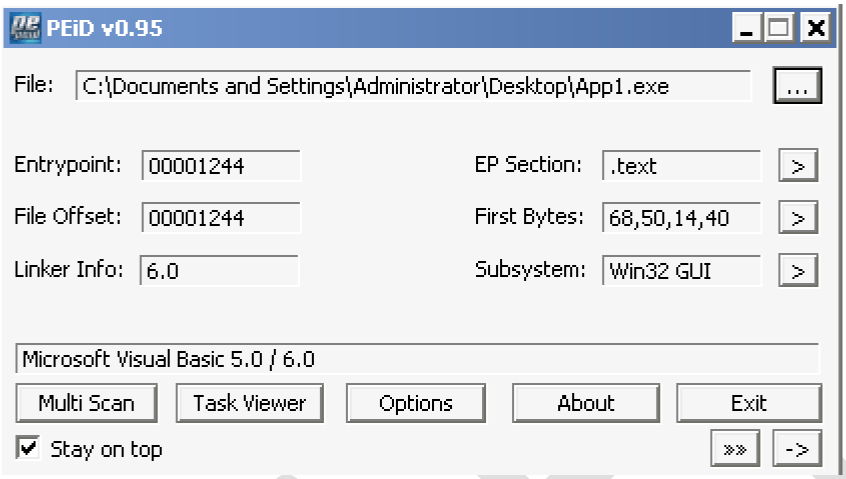
Now you are most likely wondering just how to identify what packer is being used. For this purpose we will be using a utility called PEiD by: Jibz, Qwerton, snaker, xineohP. At the time of this writing the latest version available is 0.95.

Packed Malware

Malware authors often use packer software in order to evade detection and to make sure the malware has a smaller footprint so it can squeeze in places and remain hidden for as long as possible. Packers can be quite tricky in identifying and more importantly, un-packing. A tool as the one below is used in order to attempt identifying which packer a binary is currently using. Once identified hopefully finding the packer or resources for manual unpacking will be easier to find.

PEiD

PEiD detects most common packers, cryptors and compilers for PE files. It can currently detect more than 600 different signatures in PE files.



PEiD v0.95

File: C:\Documents and Settings\Administrator\Desktop\App1.exe

Entrypoint: 00001244

EP Section: .text

File Offset: 00001244

First Bytes: 68,50,14,40

Linker Info: 6.0

Subsystem: Win32 GUI

Microsoft Visual Basic 5.0 / 6.0

☑ Stay on top

If you are lucky your job will be made tremendously easier and there will not be a packer being used on the executable; however that would be in the perfect world. Most malware authors do not like people poking around in and with their code. So they will go to great lengths with packers, cryptors and other obfuscation techniques they manage to conjure up in order to keep the prying eyes away from the prize.

So, you ran into a packed file huh?
Once you have identified that you are dealing with a packed piece of malware, hopefully PEiD will be able to identify the packer currently being used. If so, this will tremendously speed up the efforts at reverse engineering the piece of malware successfully. Once you have determined the name of the packer, we rush away to several places on the web with a vast resource of information on specific packers and how to unpack them. In some cases you will be unable to successfully unpack a binary. In these cases your only option is to perform dynamic analysis while the malware is running. More on this later…

Manual Un-Packing Malware

When attempting to unpack binaries by hand manually we will be on the hunt for the OEP entry. Sounds easy enough, yeah? ;) Olly and IDA both have some plugins to help in the finding of the OEP, other times this is a pain staking manual process. When attempting to locate the OEP, keep an eye out for the following:

- JMPs or CALLs to EAX.
  This may point you to the OEP, this JMP or CALL could possibly be preceded by POPA or POPAD.
- Tricky jumps via SEH, CALL, RET etc…
- If the packer you are dealing with uses SEH, You can anticipate OEP by tracking stack areas used to store the packers' handlers.
- Breaking on the un-packers calls to LoadLibraryA or GetProcAddress may help in getting closer to OEP.
- When unpacking in OllyDbg, try SFX (bytewise) and OllyDump's "Find OEP by Section Hop".

Unpacking a binary quick and dirty

Infect the victim system and dump the memory using LordPE or OllyDump.

Unpacking a binary butcher style

Find the OEP 'original entry point' after the un-packer has been executed.

Packer Analysis Database: http://www.openrce.org/reference_library/packer_database
MW-Blog: http://www.teamfurry.com/wordpress/

The above sites are some pretty good resources that I have found on the net with information related to packers. The MW-Blog seems to have not been updated for awhile, at the time of this writing the last update was on June 15th 2009. None the less still contains some good information on un-packing packers. Most of the blog postings seem to use IDA Pro for the actual unpacking process which leads us into the next section…

Disassembling – (static analysis)

The art of disassembly is another paper on it's own to cover in detail and give it due justice. However that is a bit out of scope for this paper, so we will briefly cover it and some tools you can use along with tips to help you out along the way.

Almost all of the software you see available today is written using a high level language like C, C++, and Delphi etc... Same goes for the malware... These languages generally speaking, are what are known as a compiled language that will turn the high-level code into low-level code that a computer can understand. This is what we know as Assembly. Not to be confused with de-compilers, Disassemblers will take a binary file that could have been written in any available compiled language, and disassemble the binary into assembly source code. While a de-compiler takes a binary file generated by a specific compiler and attempts to reverse the binary back into the high-level code it was created from originally.

Common x86 Registers and Uses

EAX	Addition, multiplication, function results
ECX	Counter
EBP	Base for referencing function arguments (EBP+value) and local variables (EBP-value)
ESP	Points to the current "top" of the stack; changes via PUSH, POP, and others
EIP	Points to the next instruction
EFLAGS	Contains flags that store outcomes of computations (e.g., Zero and Carry flags)

Picking a good disassembler

A good disassembler needs to have the ability to accurately distinguish between data and code of a binary. While a good de-compiler will need to do this and have the ability to understand what code construct in the original high-level language generated the code to begin with.

IDA Pro

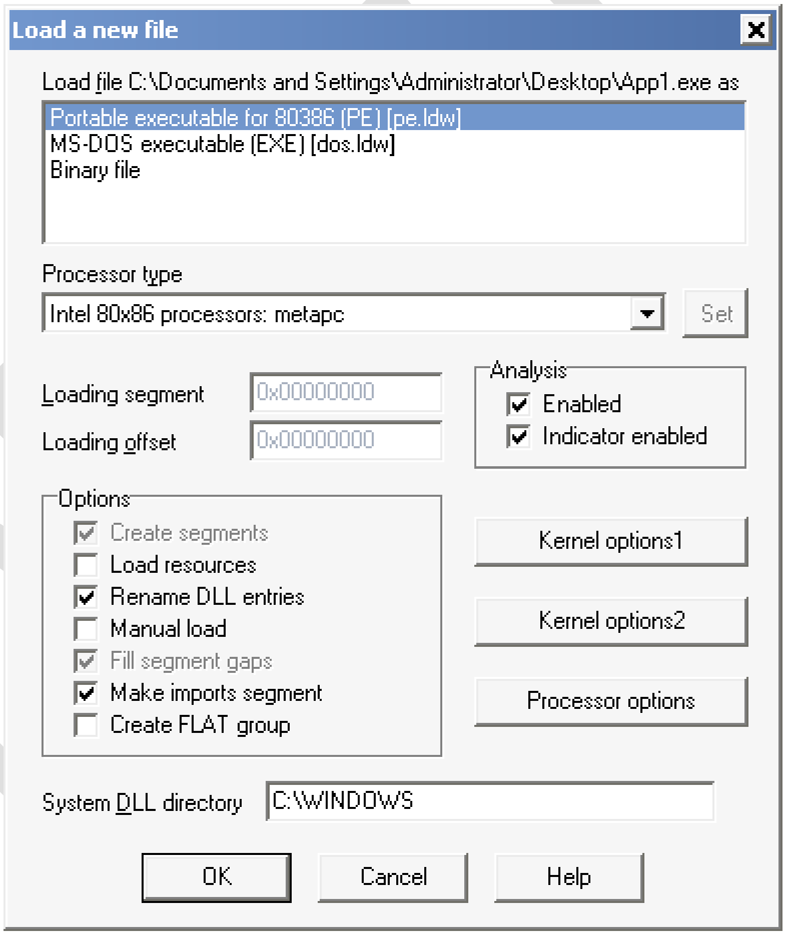
For static analysis my personal choice in preference is IDA Pro, why IDA Pro you may ask?

Well IDA Pro is nor a de-compiler or a disassembler... Think of IDA as the love child if W32dasm and OllyDBG had a baby... IDA stands for the interactive disassemble, and it can withstand living up to its name. IDA also has loads of features that you could classify it as a de-compiler due to a feature known as FLIRT.

If you have ever dabbled in C you will know that every program ends up using some functions that are supplied by the compiler it's self, or as a part of the Win32 API as an example. Take a look at a printf() statement, any C program that makes a call to printf will have the same piece of code inside them. During compile time the process of the compiler will in-turn link the code for the function from its included library. If it is not so obvious yet, this will allow for the disassemble to recognize the code patterns of specific functions then add the ability of mapping a name to it that is somewhat meaningful.

If you ever dabbled in reverse engineering you can see how this will become useful, ever spent an hour tracing through a function only to find out later that was only that specific compilers variant of fseek(). To me this is what sets IDA aside from others with the FLIRT abilities. The FLIRT libraries come with a huge set of signatures for specific functions from the various available compilers on the internet. Best of all they do not stop at C alone, there is also signatures for compilers like Pascal, Delphi, VB and more… To learn more about FLIRT, check out the IDA page in resources. FLIRT is just one of the many great things about IDA, there is a vast resource of plugins you can find online or learn how to create your own.

Loading of a file in IDA Pro…



Load a new file

Load file C:\Documents and Settings\Administrator\Desktop\App1.exe as

Portable executable for 80386 (PE) [pe.ldw]
MS-DOS executable (EXE) [dos.ldw]
Binary file

Processor type: Intel 80x86 processors: metapc

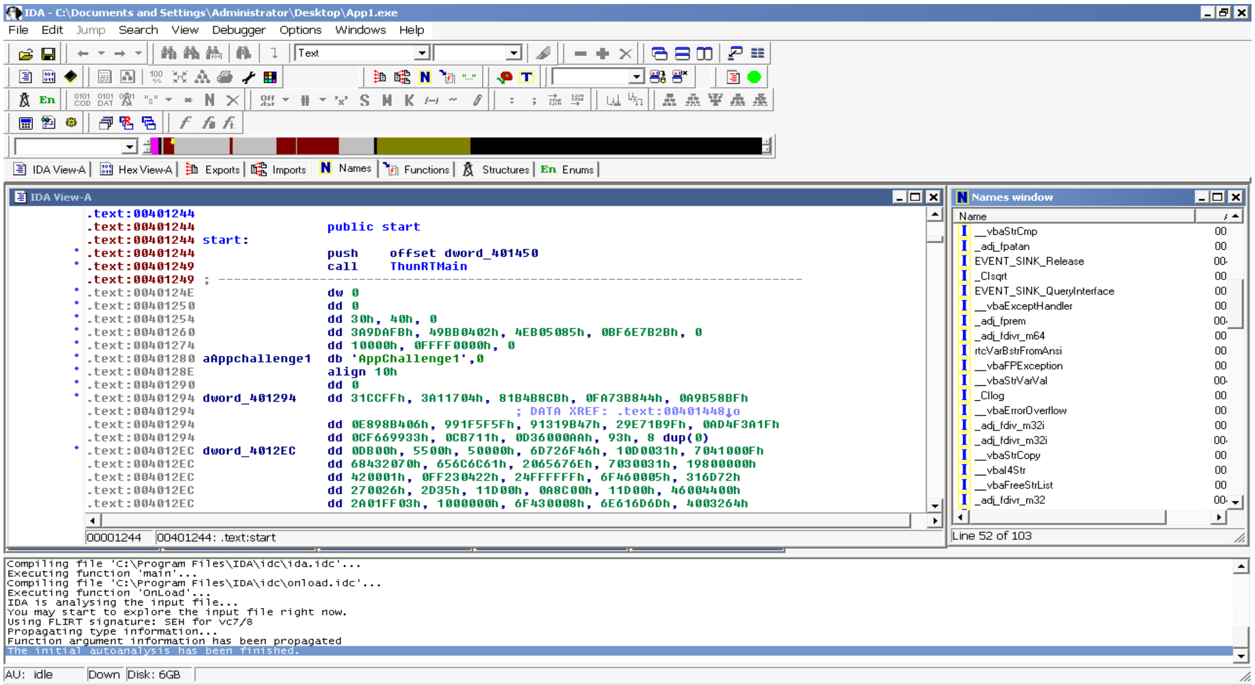
Loading segment: 0x00000000
Loading offset: 0x00000000

Analysis
☑ Enabled
☑ Indicator enabled

Options
☑ Create segments
☐ Load resources
☑ Rename DLL entries
☐ Manual load
☑ Fill segment gaps
☑ Make imports segment
☐ Create FLAT group

System DLL directory: C:\WINDOWS

IDA Pro main window after a binary has been loaded, the green circle also indicates that the IDA analysis has been completed…


IDA - C:\Documents and Settings\Administrator\Desktop\App1.exe
File Edit Jump Search View Debugger Options Windows Help
Text
IDA View-A | Hex View-A | Exports | Imports | Names | Functions | Structures | Enums
IDA View-A
Names window
Name

```
.text:00401244
.text:00401244                 public start
.text:00401244 start:
.text:00401244                 push    offset dword_401450
.text:00401249                 call    ThunRTMain
.text:00401249 ; ---------------------------------------------------------------------------
.text:0040124E                 dw 0
.text:00401250                 dd 0
.text:00401254                 dd 30h, 40h, 0
.text:00401260                 dd 3A9DAFBh, 49BB0402h, 4EB05085h, 0BF6E7B2Bh, 0
.text:00401274                 dd 10000h, 0FFFF0000h, 0
.text:00401280 aAppchallenge1  db 'AppChallenge1',0
.text:0040128E                 align 10h
.text:00401290                 dd 0
.text:00401294 dword_401294    dd 31CCFFh, 3A11704h, 81B4B8CBh, 0FA73B844h, 0A9B58BFh
.text:00401294                                         ; DATA XREF: .text:00401448↓o
.text:00401294                 dd 0E898B406h, 991F5F5Fh, 91319B47h, 29E71B9Fh, 0AD4F3A1Fh
.text:00401294                 dd 0CF669933h, 0CB711h, 0D36000AAh, 93h, 8 dup(0)
.text:004012EC dword_4012EC    dd 0DB00h, 5500h, 50000h, 6D726F46h, 10D0031h, 7041000Fh
.text:004012EC                 dd 68432070h, 656C6C61h, 2065676Eh, 7030031h, 19800000h
.text:004012EC                 dd 420001h, 0FF230422h, 24FFFFFFh, 6F460005h, 316D72h
.text:004012EC                 dd 270026h, 2D35h, 11D00h, 0A8C00h, 11D00h, 46004400h
.text:004012EC                 dd 2A01FF03h, 1000000h, 6F430008h, 6E616D6Dh, 4003264h
00001244  00401244: .text:start
```

```
I __vbaStrCmp                  00
I _adj_fpatan                  00
I EVENT_SINK_Release           00
I _CIsqrt                      00
I EVENT_SINK_QueryInterface    00
I __vbaExceptHandler           00
I _adj_fprem                   00
I _adj_fdivr_m64               00
I rtcVarBstrFromAnsi           00
I __vbaFPException             00
I __vbaStrVarVal               00
I _CIlog                       00
I __vbaErrorOverflow           00
I _adj_fdiv_m32i               00
I _adj_fdivr_m32i              00
I __vbaStrCopy                 00
I __vbaI4Str                   00
I __vbaFreeStrList             00
I _adj_fdivr_m32               00
Line 52 of 103
```

```
Compiling file 'C:\Program Files\IDA\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Using FLIRT signature: SEH for vc7/8
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
AU:  idle   Down   Disk: 6GB
```

IDA Pro Interface

`IDA View A – Action     name: WindowOpen`

Disassembly window

`Exports – Action     name: OpenExports`

Exports Window

`Imports – Action     name: OpenImports`

Imports Window

`Names – Action     name: OpenNames`

Names Window

The GUI version displays a small icon for each name:

L (dark blue) - library function	F (dark blue) - regular function
C (light blue) - instruction	A (dark green) - ascii string
D (light green) – data	I (purple) - imported name

```
Functions - Action    name: OpenFunctions
```

Functions Window

Listed for each function are:

```
        - function name
        - segment that contains the function
        - offset of the function within the segment
        - function length in bytes
```

The last column of this window has the following format:

```
        R - function returns to the caller
        F - far function
        L - library function
        S - static function
        B - BP based frame.
            IDA will automatically convert all frame pointer
            [BP+xxx]operands to stack variables.
        T - function has type information
        = - Frame pointer is equal to the initial stack pointer.
            In this case the frame pointer points to the bottom  of the frame
        M - reserved
        S - reserved
        I - reserved
        C - reserved
        D - reserved
        V – reserved
```

```
Structures - Action    name: OpenStructures
```

Structures Window

You can modify structure definitions here: add/rename/delete structures, add/delete/define structure members.

```
Enums - Action    name: OpenEnums
Enums Window
```

You can modify enum definitions here: add/edit/delete enums, add/edit/delete enum members (i.e. user-defined symbolic constants)

IDA Pro Tutorials

http://www.hex-rays.com/idapro/idasupport.htm

http://ebook-net.blogspot.com/2008/05/reverse-engineering-with-ida-pro.html

http://www.woodmann.com/crackz/Tutorials/IdaTut.zip

http://www.hex-rays.com/idapro/debugger/gdb_qemu.pdf

http://www.hex-rays.com/idapro/debugger/gdb_vmware_linux.pdf

http://www.hex-rays.com/idapro/debugger/gdb_vmware_winkernel.pdf

IDA Pro Shortcuts

Here we will list some handy shortcuts…

Text search	*Alt+T*
Show strings window	*Shift+F12*
Show operand as hex value	*Q*
Insert comment:	
Follow jump or call in view	*Enter*
Return to previous view	*Esc*
Go to next view	*Ctrl+Enter*
Show names window	*Shift+F4*
Display function's flow chart	*F12*

Display graph of function calls	*Ctrl+F12*
Go to program's entry point	*Ctrl+E*
Go to specific address	*G*
Rename a variable or function	*N*
Show listing of names	*Ctrl+L*
Display listing of segments	*Ctrl+S*
Show cross-references to selected function	*Select function name » Ctrl+X*
Show stack of current function	*Ctrl+K*

Debugging and Debuggers – (dynamic analysis)

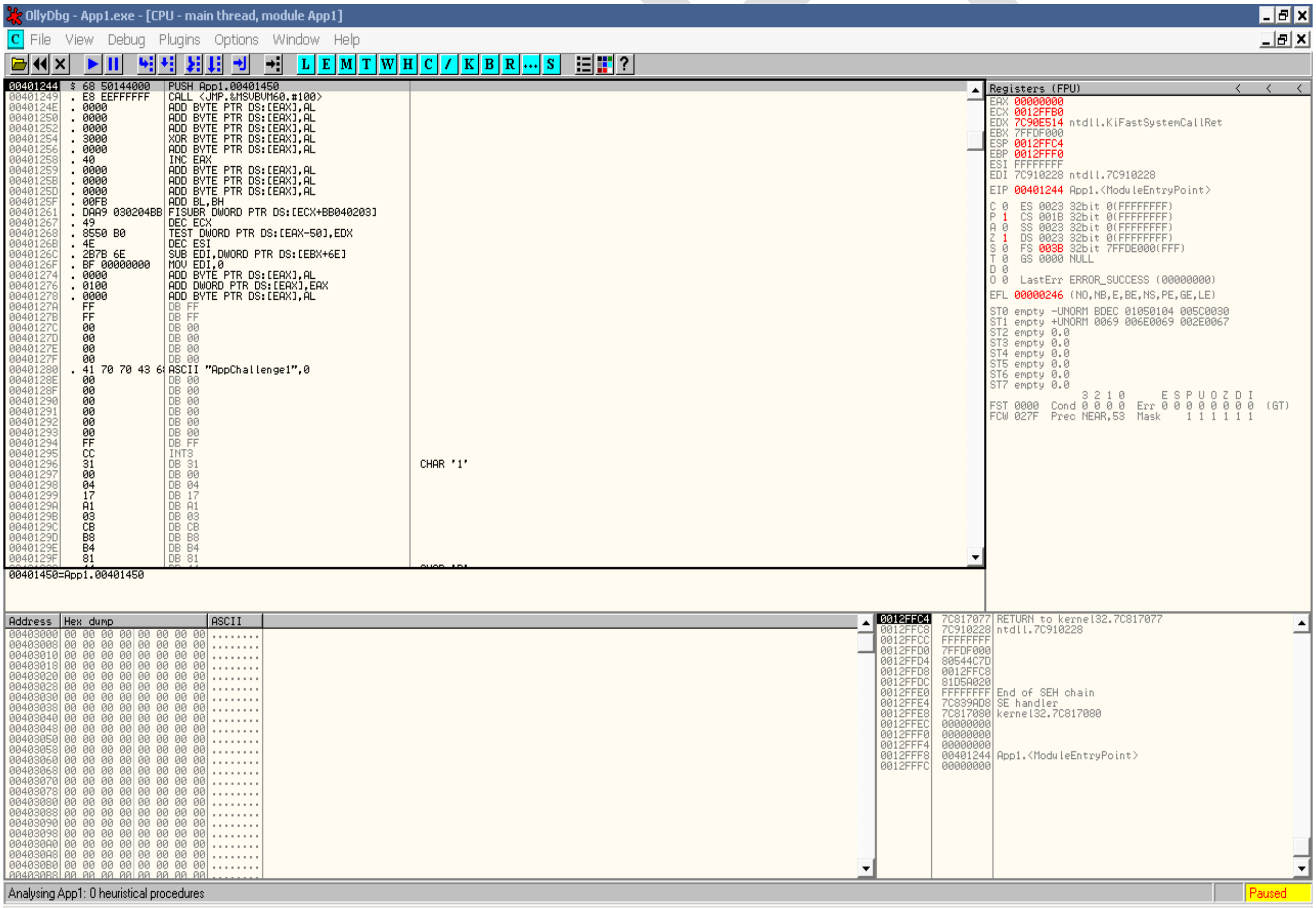
Debugging and debuggers themselves play another important part in analyzing malware. These are typically used once you get to the dynamic analysis portion when analyzing a new piece of malware in your collection.

When it comes to debuggers I am torn between two like Michael Jordan and baseball?? Good you are still paying attention. ;)

Anyways, the two debuggers of my preference for analyzing malware is Immunity DBG and Olly DBG. They both are very good and what they do with some advantages and disadvantages over one another. Which we will not go into here… For sole purpose of simplicity and more information online, we will mainly be focusing on Olly DBG for this paper.

OllyDBG

OllyDBG main window…


OllyDbg - App1.exe - [CPU - main thread, module App1]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
?
Registers (FPU)
Paused

```
00401244  $ 68 50144000    PUSH App1.00401450
00401249  . E8 EEFFFFFF    CALL <JMP.&MSVBVM60.#100>
0040124E  . 0000           ADD BYTE PTR DS:[EAX],AL
00401250  . 0000           ADD BYTE PTR DS:[EAX],AL
00401252  . 0000           ADD BYTE PTR DS:[EAX],AL
00401254  . 3000           XOR BYTE PTR DS:[EAX],AL
00401256  . 0000           ADD BYTE PTR DS:[EAX],AL
00401258  . 40             INC EAX
00401259  . 0000           ADD BYTE PTR DS:[EAX],AL
0040125B  . 0000           ADD BYTE PTR DS:[EAX],AL
0040125D  . 0000           ADD BYTE PTR DS:[EAX],AL
0040125F  . 00FB           ADD BL,BH
00401261  . DAA9 030204BB  FISUBR DWORD PTR DS:[ECX+BB040203]
00401267  . 49             DEC ECX
00401268  . 8550 B0        TEST DWORD PTR DS:[EAX-50],EDX
0040126B  . 4E             DEC ESI
0040126C  . 2B7B 6E        SUB EDI,DWORD PTR DS:[EBX+6E]
0040126F  . BF 00000000    MOV EDI,0
00401274  . 0000           ADD BYTE PTR DS:[EAX],AL
00401276  . 0100           ADD DWORD PTR DS:[EAX],EAX
00401278  . 0000           ADD BYTE PTR DS:[EAX],AL
0040127A    FF             DB FF
0040127B    FF             DB FF
0040127C    00             DB 00
0040127D    00             DB 00
0040127E    00             DB 00
0040127F    00             DB 00
00401280  . 41 70 70 43 6  ASCII "AppChallenge1",0
0040128E    00             DB 00
0040128F    00             DB 00
00401290    00             DB 00
00401291    00             DB 00
00401292    00             DB 00
00401293    00             DB 00
00401294    FF             DB FF
00401295    CC             INT3
00401296    31             DB 31                                    CHAR '1'
00401297    00             DB 00
00401298    04             DB 04
00401299    17             DB 17
0040129A    A1             DB A1
0040129B    03             DB 03
0040129C    CB             DB CB
0040129D    B8             DB B8
0040129E    B4             DB B4
0040129F    81             DB 81
00401450=App1.00401450
```

```
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C910228 ntdll.7C910228

EIP 00401244 App1.<ModuleEntryPoint>

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDE000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty -UNORM BDEC 01050104 005C0030
ST1 empty +UNORM 0069 006E0069 002E0067
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Address  Hex dump                 ASCII
00403000 00 00 00 00 00 00 00 00 ........
00403008 00 00 00 00 00 00 00 00 ........
00403010 00 00 00 00 00 00 00 00 ........
00403018 00 00 00 00 00 00 00 00 ........
00403020 00 00 00 00 00 00 00 00 ........
00403028 00 00 00 00 00 00 00 00 ........
00403030 00 00 00 00 00 00 00 00 ........
00403038 00 00 00 00 00 00 00 00 ........
00403040 00 00 00 00 00 00 00 00 ........
00403048 00 00 00 00 00 00 00 00 ........
00403050 00 00 00 00 00 00 00 00 ........
00403058 00 00 00 00 00 00 00 00 ........
00403060 00 00 00 00 00 00 00 00 ........
00403068 00 00 00 00 00 00 00 00 ........
00403070 00 00 00 00 00 00 00 00 ........
00403078 00 00 00 00 00 00 00 00 ........
00403080 00 00 00 00 00 00 00 00 ........
00403088 00 00 00 00 00 00 00 00 ........
00403090 00 00 00 00 00 00 00 00 ........
00403098 00 00 00 00 00 00 00 00 ........
004030A0 00 00 00 00 00 00 00 00 ........
004030A8 00 00 00 00 00 00 00 00 ........
004030B0 00 00 00 00 00 00 00 00 ........
004030B8 00 00 00 00 00 00 00 00 ........
```

```
0012FFC4  7C817077  RETURN to kernel32.7C817077
0012FFC8  7C910228  ntdll.7C910228
0012FFCC  FFFFFFFF
0012FFD0  7FFDF000
0012FFD4  80544C7D
0012FFD8  0012FFC8
0012FFDC  81D5A020
0012FFE0  FFFFFFFF  End of SEH chain
0012FFE4  7C839AD8  SE handler
0012FFE8  7C817080  kernel32.7C817080
0012FFEC  00000000
0012FFF0  00000000
0012FFF4  00000000
0012FFF8  00401244  App1.<ModuleEntryPoint>
0012FFFC  00000000
```

Analysing App1: 0 heuristical procedures

Note: I did not forget about gdb, we will be using some linux tools soon for performing some analysis.

OllyDBG Shortcuts

Here we will list some handy shortcuts…

Step into instruction	*F7*
Step over instruction	*F8*
Execute till next breakpoint	*F9*
Execute till next return	*Ctrl+F9*
Show previous executed instruction	*-*
Show next executed instruction	*+*
Return to previous view	*
Show memory map	*Alt+M*
Follow expression in view	*Ctrl+G*
Insert comment	*;*
Follow jump or call in view	*Enter*
Show listing of names	*Ctrl+N*
New binary search	*Ctrl+B*
Next binary search result	*Ctrl+L*
Show listing of software breakpoints	*Alt+B*
Assemble instruction in place of selected one	*Select instruction » Spacebar*
Edit data in memory or instruction opcode	*Select data or instruction » Ctrl+E*
Show SEH chain	*View » SEH chain*
Show patches	*Ctrl+P*

Defeating Malware Defenses

Most malware nowadays implement some sort of defense aimed at defeating analysis of the binary or any sort of reverse engineering in order to evade detection. Along with the evasion of detection, packing techniques are also used to make the malware smaller and easier to cram in places. As we have mentioned earlier on that some types of malware have the ability to detect if they are running within an emulated environment or not. Malware gets trickier and more obfuscated by the day; the creators are even more insane with new creations...

They also have the ability to detect well known debuggers running on the host system and/or implement obfuscation techniques within the code its self. I will include a tutorial in the references for a great read on anti-debugging tips, tricks and other techniques used.

OllyDBG undercover

Conceal OllyDbg **using** HideOD and OllyAdvanced.
http://www.openrce.org/downloads/details/238/Hide_Debugger
http://www.openrce.org/downloads/details/241/Olly_Advanced

IDA Stealth

IDA Stealth is a plugin which aims to hide the IDA debugger from most common anti-debugging techniques. The plugin is composed of two files, the plugin itself and a dll which is injected into the debuggee as soon as the debugger attaches to the process. The injected dll actually implements most of the stealth techniques either by hooking system calls or by patching some flags in the remote process.
http://newgre.net/idastealth

Detecting Virtualized Environments

As we have discussed earlier on, a lot of malware that you see nowadays have the ability to detect that they are currently running within a virtual environment. Below we will look into the way's a virtual environment could be detected.

VMWare

VMWare is a bit easier to detect depending on your configuration. For the main part vmware-tools is usually always installed on the OS anyways. There is also a small C routine by the name of Jerry.c that works well for the most part, though does have some quirks. Jerry.c uses a simple communication channel that VMware has left open. As mentioned depending on the configuration, the administrator can change this channel. Chances are high of the OS being designed to act as a honey pot if this channel has been changed anyways…

You can think of Jerry.c is a first generation of it's kind. As with anything good, usually better improvements follow. This is where Scoopy comes into play, written by the same author as Jerry.c.

Scoopy is far more complex in the way it works as compared to Jerry.c. Scoopy inspects how the system is actually virtualized using the SIDT CPU instruction instead of a communication channel, as Jerry.c does. Another interesting write up on detecting virtualized operating systems if the red pill paper.

Jerry.c - http://www.trapkit.de/tools/index.html - Outdated now, see ScoopyNG.
Scoopy - http://www.trapkit.de/tools/index.html
Red Pill Paper: http://invisiblethings.org/papers/redpill.html

VMWare can also be detected via the MAC address being used. VMware by default will use a MAC address starting with 00-05-69. Though keep in mind, a MAC address can be changed fairly easy.

Wine

For wine we also have several options in order to detect being run within the environment. When you are using wine they have created a special dll entry to NTDLL.DLL called wine_nt_to_unix_file_name.

When you are using LoadLibrary and getProcAddress, you can determine if you are under a wine environment or not by checking for the presence of the new entry into NTDLL.DLL. Jerry.c also works under wine, just will state that vmware has been detected.

Registry Keys

Wine can also be detected by opening the following registry keys:
HKLM\Software\Wine or HKCU\Software\Wine

Last but not least. Open up any critical Windows file and locate the OEP, when running under a wine environment the function will disassemble to the following instructions:

```
.text:10001000              public start
.text:10001000 start        proc near
.text:10001000              mov     eax, 1
.text:10001005              retn    4
.text:10001005 start        endp
```

For the quick and dirty, Just search for the following binary string B8 01 00 00 00 C2 04 00 at .text:10001000 ;D

XEN

Xen can be detected fairly reliable using the WMI interface and querying the BIOS manufacturer information. Xen can also be detected via the MAC address being used. Xen host will use a MAC address starting with 00-16-3E

VirtualBox

Virtual Box MACs all start with 08:00:27

Code Examples

Detecting VMware:

```
procedure TForm1.btnJerryClick(Sender: TObject);
var a, b:cardinal;
begin
a:=0;
try
   asm
      push eax
      push ebx
      push ecx
      push edx
      mov eax, 'VMXh'
      mov ecx, 0Ah
      mov dx, 'VX'
      in eax, dx
      mov a, ebx
      mov b, ecx
      pop edx
      pop ecx
      pop ebx
      pop eax
   end;
except on E:Exception do ShowMessage(E.Message);
end;
if a=$564D5868 then
   begin
   ShowMessage('In VMware');
   case b of
   1  : ShowMessage('Express');
   2  : ShowMessage('ESX');
   3  : ShowMessage('GSX');
   4  : ShowMessage('Workstation');
   else ShowMessage('Unknown version')
   end;
   end
else
   ShowMessage('Native system');
end;
```

Detecting Wine:

```
function TForm1.IsRunningWine: boolean;
var hnd:THandle;
    wine_get_version: function : pchar; {$IFDEF Win32} stdcall;
{$ENDIF}
    wine_unix2fn: procedure (p1:pointer; p2:pointer); {$IFDEF Win32}
stdcall; {$ENDIF}
begin
result:=false;
hnd:=LoadLibrary('ntdll.dll');
if hnd>32 then
   begin
   wine_get_version:= GetProcAddress(hnd, 'wine_get_version');
   wine_unix2fn:= GetProcAddress(hnd, 'wine_nt_to_unix_file_name');
   if assigned(wine_get_version) or assigned(wine_unix2fn) then
      result:=true;
   FreeLibrary(hnd);
   end;
end;
```

Detecting VirtualBox:

This was written by the VirtualBox author, just detects the existence of VBoxService.exe.

```
function InVirtualBox:boolean;
var handle:THandle;
procinfo:ProcessEntry32;
begin
result:=false;
handle := CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
procinfo := sizeof(PROCESSENTRY32);

while(Process32Next(handle, procinfo)) do
begin
if POS("VBoxService.exe",procinfo.szExeFile)>0) then
begin
CloseHandle(handle);
result:=true;
exit;
end;
end;
CloseHandle(handle);
end;
```

Detecting Virtual PC:

```
function TForm1.IsRunningVirtualPC: boolean;
asm
  push ebp;
  mov ebp, esp;

  mov ecx, offset @exception_handler;

  push ebx;
  push ecx;

  push dword ptr fs:[0];
  mov dword ptr fs:[0], esp;

  mov ebx, 0; // Flag
  mov eax, 1; // VPC function number

  // call VPC
  db $0F, $3F, $07, $0B

  mov eax, dword ptr ss:[esp];
  mov dword ptr fs:[0], eax;

  add esp, 8;

  test ebx, ebx;

  setz al;

  lea esp, dword ptr ss:[ebp-4];
  mov ebx, dword ptr ss:[esp];
  mov ebp, dword ptr ss:[esp+4];

  add esp, 8;

  jmp @ret1;


@exception_handler:
  mov ecx, [esp+0Ch];
  mov dword ptr [ecx+0A4h], -1; // EBX = -1 ->; not running, ebx = 0 -
> running
  add dword ptr [ecx+0B8h], 4;  // ->; skip past the call to VPC
  xor eax, eax;                 // exception is handled

@ret1:
end;
```

Detect Virtualized guest based on MAC address

```
#!/usr/bin/perl
# Detect virtualized guest based on MAC address.
# Do not expect this method to be very reliable at all. MAC addresses
# can easily be changed. ;D


my $ifconfig = `/sbin/ifconfig -a eth0 2>&1 | /bin/grep HWaddr`;
my @info = split(/\s+/, $ifconfig);
print $info[4] . "\n";

if ($info[4] =~ /08:00:27/) { print "\nVirtualBox MAC Detected\n"; }
if ($info[4] =~ /00:16:3E/) { print "\nXen MAC Detected\n"; }
if ($info[4] =~ /00:05:69/) { print "\nVmware MAC Detected\n"; }
if ($info[4] =~ /00:0C:29/) { print "\nVmware ESX MAC Detected\n"; }
if ($info[4] =~ /00:50:56/) { print "\nVmware VC ESX MAC Detected\n";
}
```