Malware Forensics: Discovery of the Intent of Deception

Murray Brand, Craig Valli and Andrew Woodward
secau - Security Research Centre
Edith Cowan University
Perth, Western Australia
m.brand@ecu.edu.au, c.valli@ecu.edu.au, a.woodward@ecu.edu.au

ABSTRACT

Malicious software (malware) has a wide variety of analysis avoidance techniques that it can employ to hinder forensic analysis. Although legitimate software can incorporate the same analysis avoidance techniques to provide a measure of protection against reverse engineering and to protect intellectual property, malware invariably makes much greater use of such techniques to make detailed analysis labour intensive and very time consuming. Analysis avoidance techniques are so heavily used by malware that the detection of the use of analysis avoidance techniques could be a very good indicator of the presence of malicious intent. However, there is a tendency for analysis tools to focus on hiding the presence of the tool itself from being detected by the malware, and not on recording the detection and recording of analysis avoidance techniques. In addition, the coverage of anti-anti-analysis techniques in common tools and plugins is much less than the number of analysis avoidance techniques that exist. The purpose of this paper is to suggest that the discovery of the intent of deception may be a very good indicator of an underlying malicious objective of the software under investigation.

Keywords: Malware, anti-forensics, anti-analysis, digital forensics, cyber crime

1. INTRODUCTION

Modern malicious software (malware) employs stealth and deception techniques in an attempt to remain undetected on computer systems and difficult to fully analyse (Harbour 2007). Legitimate software can employ anti-analysis techniques to hinder reverse engineering attempts and to protect intellectual property (IP). However, software with a malicious intent may be considered to be far more likely to employ anti-analysis techniques than legitimate software (Vuksan, Peričin et al. 2009), to the extent that, detection of the presence of anti-analysis techniques may indicate the presence of malware (Wysopal 2009).

Initial static analysis of code may quickly reveal the presence of obfuscation or other high level anti-static analysis techniques. However, malware can be so heavily obfuscated that static analysis alone may reveal very little information of benefit to the digital forensic analyst. Equally, malware that is executing has every opportunity to examine the environment it is running in and alter its behaviour if it detects that dynamic analysis is being conducted. In such cases, it may use deception to hide its true functionality and intent.

Executing code exhibits complex dynamic behaviour, making the detection of anti-dynamic analysis techniques more difficult. This is because numerous paths of execution are available, any of which are generally preceded by conditional branch logic that alters the path of execution based on the results of preceding code that may include analysis detection routines. If the preceding code to the branch contains code used to detect the analysis environment, the alternate path to deceptive code could very well lead to the path of execution where the true intent of the code is performed. Such a path could reveal the true intent of the malware. In addition, an execution path of deception may include any number of destructive acts that hinder the collection of evidence. The detection of the use of anti-analysis techniques could assist the digital forensic investigator to reveal intent.

2. ANALYSIS AVOIDANCE

An extensive array of techniques is available to programmers to hinder attempts at reverse engineering their code in an endeavour to protect intellectual property (Falliere 2007; Yason 2007; Ferrie 2008). These very same techniques are employed by malicious software (malware) to hinder digital forensic analysis. Malware analysis is core business to the anti-virus (AV) industry, who work to extract signatures from the malware for the purpose of threat detection, and to formulate eradication strategies. However, the resultant recognition rules and signatures may provide very little benefit to the forensic analyst. This is because the rules of recognition are generally dependent upon a malware analyst having already extracted suitable recognition rules, whether the rules are signature and/or heuristics based. Anti-virus software may be of very little benefit to assist the forensic investigator in the identification of the intent of malware. The detection performance of AV software has been shown by a number of researchers to be far less than ideal (Rutkowska 2006; Yin, Song et al. 2007; Yan, Zhang et al. 2008; Zhou and Meador Inge 2008). If the malware has not been analysed before, then it is highly unlikely that rules of recognition exist. This is especially true of malicious, customized malware that is targeting individuals or specific organisations that is not circulating via the internet. In such cases, the digital forensic analyst may be required to analyse the malware in detail. Although online analysis engines exist, it may not be prudent to submit malicious code to such sites, particularly when confidentiality must be maintained. This is because online analysis engines typically share submitted samples with AV companies.

Online analysis engines are available that can provide very useful reports such as detected virus signatures, file activities, registry activities, process activities, service activities and network activities. Online analysis engines such as Anubis

(International Secure Systems Lab, Vienna University of Technology et al. 2008) have limitations (Bayer 2009), such as the virtual machines in which the dynamic analysis is being conducted being detected (Innes and Valli 2006; Smith and Quist 2006). This is because virtual machines do not perfectly emulate the operating system, and attackers need only determine small differences in the environment to detect the presence of a virtual machine. Understandably, online analysis engines incorporate a time out period to limit the time allocated to analysis. Malware need only wait a certain period greater than the time out period to hinder detection by such engines. Since the engines are automated, a simple check which malware can use to detect it is running inside known engines is to read the environment and compare the results against known baselines of common analysis engines. Once the analysis environment has been detected, the malware can branch to deception code, or choose not to run at all. Another limitation is that only a single path of execution is executed and alternate, significant paths of execution that show the intent of the malware may be missed entirely (Bayer 2009). This is not to say that online analysis engines cannot provide fruitful results, but the digital forensic analyst needs to be aware of the limitations of the use of such tools. Commercial analysis engines are available that may provide the analyst more control over the parameters of analysis than the online variety. A series of articles by Hudak (2009; Hudak 2009) provides an introduction to automating malware analysis that can be further customized and extended by using additional tools and scripts. Advantages of establishing a customized analysis service for forensic investigation is that the use of analysis avoidance detection scripts that log deception events can be developed.

Debuggers such as OllyDbg (Yuschuk 2008) and IDA Pro (Hex-Rays 2008) are commonly used for the analysis of malware. Plugins such as Olly Advanced (MaRKuS 2006) for OllyDbg and IDA Stealth (Newger 2008) for IDA Pro focus on hiding the presence of the tool from the software under investigation, in an effort to avoid detection. Unfortunately, the number of anti analysis techniques covered by the plugins is far less than the number of analysis avoidance techniques that are available. In addition, the plugins do not generally log the detection of analysis avoidance techniques.

3. SPECIFIC DECEPTION TECHNIQUES

The following subsections outline a simple taxonomy of deception techniques, together with references on how they can be detected and mitigated. It is important to note that these techniques can be employed in various combinations and will not likely be encountered in isolation.

3.1 Anti-Emulation

Virtual machines offer many advantages to the forensic analyst such as the ability to quickly restore images to a known state. A range of techniques exist to detect that the malware is running inside a virtual machine such as VMWare or Virtual PC (Innes and Valli 2006; Smith and Quist 2006). The use of these techniques can be detected and mitigated (Eagle 2004). In critical cases, or where the use of anti-emulation techniques cannot be mitigated due to various constraints, it may be prudent to use a real machine rather than a virtual machine.

3.2 Anti-Online Analysis

A variety of online analysis engines are available to the forensic analyst such as Anubis and Norman Sandbox. An advantage of these engines is that they can give very good reports on the functionality of the malware. However, there are limitations that the analyst needs to be aware of. The analysis engine may not match the desired target environment of the malware, triggering behaviour may be missed, only single paths of execution are typically examined, analysis is time limited and a variety of methods are available to detect the analysis environment (Bayer 2009). Due to confidentiality concerns, it may also not be wise to submit malware where there is no guarantee that the malware will not be shared with a variety of researchers and AV companies. Commercial products of some of these engines are also available that can be purchased and configured in the analysts premises such that malware need not leave the laboratory (Norman 2009). However, the focus of the engine may be limited to the reporting of behaviour and not necessarily on the detection of deception. Tailored, automated analysis environments can be established (Hudak 2009; Hudak 2009). Such an environment could be used to include specialised environments that closely resemble the original targeted environment of the malware together with the same operating system version, service patch level, installed software, peripheral equipment, attached subsystems, emulated or real loads and services. An example of such a tailored environment could be an emulation of a Critical Infrastructure system.

3.3 Anti-Hardware

Malware can use a variety of techniques to detect if it is being analysed by taking advantage of the way hardware such as the CPU and registers are used during debugging sessions. This can include techniques to exploit the way the prefetch queue works when software is being debugged and by execution timing. The use of these techniques can be detected by using appropriate techniques (Ferrie 2008).

3.4 Anti-Debugger

A plethora of opportunities exist to malware to determine if it is running in a debugger and to employ deception if it detects that it is being analysed. These techniques target the way debuggers work and use this to take control of the flow of execution of the code. A number of structures are associated with any loaded program. Bits set in these structures can be examined by running malcode to determine if it is being debugged (Falliere 2007; Yason 2007; Ferrie 2008). Debuggers can be scripted to detect the use of these techniques and mitigate them as the programs are running (Eagle 2008; Seitz 2009). Scripts to perform such tasks can also be found on a number of reverse engineering websites for a variety of debuggers in a number of different scripting languages. Techniques employed in one language can be converted and extended to the analyst's language of choice. It is advisable to learn a number of scripting languages to take advantage of this situation.

3.5 Anti-Disassemblers

These techniques target the way disassemblers work to produce false disassemblies. Two methods used by disassemblers are linear sweep and recursive traversal. Linear sweep is used by the disassemblers/debuggers SoftIce (Compuware 2008) and WinDbg (Microsoft 2008), which conduct disassembly in a sequential manner. Recursive traversal in contrast (used by OllyDbg and IDA Pro), follows the flow of each branch and is more tolerant to anti-disassembly tricks. Linear sweeps can be easily confused with junk bytes, but the recursive sweep technique can also be fooled with opaque predicates (Eilam 2005). Opaque predicates are simply code that appears to make a decision that could alter program flow. But in reality, only one branch of execution is possible to follow. Detection of the use of this technique can be assisted by comparing the results of disassemblies from various disassemblers. If the results vary considerably, an instance of deception may have been discovered. The point of implementation is likely to be at the point of divergence between similarities testing of resultant disassemblies.

3.6 Anti-Tools

Tools used by the analyst can be detected by the running malware. If discovered, the malware may enter a deceptive mode. Malware may even use weaknesses or vulnerabilities of the tools against themselves to give the malware another opportunity to employ deception. Packers and protectors can include options to detect popular tools such as OllyDbg and IDA Pro as well as the popular online analysis engines (Bayer 2009). Specific techniques can be uncovered by scripting to search for the use of the techniques or can be discovered by comparing results from the use of different tools.

3.7 Anti-Memory

Dumping of memory can be useful for the malware analyst after letting obfuscated programs unpack themselves. This is achieved by catching and halting the program at the moment the unpacking stops, and then dumping the unpacked program from memory. This allows the code to then be analysed. Packers can make this dumping process less useful by deleting a section of code as soon as it has finished executing. This technique is known as “stolen bytes”. These bytes must be restored if the dumped program is to be run again.

3.8 Anti-Process

Techniques can be used to target the way processes are handled whilst being debugged. An example of this is exploiting the way Thread Local Storage (TLS) is used. This technique is used to change the original entry point of a program to a different entry point so that an initial check can be made to see if a debugger or other analysis tools are being run. It changes the Portable Executable (PE) loader so that the entry point of the program is referenced in Thread Local Storage (TLS), which is the 10th directory entry in the optional PE header (Falliere 2007). TLS callbacks can be identified by examining the Data Directory of the PE header using a tool such as pedump (Pietrek n.d.) because it will show if a TLS directory is in the executable and these can be examined for malicious intent (Yason 2007)

3.9 Anti-Analysis

Various techniques can be used to target the way analysis is conducted. Deceptive practices include transformations of the original code to make the resultant code harder to read. Transformation characteristics include potency, which is the level of complexity added to the code and can be measured by complexity metrics. This includes measuring the depth of nesting in a particular sequence and the number of predicates the code contains. Another characteristic is that the transformation must be resilient. A highly resilient transformation is hard to undo. Deobfuscators can conduct data-flow analysis to reverse the transformation. There is also a cost characteristic of the obfuscation transformation in terms of increased size of the resultant code and slower execution time (Eilam 2005).

3.10 Packers and Protectors

Packers and protectors make static analysis of software difficult because the actual code instructions and data is not able to be read until the code has been unpacked. It is very similar to compression. Unpackers exist for many packers in the form of scripts and plugins for debuggers. Packed programs can also be unpacked manually by using a debugger. The unpacked code can then be analysed with a debugger such as IDA Pro, or Ollydbg. If malware to be analysed has been packed by an unknown packer, it can often be loaded into memory, and then process dumped using LordPE (yoda 2005) or any other memory dumping tool. It should be noted that the code may use techniques to determine if a debugger is being used and respond by protecting itself using some combination of the anti forensic techniques that have been discussed in this paper. The analyst needs to be in a position to statically analyse the executable as soon as it has unpacked itself, by starting analysis at the Original Entry Point (OEP), otherwise code can be written over and evidence overlooked. Packer signature detectors compare the Entry Point of the code against known signatures to try to determine the type of packer used. However, deception may be employed by incorporating a false signature, and/or by implementing the real unpacker much further into the code than the Entry Point. Packers may even use multiple levels of packing. Measurement of entropy is a very good indicator of packing (Mandiant 2007).

3.11 Root Kits

Windows uses four privilege levels, known as rings, to determine the access level for access control. Access control determines how hardware can be accessed, what instructions a process may use, what files may be modified and which areas of memory can be accessed or changed. Ring 0 is the most privileged level and Ring 3 has the least amount of privilege. Most applications users run are run in Ring 3 and these applications cannot access hardware directly and have limited access to memory. Ring 3 is often referred to as "user land". Ring 0 applications run with full system privileges and can perform Input/Output (I/O), memory management, run device drivers, execute privileged instructions, access all memory space, access all hardware and access all components of the kernel. This is often referred to as "kernel land". A special mechanism exists so that a user land program can access kernel land in a controlled fashion so that device drivers (*.sys file) can be installed. Root kits exploit this mechanism so that they can install their own device driver into kernel land, giving their program full privileges at Ring 0 and hence control the environment in which other software runs. In this way, it can avoid detection (Hoglund and Butler 2005).

4. RECOMMENDED ANALYSIS METHODOLOGY

An iterative and recursive methodology that alternately uses static and dynamic analysis techniques to discover and mitigate anti forensic techniques has been proposed from the conduct of this research. Essentially it extends a malware analysis methodology discussed by Zeltser (2007) in which static and dynamic analysis phases are interspersed with the moulding of the analysis environment such that the behaviour of the malware can be determined. The proposed spiral analysis methodology, depicted in *Figure **1***, extends Zelter's methodology by adding focus to the detection and mitigation of anti-analysis techniques. This is because the use of such techniques could be considered to be precursors of intent to employ deception in the code and assists in the detection of the use of deception. It also allows the analyst to zero in on regions of interest more quickly. Another view of this methodology is presented in *Figure 2* in the form of a process diagram that could be implemented in software. It shows malware under investigation as the input to the process that employs the spiral analysis methodology and that a manual, semi-automated or automated analysis control supervisor as central to the analysis, where recording, processing and reporting is managed. The supervisor function interacts with each phase by providing control over the constituent steps in each phase. It also acts as the recipient of data which is produced by each phase which is required to make decisions on how to tailor the subsequent phases.

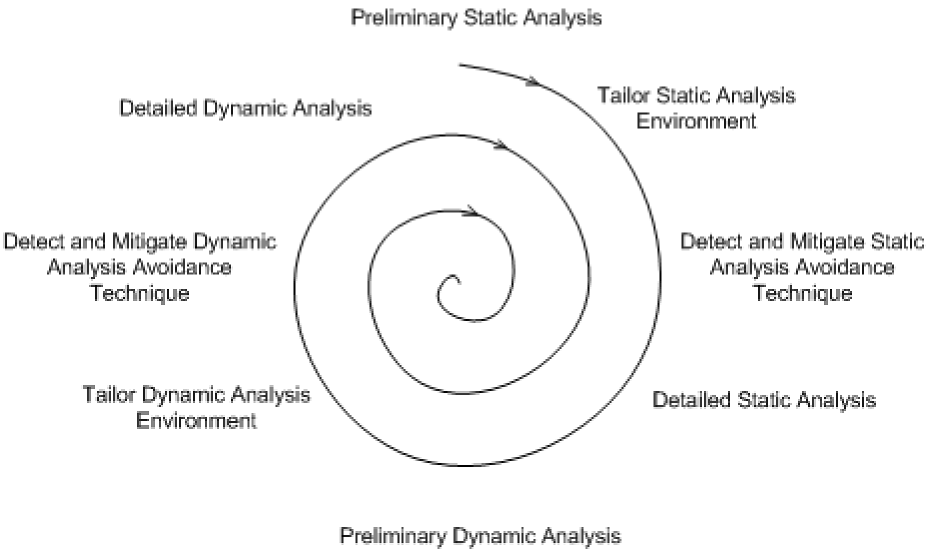
Preliminary Static Analysis
Tailor Static Analysis Environment
Detect and Mitigate Static Analysis Avoidance Technique
Detailed Static Analysis
Preliminary Dynamic Analysis
Tailor Dynamic Analysis Environment
Detect and Mitigate Dynamic Analysis Avoidance Technique
Detailed Dynamic Analysis

Figure 1 A proposed Spiral Analysis Methodology for determining whether a program may be malicious, based on obfuscation or other similar techniques

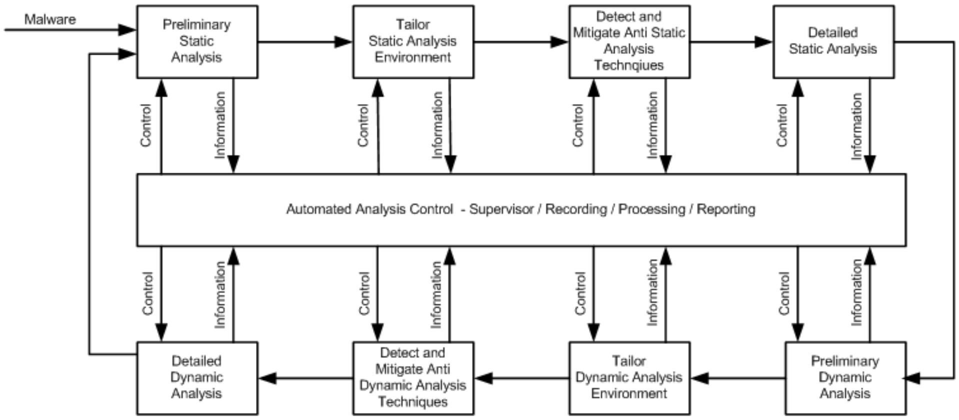
Malware
Preliminary Static Analysis
Tailor Static Analysis Environment
Detect and Mitigate Anti Static Analysis Technqiues
Detailed Static Analysis
Control
Information
Automated Analysis Control - Supervisor / Recording / Processing / Reporting
Detailed Dynamic Analysis
Detect and Mitigate Anti Dynamic Analysis Techniques
Tailor Dynamic Analysis Environment
Preliminary Dynamic Analysis

Figure 2 Process Diagram - Implementation of Spiral Analysis Methodology

5. CONCLUDING REMARKS

Malware is invariably heavily obfuscated and static analysis may provide little benefit if conducted in isolation from dynamic analysis. The benefits of dynamic analysis can also be subverted if the running code is treated as a 'black box' where only inputs and outputs are measured and internal inspection of the binary code is not performed. This is because the malware can detect that it is being analysed and can use deception techniques to deceive the analyst.

The discovery of deception in software under investigation may be a very good indicator of malicious intent. Although legitimate software can use the same techniques to protect intellectual property, malicious software uses anti analysis techniques so much, that detection of such techniques may prove to be a strong indicator of malicious intent. The digital forensic analyst may have to perform analysis of the software under investigation themselves because detection by AV software may provide less than ideal results. In addition, submission of suspicious files to online analysis engines may breach confidentiality agreements. This leads to the forensic analyst having to perform analysis of suspicious files in the lab. Although this can potentially be a very labour intensive activity, it can be assisted by the use of an appropriate technique such as the proposed spiral analysis methodology where anti-analysis techniques are detected, recorded and mitigated as analysis proceeds.

REFERENCES

Bayer, U. (2009). Anubis A platform the analysis of malicious code. Journal. Retrieved from http://www.ossir.org/paris/supports/2009/2009-06-09/ANUBIS-OSSIR-EN-June-2009-v1.1.00.pdf

Compuware. (2008). SoftIce.

Eagle, C. (2004). Honeynet Scan of the Month 32 Analysis. Retrieved October 19, 2007, from http://honeynet.org/scans/scan32/sols/1-Chris_Eagle/analysis.html

Eagle, C. (2008). The IDA Book: No Starch Press.

Eilam, E. (2005). Reversing : Secrets of Reverse Engineering. Indianapolis: Wiley Publishing, Inc.

Falliere, N. (2007). Windows Anti-Debug Reference. Retrieved October 1, 2007 from http://www.securityfocus.com/infocus/1893

Ferrie, P. (2008). Anti-Unpacker Tricks. Paper presented at the 2nd International Caro Workshop. from http://www.datasecurity-event.com/uploads/unpackers.pdf

Harbour, N. (2007). Stealth Secrets of the Malware Ninjas. Retrieved October 20, 2007 from https://www.blackhat.com/presentations/bh-usa-

07/Harbour/Presentation/bh-usa-07-harbour.pdf

Hex-Rays. (2008). IDA Pro.

Hoglund, G., & Butler, J. (2005). Rootkits: Subverting the Windows Kernel. Upper Saddle River, NJ: Addison Wesley Professional.

Hudak, T. (2009a, May 2009). Automating Malware Analysis. Hakin9, 3/2009 (22), pp. 50-57.

Hudak, T. (2009b, July 2009). Automating Malware Analysis. Hakin9, 4/2009 (23), pp. 64-69.

Innes, S., & Valli, C. (2006). Honeypots: How do you know when you are inside one? Paper presented at the 4th Australian Digital Forensics Conference, Edith Cowan University, Perth, Western Australia.

International Secure Systems Lab, Vienna University of Technology, Eurecom France, & UC Santa Barbara. (2008). Anubis: Analyzing Unknown Binaries. Retrieved October 4, 2008, from http://anubis.iseclab.org/

Mandiant. (2007). Red Curtain. Retrieved October 20, 2007, from http://www.mandiant.com/mrc

MaRKuS. (2006). Olly Advanced.

Microsoft. (2008). windbg.

Newger, J. (2008). IDA Stealth Plugin.

Norman. (2009). Norman Green Book on Analyzing Malware Executive Whitepaper 2009. Retrieved 07 Sept 2009, from http://download.norman.no/whitepapers/sb_executive_folder.pdf

Pietrek, M. (n.d.). PEdump.

Rutkowska, J. (2006). Introducing Stealth Malware Taxonomy. Retrieved April 12 2009 from http://www.invisiblethings.org/papers/malware-taxonomy.pdf

Seitz, J. (2009). Gray Hat Python. San Francisco: No Starch Press.

Smith, S., & Quist, D. (2006). Hacking Malware: Offense is the new Defense. Retrieved July 24, 2007 from http://www.offensivecomputing.net/dc14/valsmith__dquist_hacking_malware_us06.pdf

Vuksan, M., Peričin, T., & Milunovic, V. (2009). Fast & Furious Reverse Engineering with TitanEngine. Black Hat USA 2009, from http://www.reversinglabs.com/blackhat/TitanEngine_BlackHat-USA-09-Slides.pdf

Wysopal, C. (2009). Good Obfuscation, Bad Code. Retrieved May 03 2009, from http://www.securityfocus.com/columnists/498?ref=oc

Yan, W., Zhang, Z., & Ansari, N. (2008). Revealing Packed Malware. IEEE Security and Privacy 6 (5), 65-69.

Yason, M. (2007). The Art of Unpacking. Retrieved Feb 12, 2008 from https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf

Yin, H., Song, D., Egele, M., Kruegel, C., & Kirda, E. (2007). Panorama: capturing system-wide information flow for malware detection and analysis. Paper presented at the Proceedings of the 14th ACM conference on Computer and communications security.

yoda. (2005). LordPE.

Yuschuk, O. (2008). OllyDbg.

Zeltser, L. (2007). Reverse Engineering Malware: Tools and Techniques Hands-On. Bethesda: SANS Institute.

Zhou, Y., & Meador Inge, W. (2008). Malware detection using adaptive data compression. Paper presented at the Proceedings of the 1st ACM workshop on Workshop on AISec.