Dynamic Analysis of Malware

Laurent Weber

Master Seminar
Research Group “Embedded Malware”
Prof. Dr. Thorsten Holz

July 26, 2010

Horst-Görtz Institute Ruhr-University Bochum

Contents

1 Introduction **1**

1.1 Static Analysis of Malware 1
1.2 Dynamic Analysis of Malware 1
1.3 Different Dynamic Malware Analysis Methods 2
1.3.1 Behavior-based Malware Analysis 2
1.3.2 Different Kinds of Dynamic Analysis 2
1.4 Challenges in Malware Analysis Taken From the Wild 3
1.4.1 Complexity 3
1.4.2 Lack of High Level Semantics 3
1.4.3 Whole System View 4

2 State of the Art **5**

2.1 Current Dynamic Analysis Tools 5
2.2 Useful Techniques for Malware Analysis 5
2.2.1 API Hooking 5
2.2.2 DLL Injection 7
2.2.3 Virtual Machine Introspection 8
2.3 Presentation of Different Malware Analysis Frameworks 9
2.3.1 General Overview of the Frameworks 9
2.3.2 BitBlaze Project 10
2.3.3 TTAnalyze 11
2.3.4 CWSandbox 13

3 Conclusion **17**

3.1 Security Application 17
3.2 Malware Analysis and Defense 17
3.3 Personal Conclusion 18

1 Introduction

Nowadays malware has become a very lucrative business, spying out confidential information like online-banking passwords, attacking targets with Distributed Denial of Services (DDOS) is an well known problem in the everyday life. Cyber-criminals use bots distributed on huge amounts of computers to remotely attack specific targets. Malware has become a million dollar business [Sym10] in the underground industry of the Internet.

Researchers want to find ways to analyse malware in a very quick and reliable manner, in order to be able to stop the spreading of a new malware. The cat and mouse game has been launched a long time ago. This paper will describe the actual state of the art in malware analysis, as well as the turnover from static to dynamic analysis. Furthermore, it will briefly introduce three frameworks: TTAnalyze, CWSandbox and the Bitblaze project.
In this paper we will only focus on analysis of windows binaries because this represents the majority of the malware.

1.1 Static Analysis of Malware

Malware is a generic term used for every kind of unwanted software, like backdoors, viruses and worms. In order to keep those malicious pieces of code away from our computers users use anti-virus software, that is able to detect known malware. These anti-virus software use signature based recognition, they scan binaries for byte sequences that are characteristic for different malware. Malware developers adapted to this approach and started to create polymorphic codes that are changing their signatures and through this are hard to detect for anti-virus software.

Researcher and anti-virus companies are using reverse engineering and manual disassembly methods to create those signatures and analyse the behaviour of the malware. Due to the increasing amount of malware sent to anti-virus companies every day, it is impossible for them to analyse the malware manually every time they get a new sample as it is too time consuming and very challenging, due to the complexity of the task and the lack of semantic information.

1.2 Dynamic Analysis of Malware

Due to the problems of manual malware analysis described in Section 1.1 other methods have to be found to analyse large amounts of malware in a quick and reliable way. Researchers are focusing on different kinds of dynamic malware analysis. Three of them will be described in this paper.

The process of dynamic malware analysis should give insight in the action performed by a given malware, mostly without manual reverse engineering, even if a reverse engineer would be desired in some cases to gain further information about the malware once the automated test revealed that it is an interesting piece of malicious code.

For this, automated tools that give an overview of the behaviour of the malware have to be created, and should follow different guidelines in order to give the best possible results and avoid false positives. Such tools should be able to log the security relevant behaviour of the malware and it should create a human readable report for further analysis by humans. Machine readable reports are a requirement too, they could be used as Intrusion Detection System signature.

1.3 Different Dynamic Malware Analysis Methods

1.3.1 Behavior-based Malware Analysis

Combining some well known techniques, that will be presented in Section 2.2 it is possible to trace the behaviour of a given malware in a quite accurate way. It allows one to know exactly what actions the malware performed:

- Modified or created files
- Registry modification
- Which DLLs were loaded before execution
- Accessed virtual memory
- Created processes
- Network connections opened and the packets transmitted
- What storage areas the malware accessed, installed services and kernel drivers as well as other information.

As you can imagine, this allows a very accurate tracing of the actions performed by the malware.

Unfortunately, frameworks using this technique face some limits too, so for example it is not giving any information on how the malware is programmed as you are working on a sort of blackbox. All you can do is monitor API calls to find out what the malware is up to.

In general, information gathered this way are sufficient to rank how dangerous the malware is.

1.3.2 Different Kinds of Dynamic Analysis

Using dynamic malware analysis, malware is executed in a simulated environment and it's behaviour gets analyzed after. There are mainly two methods to perform this:

- Analysis the difference between two snapshots of the system, one taken before the malware execution, the other after.

- Monitor the actions performed by the malware during its execution, this is realized by the usage of a debugger, for example.

Both methods have advantages and drawbacks.

The analysis of malware by comparing the state of the system before and after the execution of the malicious code is surely easier to implement and therefore less prone to implementing errors, but the results gained out of such an analysis is coarse-grained. For example it is impossible to know if a file has been created and deleted afterwards as it is not on the snapshot performed after the execution of the malware. So, this method might be good enough to assess the danger originating from the malware, but not for a detailed analysis.

One drawback of the dynamic malware analysis is that it is not able to run and analysis multiple malware samples at one time. So, you will only know one behaviour of the malware, for one special environment/moment. One can only trace one control flow at a time. Static malware analysis gives you an general overview over the malware. The code can be analyzed in a very detailed way and through this process researchers are able to know exactly how the malware is behaving.

1.4 Challenges in Malware Analysis Taken From the Wild

Of course, analysing malware seems a simple task, security researchers have done it for years. Why should it change now? What is the difference between the lab malware and the real malware picked from the wild? The most interesting part in malware analysis is that you are playing against a real person, a hacker, a person that may have very high skills and that will do everything to prevent you to break his malware. He will use evasion, obfuscation, cryptography, and many more techniques to defeat the malware researcher and anti-virus companies.

1.4.1 Complexity

The first challenge we will introduce is the complexity of the task. When reverse engineering a piece of malware you get confronted to the assembly code of the binary, and with the modern sets of instructions an x86 (hundreds of different instructions) or similar architecture provides, this can be a very complex task to understand. Each instruction can have complex semantics, like instructions that behave differently depending on the amount of operants they get, or single instructions loops or side effects like setting processor flags.

1.4.2 Lack of High Level Semantics

Working in the assembly language has other drawbacks, that are not comfortable for long code analysis. Researchers are mostly not very familiar with this level of abstraction as they are used to proper source code.

There are no functions, binary level does not allow that level of abstraction. Instead of functions jumps are used, which fast makes the code complex and far from easy to understand and follow.

Binary code does not contain buffers it has memory. There are no user-specified type and size in memory. If there are violations of some high-level semantics given by the source-code this is a problem of the high-level semantics, not of the binary code.

In binary there are no types so no new types can be created or used as there is no type constructor. The only available types are registers and memory.

So, this makes the drawback of binary code analysis pretty clear. It is not a very attractive part of work. In addition to this, there are several other parts that make the job even harder.

1.4.3 Whole System View

For many security application a whole system view is needed, this is the case for malware analysis on binary base. This concept consists of the analysis of operations on the system kernel and how the different processes interact. Monitoring a whole system is clearly a higher challenge than traditional analysis where only a single program needs to be analyzed.

2 State of the Art

2.1 Current Dynamic Analysis Tools

Dynamic malware analysis is a very current research topic. A lot of work is currently done in this field, new techniques and ideas pop up from all over the world at every conference. This paper will mainly put the focus on three of them: the Bitblaze project, TTAnalyze and CWSandbox. Anyhow the reader should be aware that there are a lot of other frameworks out there, for example *dirtbox* from Georg Wicherski from Kaspersky, or the *ANNE* framework of Gerard Wagener of SES ASTRA, and plenty more. Some are better known than others, this might be related to the results or the techniques used, but what are the important techniques related to the dynamic analysis of malware? This section will try to answer this question.

2.2 Useful Techniques for Malware Analysis

In the following section we will introduce three important techniques often used for automated malware analysis.

2.2.1 API Hooking

API stands for Application Programming Interface it is used by programmers to access system resources (files, network information, process or the registry as well as other resources.) The API is mostly used by applications to access the system resources rather than performing direct system calls. This allows us to *hook* the API in order to monitor the behaviour of the malware during its execution. The API is located in the windows system directory. The most important files are kernel32.dll, ntdll.dll,ws2_32.dll and user32.dll.

Intercepting a call to a function is called *hooking*, this is the task we want to perform in order to trace the behaviour of the malware and through this we are able to get a control flow of the malware. The concept of hooking is simple, each time an application accesses an API function it gets sent to a different location, where the modified code is located. That code performs some tasks, like for example storing what has been done and which parameters have been attached to the API call. Once the code has been executed the hook gives the control back to the unmodified API function. Depending on what action should be performed it is even imaginable that the hook refuses to give back control to the API function in order to prevent the system to take damage through actions performed by the malware, or to analyse the behaviour of the malware if a call fails.

API hooking has to be done in a careful way in order to be transparent and undetectable for the malware. It is never good if a malware detects that it is running in a simulated environment as then it is modifying its behaviour. This has been shown in several papers. Some malware even tries to exploit vulnerabilities in the emulators and attack the host system. [Fer07]

Now getting into the practical part of API hooking, there are several different methods known to intercept system calls mainly you can:

- Intercept execution chains inside user process,
- intercept execution chains inside kernel.

There are also other methods to perform hooking, but we will not focus on them in this paper, and only present the method used by CWSandbox, in-line code overwriting.

In-line code overwriting uses a method to directly overwrite the DLL's API function code loaded in the process memory. Due to this any call to the API, independent if it is linked implicitly or explicitly gets rerouted to the new code. Implicit linking occurs when an application loads a DLL before its own execution. There are 5 steps to overwrite the functions code:

- Create a target application in suspend mode. Windows will load and initialize the application and all implicitly linked DLLs. As the application is suspended Windows will not start the main thread so no code will be executed.
- Once the initialization is done an analysis on what functions have to be hooked (lookup in the DLLs export table) has to be done and the entry points of the code written down.
- Store the original code, in order to be able to reuse it later, for the reconstruction of the original API function.
- The first instructions of every API call functions have to be overwritten with a JMP or a call (Subroutine) to the location of our code.
- In order to have a fully operational API hooking a hooking of LoadLibrary and LoadLibraryEx API function has to be done, this allows explicit binding of DLLs.

  The same idea can be used if the malware loads DLLs dynamically during it's execution. See *Figure 1* for more details.

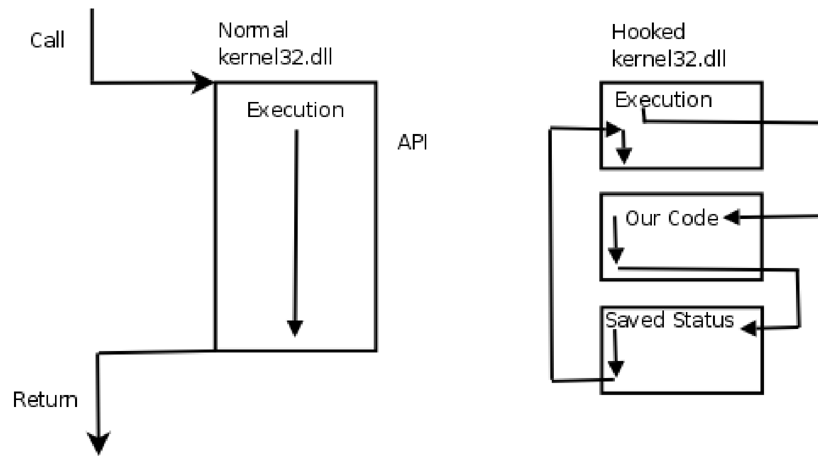

Figure 1: Example of a kernel32.dll hooked function

There are plenty of other API hooking methods, like for example system service hooking that operate at a lower level, anyway as none of the presented tools will use those techniques we will not focus on them. Anyway API hooking like described in detail in this section seems to be quite reliable and most malware is not noticing it. Tools like CWsandbox use this technique in order to get a good insight of the actions performed by the malware. The tool is placing a JMP to the custom code at the beginning of the API call.

A way to bypass the API hooking trap would be to call the kernel functions directly and avoid the usage of the API, anyhow this is uncommon as for this the programmer has to know exactly what version of operating system the victim is running and on what service pack patch level. As the goal of most malware is to infect a huge user base and not only targeted persons, direct kernel function calling is not easy to realise.

2.2.2 DLL Injection

A reusable and modular way to implement API hooking is without any doubt DLL code injection. This can be realized through API hooking with inline code overwriting. Therefore the applications has to be patched once it has been loaded into the memory. The address space of the malware has to contain our hooked function in order to be able to call the hook from inside the malware's address space. Therefore we use a technique called DLL injection. This is realised by a specialized thread located into the malware's memory allocation.

Windows allows us to implant and install API hook functions when we access other process's virtual memory and finally execute the code in a different process's context.

Two methods how this could be realised will be presented now:

- suspend a threat of the application that is running. Copy your code to the address space of that application, resume the execution of the application after changing the instruction pointer to the location of the copied code.
- Copy the code you want to execute in the application's address space and create a new threat in the target's process. The threat will hold the code location as a start address.

Through this methods we are able to inject and execute code in other processes. It makes sense to inject a DLL in the target's address space, and this is also the most used technique in praxis. Frameworks like CWSandbox use this techniques to make the malicious code load their DLL in their address space. The API hooks are installed into the target's address space and the API hooks are installed in the DLLs initialization routines, which are automatically called by Windows. Explicit linking is performed by LoadLibrary or LoadLibrary-Ex.

Finally all that has to be done is to create a new threat in the malicious application using CreateRemoteThreat function and setting the code address of the API function LoadLibrary as the new thread's starting address. When the malware executes the new threat it is calling LoadLibrary function automatically inside the target's context. As we know where kernel32.dll is located, as it is always on the same place, and also know where LoadLibrary is located, we can use these values for the target application.

2.2.3 Virtual Machine Introspection

A further technique that is widely used is *Virtual Machine Introspection* (VMI). This technique allows a monitoring of a virtual machine without risk. In fact, the machine gets monitored from outside. This has big advantages in comparison to standard *Intrusion Detection Systems* (IDS). The limits of IDS are, either you use a host based IDS, this can easily be attacked or evaded by malware that detects it. In general one could write down that an network based IDS is more resistant against attacks, as there is no direct connection to the host, but has a poorer view of the events in the host. Host-based IDS, on the contrary, have a very good view of what happens inside the host, but are more vulnerable to attacks, that get more and more popular with the increasing deployment of IDSes.

Virtual Machine Introspection allows us to have both advantages, a good resistance against attacks on the one hand, and full control of what is happening in the host on the other hand. Therefore the VMI uses access to the hardware-level state, for example the state of the physical memory pages and registers and also events like memory accesses and interrupts. The knowledge of these events and states allows us to map the events to OS-level semantics. Through this it is possible to monitor the host from outside, as if there was an host-based IDS. There are whole architectures based on this idea, like for example Livewire [GR03]. So we know that VMI

allows us to monitor the state of the hardware and from this we can deduce the software state, but what other advantages do we have?

An VMI allows us to have insight of an host even if it is completely compromised, this is not given with host based IDS. Furthermore, we have the ability to suspend the host at any moment to reset the state of it or change some settings. We can even disallow the access to resources dynamically, for example disallowing that the malware puts a network interface to promiscuous mode.

On the other hand, there has to be said that malware can implement some functionalities to detect if it is running on an virtual machine. If this is the case the malware can adapt its behaviour. One famous project implementing this detection functionality is the *Red Pill* project of Joanna Rutkowska.[1]

2.3 Presentation of Different Malware Analysis Frameworks

In the following section we will briefly introduce three different malware analysis frameworks that are well known in the actual research field of dynamic malware analysis.

2.3.1 General Overview of the Frameworks

The most frameworks the author of the paper crossed during his research on the topic as well as the three frameworks that will be described later on have the same architecture, at least, at a high level as presented in Figure 2:

[1] Joanna Rutkowska | http://invisiblethings.org/papers/redpill.html | November 2004

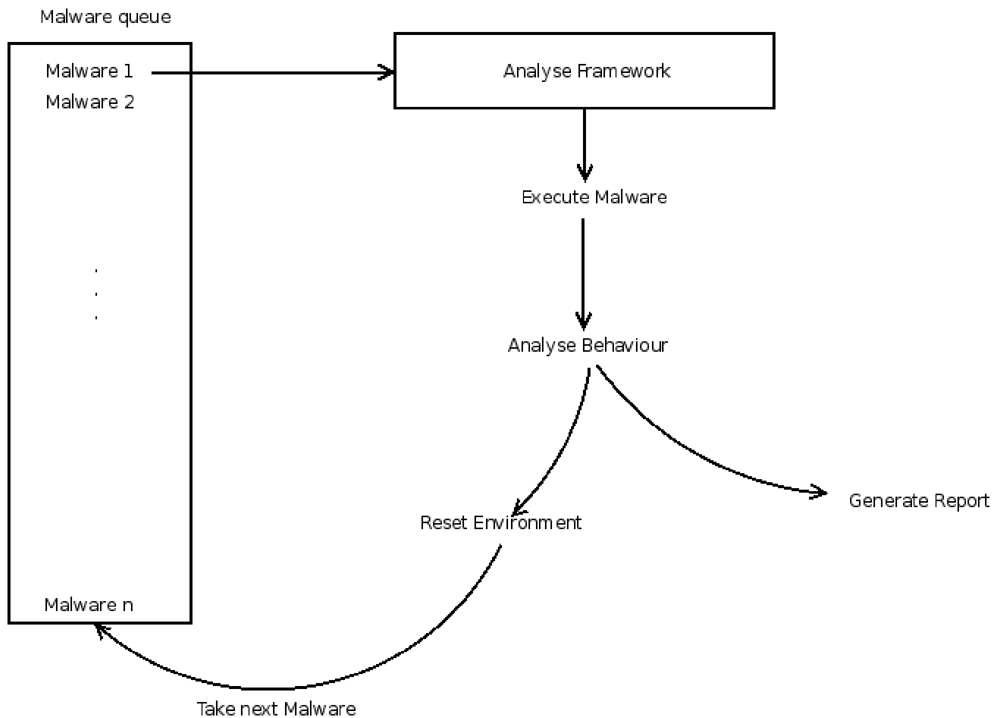

Figure 2: Life cycle of a malware in a framework

2.3.2 BitBlaze Project

The first framework we will introduce is the BitBlaze project. This framework, developed by the university of Berkley, is partly available as opensource download.

Fusion of Static and Dynamic Analysis

The Bitblaze project is a composition of different points of view in binary analysis. In opposition to lots of malware analysis frameworks it is not only relying on dynamic analysis but sets the focus on both, a full static analysis and a full dynamic analysis. This fusion of both concepts should provide a wide overview of the behaviour of the analyzed malware. In fact as we have discussed before, static analysis has advantages and drawbacks, like dynamic analysis. The static analysis covers more execution paths, but the complexity is often very high, as described in Section 1.4. Dynamic analysis hits its limits when you want to extract the whole code of the

malware. Therefore the Bitblaze project relies on the two analysis methods, the static performed by Vine and TEMU performing the dynamic analysis.

Architecture

The Bitblaze project is split into three parts:

- Vine for the static analysis, this part introduces an intermediate language, a front- and back-end.
- TEMU is responsible for the dynamic analysis
- Rudder mixes concrete and symbolic execution component

Vine consists of a platform dependent front-end and a platform independent back-end. The assembly code gets loaded into the front-end, translated to an intermediate language and send to the back-end, where different data is prepared and generate control flow graphs, optimisation, program verification, compute weakest preconditions and interfacing with decision procedures. Through the code generator even valid C code can be generated.

TEMU is the dynamic analysis part of the Bitblaze project. It is able to extract OS-level semantics, dynamic taint analysis of the whole system and has the ability to include plugins. TEMU is build on top of QEMU, a fast and portable dynamic translator [Bel05]. TEMU provides an API to be able to query the framework on its state, memory, registers, OS-Level semantics. Further more you can save and load the state of the machine at any moment. This allows multiple path examination in an eased way.

Rudder performs mixed execution and exploring of programs execution space. Rudder runs at the binary level. You can give several symbolic inputs for a binary program and Rudder will be able to perform a kind of fuzzing on the binary. This means it will detect multiple paths, assumed the paths are dependent on symbolic inputs, through its mix of concrete and symbolic execution feature. Through this, hidden paths that only get reached when certain condition occur could be detected. [PS08]

2.3.3 TTAnalyze

The second analysis framework we will present is TTAnalyze, it records native Windows system calls and Windows API functions that get invoked by the malicious program. TTAnalyze contains the following key functions:

- Complete operating system emulation build in software. This way the malware cannot easily detect the trap.
- Calls to native kernel functions and API functions are monitored.
- Function call injection, they allow an alternation of the execution on runtime, or inject own code in the running process, this technique has been described in Section 2.2.

Dynamic Analysis Techniques

Using dynamic analysis techniques TTAnalyze is able to analyse the code on run time. As the malware gets executed and only commands that really get executed are monitored things like obfuscation or code encryption play absolutely no role. In order to be efficient and safe, the malware gets analyzed on a virtual machine that is not connected to the network, in order to prevent it from spreading. Virtual machine have also the advantage that you can easily erase them and set them up, in opposition to a real PC where this would last a while. TTAnalyze uses QEMU [Bel05] as a virtualisation software, which brings us to the architecture part of this section.

Architecture

The architecture of TTAnalyze has been designed in a way to be easily able to distinguish between malicious and normal instructions of the processes. As TTAnalyze runs on an emulated PC with a normal operating system, the framework has to be able to track exactly what commands have been called by the malware and which are usual commands called by normal processes. The technique used by the TTAnalyze team relies on the CR3 register of the processor. This register, which is also known as page-directory base register, contains the physical address of the base of the page directory for the executed process. Knowing that the virtual address space is computed from the physical page directory it seems obvious that TTAnalyze can find out when the malware is executed: It simply has to monitor the CR3 registry after each context switch, if the CR3 registry points to the physical address space of the malware, we can be sure that the currently executed instructions have a relation to the malware. Now this is for the concept, the realisation of this is a little bit harder, as you have to find out the physical address space of the malware, start the malware in suspend mode and somehow communicate the address to the TTAnalyze framework. This will not be covered in this paper, but closer information can be found in [Kir06].

Another technique used by TTAnalyze on order to monitor the access to operating system services in an inconspicuous way is to compare the value of the instruction pointer of the processor running inside the virtual machine with the start address of the known operating system functions. TTAnalyze uses a callback function to realize that. Windows application access operating system functions through dynamic linking, and calling their export functions. We can now access the dll's export table in order to gain back the start address of the function that is called (CreateFile for example is located in kernel32.dll)

Now that TTAnalyze is able to retrieve operating system function calls it would be nice to know what parameters have been passed to the functions. This has also been realised in TTAnalyze, the technique used to make this possible relies on callback functions. Assume an analyzed malware sample tries to create a file by invoking the CreateFile function, TTAnalyze is then able to use a callback function that accesses and logs the parameter given to the function, and through this it is able to know the name of the created file. Details on this techniques are described in [Kir06].

Another concept that is extensively used in TTAnalyze is the, already presented in Section 2.2.2, code injection. This is used to change the flow of execution of the malware. Using this technique the malware can be forced to perform different actions that allow a more accurate analysis of the sample. For example there is no possibility to know if CreateFile is really creating a file or just reading an existent. The real behaviour of the call can be detected by a smart injection of code. Similar problem is the fact that it is hard to detect if the malware works on a folder or a file, or if TTAnalyze is confronted with unknown handlers.

All the techniques described here are implementable in the testing environment, as a virtualized CPU is used. Simply inserting additional instructions into QEMU's translation blocks does the trick.

Analysis Report

The analysis report generated by a tool like TTAnalyze is a very important part of the tool, as it provides a lot of information to the user or researcher and allows him to gain an overview over the analyzed malware. It is crucial to have good and reliable reports if you want to generate signatures for IDS or similar intrusion detection technologies. Therefore we'll present the analysis report provided by TTAnalyze.

The report is generated by a set of callback routines that log security related actions linked to the execution of the malware in the emulated environment. The report generated by TTAnalyze contains the following information:

- General information about the malware sample but also about the invocation of the tool, including command line parameters.
- File activity. Monitors the activity on files, creation, modification, deletion, etc.
- Activity performed by the registry: Monitoring of the Windows registry.
- Service activity: What services have been touched, stopped or started.
- Activity by the processes: Monitoring of the process activity.
- Network activity: Contains a dump of the packets send and received by the malware.

This provides a lot of information about a malware sample that has been executed in the environment, it provides us with a lot of information regarding the activity of the malware.

2.3.4 CWSandbox

Finally, we will introduce CWSandbox, a framework known for:

- executing malware in a simulated environment,

- monitoring all system calls,
- automatically generating detailed reports.

Architecture

The architecture of CWSandbox has been conceived in a smart way, it allows a good balance between different known techniques and generates a detailed report of the analyzed malware samples.

First of all we will describe the setup used: CWSandbox executes a malware in order to analyse it, it is a behaviour based malware analysis. The environment used to analyse and execute the malware is a controlled environment which is not virtualized. The security related actions performed by the malware are monitored using well known techniques described earlier in this paper. CWSandbox makes extensive use of network traffic data to get a more accurate overview over the behaviour of the malware.

To be able to use CWSandbox different task have to be performed before starting. cwsandbox.exe as well ass cwmonitor.dll have to be integrated in the virtualized environment in order to be able to analyse the malware. API hooks are installed by the .ddl file as well as exchanging information with the sandbox itself. Through API hooking a lot of information can be gained over the malware. In the best case the malware will not notice anything about the modified environment it is running on, and behave normally. In order to increase the invisibility of the framework cwmonitor.dll comes with some rootkit functionalities. While a malware gets executed a huge amount of communication happens between cwmonitor.dll and cwsandbox.exe, this is realized using IPC. Through this the sandbox is able to log a huge amount of information (API calls) provided by cwsandbox.dll. A high-level overview will be bresentet in *Figure 3*.

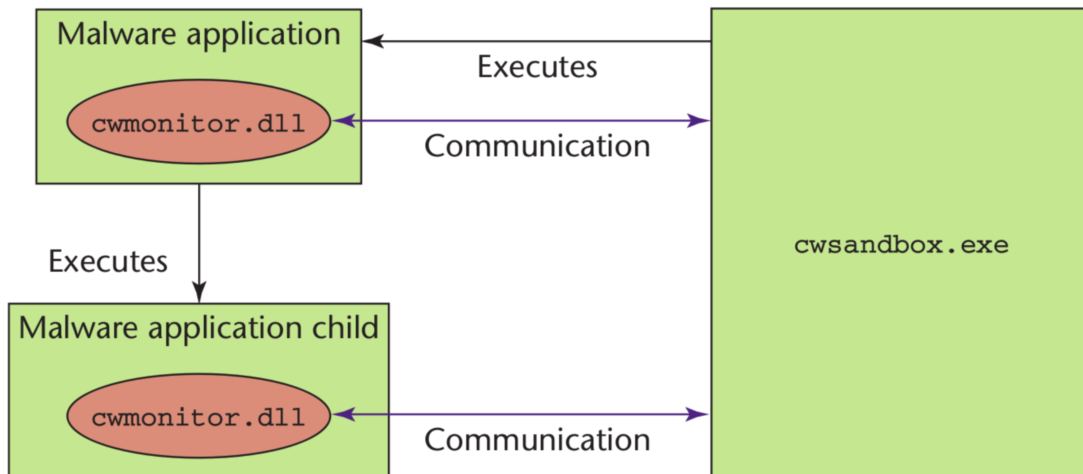

Figure 3: Overview of cwsandbox. [HF07]

At the end, once the execution is ended by the malware itself or the sandbox, the data collected is analyzed and an XML report is generated. In the next part details of the report will be described.

Analysis Report

The analysis report generated by CWSandbox is very complete, as CWSandbox performs its analysis on live systems it knows exactly what interactions with systems, network, registry or files is performed. CWSandbox is quite accurate in its analysis but of course it is only able to monitors the processes that are hooked, so minor changes might not be detected by the framework, if there has no hooking been created for those functions. Running the malware for a period of 2 minutes seems to be the best to gain accurate results.

The XML report contains a lot of details which will not be described in detail here but can be found in [HF07]. Reports include, beside other information, what protocol was accessed (IRC, HTTP,...) passwords for channels, ... what the malware tried to do, click fraud for example, or how it tried to spread. Bodies as well as destination email addresses were logged in case SMTP was used to spread.

All in all, the report should help the analyst to classify the analyzed malware in an fast, automated and reliable way. Further analysis can then be performed by the researcher on interesting or new threats. This allows a fast classification and the researcher can mask the background noise produced by the huge amount of already known malware. Through this the analyst wins huge amounts of time that he can invest in the analysis of the new malware.

Scan Summary	All Processes	File Activity	Registry Activity	Network Activity	Process Details

Technical Details By Process

Process # 1, (ID: 1048).

- File Name: **C:\68249510.exe**
- File Size: **604497 Bytes.**
- MD5: **269b4e8decff9048d5b00c9243f6f972**
- SHA1: **2cb5e58bfe0471b72af63e540ed6e57bab46eb80**
- User Name: **Dave**
- Start Time: **00:00.047**
- Termination Time: **01:00.110**
- Start Reason: **AnalysisTarget**
- Termination Reason: **Timeout**
- Execution Status: **OK**
- Application Type: **Win32Application**
- DLL Section...
- File System Changes...

OPEN FILE:

- File: \\.\PIPE\lsarpc
- File Type: namedpipe
- Creation/Distribution: OPEN_EXISTING
- Desired Access: FILE_ANY_ACCESS
- Share Access: FILE_SHARE_READ FILE_SHARE_WRITE
- Flags: SECURITY_ANONYMOUS
- Quantity: 2

- File: C:\68249510.exe
- File Type: file
- Source File Hash: 90E110C35116995E390809F2D8FBF500F34DF62B
- Creation/Distribution: OPEN_EXISTING
- Desired Access: FILE_ANY_ACCESS
- Share Access: FILE_SHARE_READ
- Flags: FILE_ATTRIBUTE_NORMAL SECURITY_ANONYMOUS
- Quantity: 2

FIND FILE:

- File: C:\DOCUME~1\Dave\LOCALS~1\Temp\nsa5.tmp
- File Type: file
- Source File Hash: hash_error
- Desired Access: FILE_ANY_ACCESS
- Flags: SECURITY_ANONYMOUS

Figure 4: Extract of an analyse report. [cw]

3 Conclusion

3.1 Security Application

When confronted with new malware every day, researchers have to be smart and find ways to analyse and protect infrastructures against unknown threats. This section will introduce the reader to some of those security applications that help protect against unknown exploits. The details of each techniques is way to wide to be explained in detail here, so we just touch on the subjects.

Dynamic taint analysis is used to detect and prevent exploitations of previously unknown attacks or exploits. A framework called *Sting* is able to produce filters to protect against those attacks in an automated way. This should protect hosts and networks against attacks of fast spreading worms, like, in the past for example CodeRed.

Input-based filters are very important. They prevent hosts from being compromised before they can be patched. It is self explaining that a good, reliable input-based filter would help best if it was working fully automated. The BitBlaze project team developed a tool able to do exactly this. Their tool is even able to do this with zero false positives. This kind of protection helps against several kind of exploits.

Another threat is the patch of the vulnerability itself. Attackers are able to use the information in the patch to produce exploits and attack hosts that are not yet patched. Researcher demonstrated ways to write patched in a secure way, these patches are not revealing any sensitive information an attacker could use to exploit vulnerable hosts.

3.2 Malware Analysis and Defense

Tools developed in the goal to protect users and retrieve information related to the malware are widely spread in the research field, we will introduce some of them here:

Panorama is a tool that is able to detect privacy-breaching malware by using whole-system dynamic taint analysis. This techniques allows researchers to detect overwrite attacks. This includes the techniques used by most of the exploits. As dynamic taint analysis works on software without the need of its source code nor special compilation flags basically every software could use this technique. [PS08]
Renovo is a tool designed and developed to have a complete dynamic approach of extraction of hidden code. It claims to be able to handle novel packers and to be resistant to various evasion

techniques. This tool can be tested and used by everyone as it is publicly reachable through the Internet. [PS08]
Hookfinder is a tool that is able to detect hooks placed by malware in an automated way. It should also be able to identify and analyse new hooking techniques. These is very important information for anti-virus companies and researchers as this allows them the react to new hooking trends and to protect users. [PS08]
BitScope is an automated malware dissector. It is uncovering hidden functionality of malicious binaries. Extracting hidden functionalities of malware is a very important task. Nowadays malware is getting smarter and smarter, developers know about the techniques used by malware fighters and they try to hide functionalities of their code from those researchers. [PS08]
Since some time the presented tools are available for everybody. The tools can be accessed over the Internet and samples of malware cane be uploaded to the platform, the platform is generating reports and provides it to the user. Different parts of the BitBlaze project are open-source.

3.3 Personal Conclusion

The following section will reflect the author's personal point of view regarding dynamic malware analysis.

The turnover from manual malware analysis to dynamic malware analysis seems to be a need for the future. Huge amounts of malware are discovered every day. This can be very low skilled kit-based malware distributed in mass by script kiddies, or malware build by professionals following one goal, earning money. Cyberwar has started to be a real threat to everyone, and the proportions it takes are increasing with every year. The need to have a method to identify malware in a fast and reliable way, and generate signature faster than nowadays seems to be a real need.

A lot of progress has been performed in the last years in the domain of automated malware analysis. Many new ideas have emerged, but also new tactics by the other side, to evade or exploit the new tools of the good guys. Many attacks on virtual machine have been noticed, intelligent malware has been released, noticing whether it is running on an emulated or native machine. It is like it has always been in the past, the cat and mouse game is still going on, and should not end here. As hacking starts to be more and more mainstream, it is sure that the amount of good and evil guys will increase and the battle will go on. Nevertheless, it seems to be an equal war, both camps have very high skilled engineers and researchers, so they are fighting with equal arms.

It is with much suspense for the future, that we will see if the approach of dynamic malware analysis will give a significant advantage to the good guys, or if the bad guys will win this battle, or give it up and turn to other methods to reach their goal.

Bibliography

[Bel05] Fabrice Bellard. Qemu, a fast and portable dynamic translator. 2005.

[cw] `http://www.sunbeltsecurity.com/partnerresources/cwsandbox`.

[Fer07] Peter Ferrie. Attacks on more virtual machine emulators. 2007.

[GR03] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. 2003.

[HF07] Carsten Willems Thorsten Holz and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. 2007.

[Kir06] Ulrich Bayer Andreas Moser Christopher Kruegel Engin Kirda. Dynamic analysis of malicious code. 2006.

[KK07] Andreas Moser Christopher Kruegel and Engin Kirda. Limits of static analysis for malware detection. 2007.

[PS08] Dawn Song David Brumley Heng Yin Juan Caballero Ivan Jager Min Gyung Kang Zhenkai Liang James Newsome Pongsin Poosankam and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. 2008.

[Sym10] Symantec. Internet security threat report: Volume xv: April 2010. 2010.

[WS07] Alexandre Dulaunoy Gerard Wagener and Radu State. Automated malware analysis. 2007.