International Journal of Economics, Commerce and Management
United Kingdom ISSN 2348 0386 Vol. IX, Issue 1, Jan 2021

http://ijecm.co.uk/

PE MALWARE TRIAGING AND EFFICIENT FALSE-POSITIVE CLASSIFICATION

Ștefan Nicula
The Bucharest University of Economic Studies, Romania
niculastefan13@stud.ase.ro

Răzvan Daniel Zota
The Bucharest University of Economic Studies, Romania
zota@ase.ro

Abstract

The Portable Executable (PE) format represents the file format for executables used by Windows based systems. Many malware families are targeting this file format, the PE being at the core of Microsoft NT. As technology progresses, so are the malware samples and the techniques used by the attackers. This article presents a high overview of some of the methods, indicators and flows used to quickly triage or analyze PE malware. The focus is being targeted towards a faster triaging process and key takeaways for researchers that are dealing with false-positive and false-negative fixing. In each sample classification, time is of the essence. Fast methods that reveal important results should be prioritized first as any indicators of malicious activity or pattern could influence the verdict and help researchers to deal with samples more efficiently. Classifying false positives is always subjective based on the context. Depending on the engine that generates FPs, this job can become harder or easier. Understanding the root cause of false-positives and having knowledge of the systems is an important factor. The article strictly refers to processes and analysis techniques used for quickly triaging and classifying tricky samples and it does not focus on deep malware family reversing and threat analysis. The results indicated that an efficient PE malware triaging process is based on several factors including the file metadata, the

telemetry, and the events surrounding the sample at the execution time. Threat intelligence information will greatly help in the analysis process and will provide an efficient overview before starting the manual reverse engineering research.
Keywords: malware, reverse engineering, PE, malware analysis, false-positive

INTRODUCTION

Portable Executable (PE or PECOFF) file format is the Windows executable format. This includes .exe, .dll, .sys and others. It is one of the most widespread file formats in terms of malware samples, directly proportionate to the popularity of Windows hosts. A security researcher or a malware analyst that focuses on studying PE malware can often be presented with difficult samples that are on the edge of the spectrum. This is especially true for teams that are working directly with some form of engine detection, be it static, dynamic or a combination of both. In order to produce detection rules or modify them, some form of manual analysis is needed first, in order to have a research baseline. The false-positives (FP) and false-negatives (FN) are samples that are either clean files that are detected by the engine or malicious files that are not detected in any way. The engine is referred to the software that implements the detection rules and makes decisions based on the implemented factors.

The article tackles some of the edge cases in terms of false-positive identification and the triage process, as well as some of them most useful and result-oriented techniques in analyzing malware. Key aspects of a PE file analysis will be presented along with an overall of processes that goes through analyzing a high number of submissions. The article is not oriented on analyzing specific samples that are known to be clean or malware but rather the edge cases that have a high probability of being both.

The entire general manual analysis process can be viewed or abstracted as a percentage sum that is directly correlated with the sample classification. Supposedly, a researcher might start with the 100% assumption that a file is clean or malware depending on the context. If the analyzed file received is a false-positive, it could be natural that the researcher will expect the sample to be a clean one. A neutral position is also something to be taken in consideration when not a lot of context is presented from the submission. During the analysis process, each factor lowers and increases the assumption rate up until the deciding point which can directly classify a sample. For example, if a sample appears to be clean but at some point, a shellcode is decoded in memory and executed inside a new process, that's an indication that the sample analyzed has clear malicious intentions. Depending on the study, an in-depth research can be further conducted, or the sample can be categorized as malware and

the detection can be fixed accordingly. Regardless if the researcher uses initial assumptions or not, in the end, the sample classification is validated by reversing the sample and placing the right verdict. [1]

Among some of the factors that influence the decision making in the first initial stage, we can note particular ones such as lack of metadata, behavior of the GUI if any, user interaction, external dependencies or general execution flow. The classification should not be done solely on these factors alone however, they can greatly help in quickly determining the cause of the miss-classification or help in narrowing down the possibilities. Static analysis and dynamic analysis methods as well as threat research are approached from a practical point of view. [2]

ADVANCE TECHNIQUES FOR BYPASSES

Some malware samples are using software solutions for packing, obfuscation, anti-debugging and anti-analysis. These solutions can be very time-consuming to reverse and oftentimes, the time allocated to reverse such solutions that hide the malware code is much greater than the researcher's time dedicated for a specific sample. However, there are some workarounds for these specific solutions. [3]

One of the common ones is Themida, which is a complex software, very customizable, that uses virtualization in order to encapsulate the malicious code. If used correctly, the malware will execute on the host machine and will be protected entirely by the Themida VM. Even so, if the malware creator decides to use "unsafe" techniques with the malware sample, we can potentially benefit. For example, if the malware is leaving the packed binary and does a process injection, we can actually catch the malware while executing in another process and we can dump that process to obtain the malicious binary. Another example could be the case of a Dropper or a Downloader. If the analyzed sample is a dropper, it will save a payload and execute it on the machine. Most of the time, the Dropper's routine will also delete the dropped file however, we can intervene and stop the process right before the file removal occurs. And again, we obtained another stage of the malware chain which we can analyze and pinpoint the exact intents of the binary. [4]

Another interesting case to study is the DLL Preload issue that is affecting a certain legit executable. We can take the example of SysInternals tool suite which has a large variety of software. Some of them are exposed to DLL Preload. This is a type of vulnerability which is defined by the executable trying to load a certain library (DLL) from the root path of the directory. If the library is not verified in any way before loading and executing it, then we can provide malicious code as external functions in DLLs to be executed by our clean executables.

Oftentimes, they come packed so when a Dropper is downloading which looks to be valid software, then most probably it will use a DLL to inject its malicious code. [5]

WINDOWS API BREAKPOINTS

The importance of Windows API usage in reversing a sample is unmatched, one of the key factors in the analysis process is represented by finding and backtracking the Windows API imported by the executable. When analyzing a complicated sample, Windows API usage can act as checkpoints and guidelines inside the maze and are very helpful for a researcher that has access to them, providing that the sample does not use some form of obfuscation. Even if the sample is using dynamically resolved imports, a process dump at the right time and an IAT manual fix will reveal the needed information. However, in cases where the PE file contains inline functions combined with static linking, the process gets considerably harder. In these situations, tools and functionalities such as IDA FLIRT [6] are recommended, they are capable of identifying functions based on checksums and function hash signatures for a big number of libraries and compilers.

Key function breakpoints are very important when analyzing PE malware. We can use the x64 debugger as an example. We can also classify Windows API calls and expect some techniques depending on the malware's intentions. Some of the examples include:

CreateProcessInternalW - API call used for process injection, the Windows API gateway for high level create process calls;

CreateRemoteThread - often times used to patch certain binaries with DLLs, can be commonly found in cracks and keygens;

WriteProcessMemory - used in conjunction with process injection techniques;

We can use x64 debugger even before the modules are being loaded, either dynamic or static from imports. This procedure places breakpoints on unresolved API calls which helps in analyzing important API calls. A word of caution is the TLS PE file section which can tamper with the breakpoint routine. [7]

When using breakpoints on the aforementioned API calls, correlation with the stack parameters and the calling convention is imperative. When the breakpoints hit, we must check the MSDN documentation for that specific call. For example, CreateRemoteThread uses the following arguments:

```
HANDLE CreateRemoteThread(
  HANDLE                 hProcess,
  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
```

```
  SIZE_T                dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
  LPVOID                lpParameter,
  DWORD                 dwCreationFlags,
  LPDWORD               lpThreadId
);
```

By looking on the stack for the first argument, hProcess, we can actually see the process in which the injection is taking place. Dumping the process and reconstructing the IAT will reveal the code that's about to be injected in the new process.

BEHAVIOR ANALYSIS

Behavior analysis is sometimes the fastest way of identifying and classifying a PE sample. Depending on the protection mechanisms used by the sample, the behavior analysis can yield relevant results or none at all. The techniques that can be used in this area are very broad, covering aspects such as sandboxing, emulation, virtualization and hypervisor analysis. Some automated systems designed for behavior analysis are even acting as automated debuggers, analyzing and monitoring memory usage, API calls, file access and more. [8]

API Monitor is a tool that helps monitor all the Windows API calls made by an executable. We can note that this tool is based on dynamic analysis and runs the sample in order to make the observation, but it can also be attached to a running process. It's important to monitor the Windows API calls when dealing with a sample that's not very verbose in terms of output or GUI messages. [9]

Anti-debugging is one of the protection mechanisms usually employed by some malware samples that are trying to avoid analysis, it can range from multi-thread anti-debug timers, to mutexes, shared files and folders and many more. Some samples are using interesting anti-debug tricks like creating a mutex in order to prevent other external processes from attaching or debugging the main process. [10]

Some old and deprecated techniques of anti-debug are using PE flags to detect the debugging status of an executable, evasion debugger plugins or custom scripts can be used in order to bypass this. When starting a process under a debugger, multiple flags are being modified. For example, the debug flag present in the Process Environment Block (PEB) at offset 0x79. This value can be modified after loading the sample inside the debugger but before the sample starts executing user code.

The above action can be achieved, for example, using a WinDBG scripts:
r?$t0 = (ntdll!_peb *) @$peb;?? @$t0->BeingDebugged;eb (@$t0+2) 0;?? @$t0->BeingDebugged

There are many implementations of obfuscation techniques, anti-debugging, anti-analysis techniques, anti-sandboxing and VM, anti-tampering [11]. Most of the common ones are detailed on technical research papers authored by many researchers. When dealing with a PE sample, it all comes down to the code that's being analyzed. There are countless factors that can dictate the analysis process, being it dynamic or static. When dealing with non-APT, common samples identified in the wild, there is a high chance that the sample will use some form of the public documented techniques to do malicious activities or to avoid detection and analysis. However, more complex malware families and standalone samples will be very subjective and custom, meaning that usual techniques might be modified, or new and alternative techniques might be used by the malware author. In this case, a high probability of the attacker making more variants of the techniques will also increase. Threat intelligence and sample hunting will definitely be a key aspect to consider, and depending on the results, a new technique might be worth documenting if it is being used in a widespread malware family. It all comes down to the sample analyzed and the intel gathered for the PE.

One key aspect in the sample classification process is represented by threat intelligence as well. Sometimes, a standalone executable might look malicious or intrusive at first glance. Dealing with false positives often requires seeing the entire documented picture. In the end, it can all be referenced to the entire data collected that creates a full overview of the file. Sometimes, looking at a file standalone will not be enough to classify it accordingly. A good example would be the file infectors who are modifying specific files across the system. [12]

INITIAL TRIAGING PROCESS

Behavior analysis or a quick sample detonation can oftentimes be very handful. Dependencies and external libraries required by the samples are sometimes a strong indication of a possible false-positive. In this case, the likelihood of the file being malicious could drastically decrease or it could be very specific. It is also worth mentioning that cracks and keygens often rely on some form of external dependencies like the program that's being targeted. They are usually very easy to catch and most of the time they are self-explanatory in their artifact search such as specific game registry keys or file paths that contain a game name, DLLs or other metadata. There's also the risk, however, that a crack or keygen might come with some additional payloads that will turn the sample into a trojan or spyware. That’s where the analysis gets more

complicated and from samples analyzed, attackers are usually embedding this type of malware into Windows crack activators. [13]

When presented with a file for the triaging process, a quick disassembly tool with file parser capabilities such as BinaryNinja can provide a very useful start [14]. A lot of important information can be accessed from such tools. Quick wins like imports, strings, segments and entropy are always important. More advanced disassemblers do provide possibly more accurate details and can present extra functionalities but those come with a heavy loading and parsing process. When looking for an initial fast overview of the disassembly, a more lightweight tool can be more suited.

The entropy represents a huge indicator in terms of initial sample classification. Entropy calculation plugins are available for most popular disassemblers or standalone executables are publicly available. These tools show a graphical representation of the entropy for given memory segments. A high entropy area inside a data section could possibly mean an encrypted payload such as shellcodes or encrypted code paths, even an entire encrypted PE [15]. Cross-references to the start address offset at the beginning of the high entropy area is a common technique to identify potential usages. In known packers such as UPX we can note that the entropy of the files is really high across the majority of the PE file. That's because the packer leaves only a small code routine that decodes the full executable and passes the execution flow to it, afterwards.

We can make a high level of abstraction of the PE malware analysis process as a threat or malicious level defined in percentages which increases or decreases depending on the number of artifacts identified.

A great benefit from a malware analysis arsenal would represent a quick triage process for PE files designed for analysis and FP or FN fixing. Just briefly looking at the code, summing up the threat intel received or obtained from the file can oftentimes create a solid baseline for the research.

The filename should not be overlooked, a file that was received or submitted with a suspect name such as *financial_report_2020.pdf.exe* can immediately reveal intentions. Or a file submitted that has a history of many different and non-related names represents an important factor as well. If the researcher has access to this type of information, it can become a key aspect in the classification process.

For more advanced systems, an important consideration that could dictate the analysis process is the similarity of the analyzed file compared to other samples. Machine learning and artificial intelligence started to be highly used across the industry by multiple top antivirus vendors. An engine that incorporates machine learning for the decision making will always have

the advantage of comparing samples and better understanding the context of the file. For this specific functionality, information collection processes such as file metadata extraction, behavior analysis and context flag generation stand at the base of the entire system. [16][17]

CONCLUSION

When dealing with a high number of samples, efficiency should always be considered. Just like applications have low-hanging fruits in terms of obvious security vulnerabilities and issues, so are the malware samples. Some samples might use advanced anti-detection methods but fail in correctly applying them or expose key artifacts that compromise the anti-forensics methods implemented. Having a high overview of the sample before starting the manual analysis process and having as much threat intel information as possible in the beginning will greatly enhance a researcher's efforts in correctly and efficiently classifying files. This article is not an in-depth research and analysis process of specific techniques but rather an overview of techniques, flows and key considerations that help in processes such as initial quick triaging, false-positives and false-negative fixing. The scope for future research includes identifying new valuable telemetry data, PE metadata, and local sensor deciders capable of catching and reporting relevant details regarding the execution environment for a particular suspicious sample.

The machine learning applied to PE malware analysis plays an important role in comparing similar samples and bulk classifications. Important research represents the analysis of encrypted or custom obfuscated samples and how those samples affect the machine learning algorithms based on generic flags extracted from PE files.

REFERENCES

[1] Bruce Dang, Alexandre Gazet, Elias Bachaalany, Sébastien Josse, Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation, Wiley, 2014

[2] Michael Sikorski, Andrew Honig, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 2012

[3] Eldad Eilam, Reversing: Secrets of Reverse Engineering, Wiley, 2005

[4] Rodrigo Rubira Branco, Gabriel Negreira Barbosa, Pedro Drimel Neto, "Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and AntiVM Technologies", BlackHat US 2012, https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf

[5] Peter Ferrie, The "Ultimate"Anti-Debugging Reference, 5 April, 2011, https://anti-reversing.com/Downloads/Anti-Reversing/The_Ultimate_Anti-Reversing_Reference.pdf

[6] Hex-Rays, IDA F.L.I.R.T. Technology: In-Depth, https://www.hex-rays.com/products/ida/tech/flirt/in_depth/

[7] Y. Kawakoya, M. Iwamura and M. Itoh, "Memory behavior-based automatic malware unpacking in stealth debugging environment," 2010 5th International Conference on Malicious and Unwanted Software, Nancy, Lorraine, 2010, pp. 39-46, doi: 10.1109/MALWARE.2010.5665794.

[8] Y. Fukushima, A. Sakai, Y. Hori and K. Sakurai, "A behavior based malware detection scheme for avoiding false positive," 2010 6th IEEE Workshop on Secure Network Protocols, Kyoto, 2010, pp. 79-84, doi: 10.1109/NPSEC.2010.5634444.

[9] Ö. Aslan and R. Samet, "Investigation of Possibilities to Detect Malware Using Existing Tools," 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, 2017, pp. 1277-1284, doi: 10.1109/AICCSA.2017.24.

[10] K. Yoshizaki and T. Yamauchi, "Malware Detection Method Focusing on Anti-debugging Functions," 2014 Second International Symposium on Computing and Networking, Shizuoka, 2014, pp. 563-566, doi: 10.1109/CANDAR.2014.36.

[11] S. N. Alsagoff, "Malware self protection mechanism issues in conducting malware behaviour analysis in a virtual environment as compared to a real environment," 2010 International Symposium on Information Technology, Kuala Lumpur, 2010, pp. 1326-1331, doi: 10.1109/ITSIM.2010.5561600.

[12] S. Samtani, K. Chinn, C. Larson and H. Chen, "AZSecure Hacker Assets Portal: Cyber threat intelligence and malware analysis," 2016 IEEE Conference on Intelligence and Security Informatics (ISI), Tucson, AZ, 2016, pp. 19-24, doi: 10.1109/ISI.2016.7745437.

[13] Markus Kammerstetter, Christian Platzer, Gilbert Wondracek, "Vanity, cracks and malware: insights into the anti-copy protection ecosystem", Proceedings of the 2012 ACM conference on Computer and communications security, DOI 10.1145/2382196.2382282, 809–820

[14] Binary Ninja User Documentation, https://docs.binary.ninja/getting-started.html

[15] Alessandro Mantovani, Simone Aonzo, Xabier Ugarte-Pedrero, Alessio Merlo, "Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem", Feb 2020, Network and Distributed System Security Symposium (NDSS 2020), DOI 10.14722/ndss.2020.23xxx

[16] I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, Jakarta, 2010, pp. 201-203, doi: 10.1109/ACT.2010.33.

[17] NightVision – Using Machine Learning to Defeat Malware, An Avira White Paper, 2017, https://assets.prod.cms.avira.com/cache-buster-1598423379/assets/oem.avira.com/resources/to%20delete/whitepaper_NightVision_EN_20170704.pdf