Runtime API Signature for Fileless Malware Detection

Radah Tarek(✉), Saadi Chaimae, and Chaoui Habiba

System Engineering Laboratory, ADSI Team,
National School of Applied Sciences, Ibn Tofail University, Kenitra, Morocco
tarekradah@gmail.com, chaimaesaadi900@gmail.com,
habiba.chaoui@uit.ac.ma, mejhed90@gmail.com

Abstract. Nowadays, cybercriminals become sophisticated and conducting advanced malware attacks on critical infrastructures, both, in the private and public sector. Therefore, it's important to detect, respond and mitigate such threat to digital protection the cyber world. They leverage advanced malware techniques to bypass anti-virus software and being stealth while conducting malicious tasks. One of those techniques is called file-less malware in which malware authors abuse legitimate windows binaries to perform malicious tasks. Those binaries are called Living Off The Land Binaries (LOLBINS). That being said, during the execution of the attack it is not used any malicious executable and, consequently, the antivirus is unable to identify and prevent such threats. This paper focuses on defining rules to monitor the binaries used by threat actors in order to identify malicious behaviors.

Keywords: Malware analysis · Fileless malwares · Malware detection

1 Introduction

In [1], authors defined a computer virus as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. They explained how it is theoretically difficult to detect and prevent virus infection. The same constraint is applied to malware in general. Malware is becoming the preferred toolkits for cyber-criminals, as it offers them a simple and effective way to infiltrate and damage computers without the user's consent. The term malware is a general term covering different kinds of threats to device safety, such as rootkits, RAT, botnet, etc.

Security researchers perform malware analysis to understand the behavior of malware and reveal their sophisticated techniques. Typically, the method by which malware is usually analyzed falls under two types: Static Malware analysis [2] and, Dynamic Malware analysis [3].

Traditional analysis and detection techniques are not suitable when dealing with advanced malware techniques. This paper aims to introduce a new way to recognize fileless and advanced malware by establishing custom rules for detection. Those rules are what we have called a dynamic signature.

This paper is organized as follows. Section 2 discusses the related works. Section 3 presents common malware detection and evasion techniques. In Sects. 4 and 5, paper describes in detail our proposed detector and the experimental results. Finally, in Sect. 6, we conclude with conclusion and future work.

2 Related Works

In this section, as related works, we present significant researches relevant to malware analysis and detection. As static analysis is not relevant to today's threats, many researchers have attempted to extract the dynamic information by executing files in a controlled environment also known as a sandbox. Authors in [3] proposed an API call sequence similarity to detect malware. Using API Hooking, they were able to trace API call sequences at runtime. Based upon this they were able to classify malware in different categories using API call sequences. This approach is useful to overcome obfuscation techniques used by malware. Another work [4] focused on machine learning detection, and showed how Adversarial Machine Learning used in malware detection can be under an attack. Using a classifier with the input of Windows Application Programming Interface (API), [5] presented a useful evasion attack model by considering different contributions of the features to the classification problem. To overcome this failure, they propose a secure-learning paradigm for malware detection, which present effectiveness against this kind of attacks. Regarding dynamic analysis evasion techniques, [6] proposed a hierarchical, classification of conventional Dynamic Analysis Evasion techniques. In the highest level, a Category of evasion means either the technique is for bypassing Automated or manual dynamic analysis process. For each category, several tactics exist. Tactics are the specific maneuvers or approach for evasion with the specified attitude of its parent category, for manual detection tactics are: Direct detection, Deductive detection, and debugger escaping. For the automated dynamic analysis evasion, tactics are either Detection-Dependent or Detection-Independent. Each tactic can be achieved using various techniques; techniques are the various practical implementations of those tactics. Author in [7] described how we could relate to various malware campaign based on similarities within binaries. They show through similarities that the developers of malware that hit several Middle East infrastructures have reutilized the same code of the old samples and the differences found in the malware binaries indicates that the developers have adapted their tools to evade systems. Authors in [8] were using YARA [9] rules in order to detect the four most relevant ransomware categories [10, 11] and [12]. Authors in [13] presented a dynamic API call sequence-based detector using machine learning with a 3rd order Markov chain that combines iterative learning process with run-time monitoring; Their proposed system has shown a higher rate of accuracy. Finally, in [14], authors presented a malware analysis framework for dynamic and static analysis of malware sample-based behavior, using Cuckoo Sandbox [15] for API call extractions. However, all the previously mentioned research was in an attempt to present a generic detection technique to identify malicious software. That is why we see the need for a technique that will give defenders the ability to decide which behavior is malicious or not, allowing them to prevent fileless malware infection. The main novelty of our proposed method is to allow malware researchers, implement their solutions to mitigate malware without having to write an antivirus, by using a dynamic signature. Malware analyst also can use it as an alternative to YARA rules as it allows them to overcome the use of packers and obfuscation techniques [16].

3 Common Malware Detection and Evasion Techniques

3.1 Detection Techniques

Signature-based detection is the most basic technique to detect malicious software [17], creators of AV software having previously identified and recorded information about viruses; as would a dictionary, the AV can detect and locate the presence of a virus. This dictionary is called the viral definition database, which contains the virus signatures. Before scrutinizing the static detection techniques, we will have a brief introduction to the PE file component [18]. Building upon this understanding, we can better understand and elaborate on the evasion techniques.

PE file: PE file or portable executable is the file format for object code, DLL (Dynamic link library) and others used in multiple versions of Windows operating system. The PE format represents a data structure that contains the necessary information for windows loader to handle the executable code, resolve sections, load the appropriate libraries, manages resources and various PE file component.

The static analysis relies on information extracted from the PE file in order to understand the program behavior and capabilities. Mainly that information is:

- Loaded DLLs: Dynamic link library is a kind of PE file in which several windows API function are stored and regrouped depending on the kind of operation functions can perform. For example, wininet.dll [19] provides functions API for network communication using the HTTP protocol.
- Imported Windows API function: From PE file we can extract windows API functions used by the program. Most Windows API functions are self-explanatory. For example, CreateProcess function under Kernel32.dll is used to create a new process; WriteFile is to write data to a file, etc. Windows API functions are well documented under the MSDN [19], but unfortunately, Microsoft does not provide documentation for some windows API, especially those related to NTDLL library (The lowest library in the user-space of operation system). Malware authors were observed using NTDLL functions in order to perform an obscure operation and to evade detection; however, a non-official and out of date documentation for NTDLL API exist on the internet [20].
- Extracted ASCII strings: ASCII strings can be extracted from a PE file, using various techniques [21], they are mainly located in the ".data" section of a PE file. Examining strings can help understand the program operations.

Resources: In a PE file, the resources section contain various data that will be used by the program at runtime, for example, icons, images, etc. In a malware analysis perspective, the entropy of those sections is taken into consideration [22]. For example, a PE file that contains a data section with higher entropy means that an encrypted, compressed or obfuscated data are embedded within the PE file.

3.2 Evasion Techniques

In this sub-section, we will present some bypass techniques used by malware authors making the traditional signature ineffective:

- String obfuscation: A malware author can hide the presence of ASCII string using simple XOR encryption. This is called string obfuscation. Even though there is some tools which can extract obfuscated string [23], those tools are ineffective with complex encryption schemes.
- API Calls obfuscation: In C and C++, rather than the static way in which a function can be called, functions can also be called dynamically at runtime [24]. This is done by first, loading the appropriate DLL in which the function is stored using LoadLibrary function, and then extracting a function handle using GetProcAddress. This operation is done using two API functions (GetProcAddress and LoadLibrary from kernel32.dll) which are legitimate and used by all windows programs. Another stealthier way is dynamically resolving GetProcAddress and LoadLibrary by parsing the PEB structure [25].
- A wide range of legitimate software uses resources sections; they contain images, and icons used by the program at runtime. Resources may also contain other PE file in case of an installer. That being said, it is not possible to rely on resources sections to detect a potential malware.

4 Contribution

To overcome the limitation of the static signature we will introduce the dynamic signature as a new way to represent a malware behavior. The dynamic signature represents the API call sequence of a given process including the past arguments. Using Microsoft detours [26], we can monitor a given process using API hooking [27].

Once the signature is obtained, we can compare it to other signatures stored in a database to decide either the monitored process is malicious or not.

Compared to the traditional signatures, dynamic signature detection cannot be bypassed using the previous evasion techniques. Dynamic signature is also effective against Fileless attacks [28] due to the use of legitimate software to accomplish malicious tasks.

4.1 Dynamic Behavior Extraction

The behavior of the analyzed software is retrieved, using API hooking. A technique that will allow us to intercept API function calls of a given process as described in Fig. 1. Giving as control over the way the process behaves within the network and the operating system. Once a process is created, this will trigger the monitor component of the solution. The monitor, in turn, will inject a DLL into the newly created process to perform hooking. The DLL is responsible for logging API calls and arguments and sends them back to the Behavior analyzer.

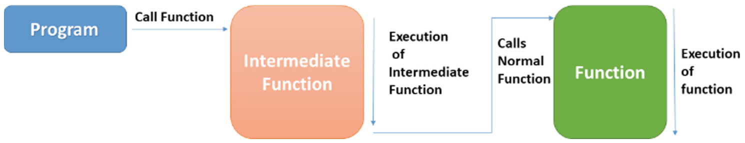


Fig. 1. Hooked function execution

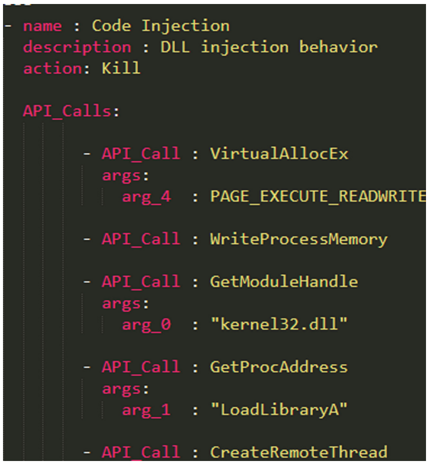
4.2 Matching

The extracted behavior from the previous section will be compared with already stored signatures. If any of the signatures match the extracted behavior, this will trigger the action mentioned in the signature.

API call sequence from a signature is converted into an integer sequence and presented as a Markov chain [29]. The Markov chain will have N possible state, where N is the number of API call sequence within a signature.

4.3 Signature Format

Signatures will be represented using YAML syntax [30], a commonly used human-readable data-serialization language for configuration files, but could be used in many applications where data is being stored or transmitted.



```
- name : Code Injection
  description : DLL injection behavior
  action: Kill

  API_Calls:

          - API_Call : VirtualAllocEx
            args:
              arg_4   : PAGE_EXECUTE_READWRITE

          - API_Call : WriteProcessMemory

          - API_Call : GetModuleHandle
            args:
              arg_0   : "kernel32.dll"

          - API_Call : GetProcAddress
            args:
              arg_1   : "LoadLibraryA"

          - API_Call : CreateRemoteThread
```

Fig. 2. DLL injection detection using a dynamic signature

As shown in Fig. 2, a given signature is represented with:

- **Signature name:** The name of the signature
- **Action:** The action to be performed if any process behavior match this signature
- **Process name:** (OPTIONAL) the name of the process.
- **Parent process:** (OPTIONAL) the name of the parent process

- **Function_1:** Monitored function
    1. **Arg1:** Function argument
    2. **Arg2:** Function argument
- **Function_2:**
    1. Arg1: Function argument
- **Function_3**
- **MACROS:** such as PID, PROC_NAME …

This allows as looking for particular behaviors rather than merely detecting a specific tool.

4.4 Architecture

Figure 3 represents a brief overview of the architecture of our detector. The monitor component is a Windows driver that operates at a kernel level and is responsible for injecting DLL into user-mode processes using APC [31]. The injected DLL will hook the interesting API calls and send results to the Behavior Analyzer using named pipes. Behavior analyzer will compare the received data with signatures stored in the database, and then perform the appropriate action.

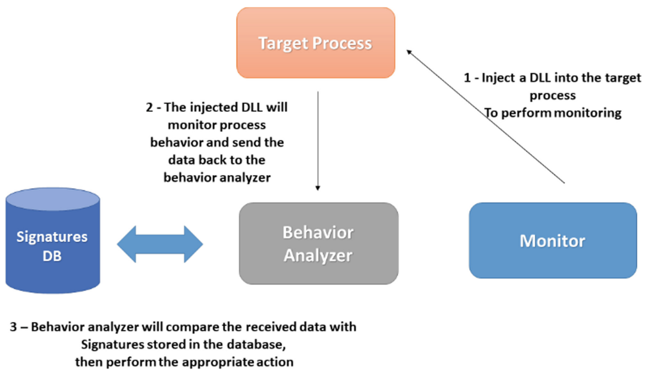

Target Process
1 - Inject a DLL into the target process To perform monitoring
2 - The injected DLL will monitor process behavior and send the data back to the behavior analyzer
Signatures DB
Behavior Analyzer
Monitor
3 – Behavior analyzer will compare the received data with Signatures stored in the database, then perform the appropriate action

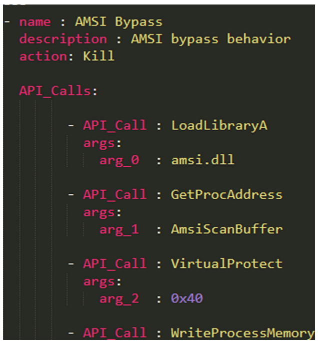
Fig. 3. Dynamic signature based detector architecture

5 Experimental Results

The experiments are conducted using Windows 10 operating system, Intel Core i7, and 8 GB ram. The components of our detector are written in C++, and compiled with Visual Studio 2017. API hooking is performed using EasyHook library [32], and yaml-cpp [33] is used for YAML parsing. Our detector uses PsSetLoadImageNotifyRoutine [34] to register a driver callback that will notify the monitor whenever an image is loaded or mapped into memory, this will allow the monitor to inject a DLL in very early process initialization stage in order to perform API hooking. For experimental purpose, we have written three signatures to identify three different malicious behavior. For the sake of simplicity, we have hooked the highest level API. Consider hooking lowest level API usually found in the ntdll.dll when dealing with malware.

5.1 AMSI Bypass Detection

Antimalware Scan Interface, AMSI [35] is an interface provided by Microsoft that allows applications and services to integrate with antimalware products. AMSI is capable of detecting and blocking of script-based attacks. Malware authors aim to bypass AMSI in order to execute any Powershell script without being detected. Figure 4 shows a signature that we can apply to the AMSI bypass technique available on the internet [36].

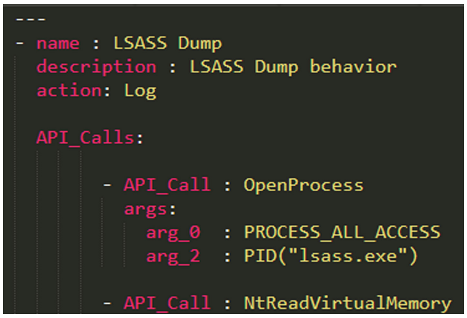


```
- name : AMSI Bypass
  description : AMSI bypass behavior
  action: Kill

  API_Calls:

        - API_Call : LoadLibraryA
          args:
            arg_0  : amsi.dll

        - API_Call : GetProcAddress
          args:
            arg_1  : AmsiScanBuffer

        - API_Call : VirtualProtect
          args:
            arg_2  : 0x40

        - API_Call : WriteProcessMemory
```

Fig. 4. AMSI bypass signature

5.2 Lsass.exe Process Dump Detection

Malware dump Lsass.exe process, in order to obtain windows credentials [37]. The signature in Fig. 5 aims to prevent any process from dumping lsass.exe memory.

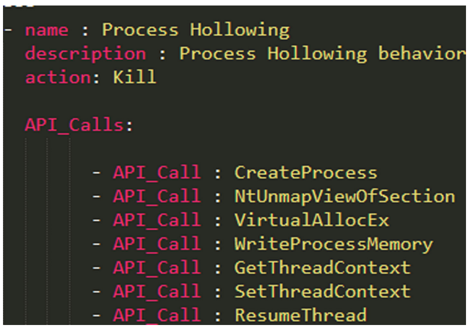


```
---
- name : LSASS Dump
  description : LSASS Dump behavior
  action: Log

  API_Calls:

        - API_Call : OpenProcess
          args:
            arg_0  : PROCESS_ALL_ACCESS
            arg_2  : PID("lsass.exe")

        - API_Call : NtReadVirtualMemory
```

Fig. 5. LSASS dump behavior

5.3 Process Hollowing Detection

Process hollowing [38] allows the injection of entire executable files into a target process. Malwares use this technique to evade anti-virus detection. We aim to detect and prevent this technique using the signature form Fig. 6.



```
- name : Process Hollowing
  description : Process Hollowing behavior
  action: Kill

  API_Calls:

          - API_Call : CreateProcess
          - API_Call : NtUnmapViewOfSection
          - API_Call : VirtualAllocEx
          - API_Call : WriteProcessMemory
          - API_Call : GetThreadContext
          - API_Call : SetThreadContext
          - API_Call : ResumeThread
```

Fig. 6. Process-hollowing signature

The following Table 1 represents a synthesis of the experimental results:

Table 1. Synthesis of results

Behavior	API call count	Result	False positive
AMSI bypass	4	OK	NO
LSASS dump	2	OK	YES
Process hollowing	7	OK	NO

Using the three signatures, we were able to identify the malicious process (Result "OK"); however, "LSASS dump" signature can lead to some false positive (False positive "Yes").

6 Conclusion and Future Works

A dynamic signature model is proposed to represent the behavior of a fileless malware. The signature contains API call sequence that identifies a behavior along with past arguments. Additional information could be mentioned in the signature, like the process name or the parent process name. A detector takes this signature as input then identifies malicious process using API hooking. Our detector showed good results for identifying fileless malware tricks. However, it can lead to a false positive alert. When a new malware technique appears, malware analyst can write a specific signature to identify it so that defenders can update their signatures database to include mitigation for the new techniques. This will reduce malware spreading considerably.

Improvements will be made to the signature structure in order to handle much more behavior criteria and reduce the false positive rate.

References

1. Cohen, F.: Computer viruses. Computers & Security (1987)
2. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Miami Beach, FL, pp. 421–430 (2007)
3. Egele, M., Scholte, T., Kirda, E., Kruegel, C.: A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surv. **44**(2), 1–42 (2012)
4. Ki, Y., Kim, E., Kim, H.: A novel approach to detect malware based on API call sequence analysis. Int. J. Distrib. Sensor Netw. **11**(6), 659101 (2015)
5. Chen, L., Ye, Y., Bourlai, T.: Adversarial machine learning in malware detection: arms race between evasion attack and defense. In: 2017 European Intelligence and Security Informatics Conference (EISIC) (2017)
6. Afianian, A., Niksefat, S., Sadeghiyan, B., Baptiste, D.: Malware Dynamic Analysis Evasion Techniques: A Survey (2018)
7. Moubarak, J., Chamoun, M., Filiol, E.: Comparative study of recent MEA malware phylogeny. In: 2017 2nd International Conference on Computer and Communication Systems (ICCCS) (2017)
8. Naik, N., Jenkins, P., Savage, N., Yang, L.: Cyberthreat hunting - part 1: triaging ransomware using fuzzy hashing, import hashing and YARA rules. In: 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), New Orleans, 23–26 June 2019 (2019)
9. Yara.readthedocs.io: Welcome to YARA's documentation! — yara 3.8.1 documentation (2019). https://yara.readthedocs.io
10. Trautman, L., Ormerod, P.: Wannacry, ransomware, and the emerging threat to corporations. SSRN Electron. J. (2018)
11. Homayoun, S., Dehghantanha, A., Ahmadzadeh, M., Hashemi, S., Khayami, R.: Know abnormal, find evil: frequent pattern mining for ransomware threat hunting and intelligence. IEEE Trans. Emerg. Top. Comput. (2017)
12. Cabaj, K., Mazurczyk, W.: Using software-defined networking for ransomware mitigation: the case of cryptowall. In: IEEE Network, vol. 30, no. 6, pp. 14–20, November–December 2016
13. Ravi, C., Manoharan, R.: Malware detection using windows API sequence and machine learning. Int. J. Comput. Appl. **43**(17), 12–16 (2012)
14. Sethi, K., Chaudhary, S., Tripathy, B., Bera, P.: A novel malware analysis framework for malware detection and classification using machine learning approach, pp. 1–4 (2018)
15. Cuckoosandbox.org: Cuckoo Sandbox - Automated Malware Analysis (2019). https://cuckoosandbox.org/
16. Yan, W., Zhang, Z., Ansari, N.: Revealing packed malware. In: IEEE Security & Privacy, vol. 6, no. 5, pp. 65–69, September–October 2008
17. Sai, S.V., Kohli, P., Bezawada, B.: Signature generation and detection of malware families, vol. 5107, pp. 336–349 (2008)
18. Docs.microsoft.com: PE Format - Windows applications (2019). https://docs.microsoft.com/en-us/windows/desktop/debug/pe-format

19. Docs.microsoft.com: About WinINet - Windows applications (2019). https://docs.microsoft.com/en-us/windows/desktop/wininet/about-wininet
20. Undocumented.ntinternals.net: NTAPI Undocumented Functions (2019). https://undocumented.ntinternals.net/. Accessed 10 May 2019
21. Shafiq, M.Z., Tabish, S.M., Mirza, F., Farooq, M.: PE-Miner: mining structural information to detect malicious executables in realtime. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) Recent Advances in Intrusion Detection. RAID 2009. Lecture Notes in Computer Science, vol. 5758. Springer, Heidelberg (2009)
22. Katja, H.: Robust Static Analysis of Portable Executable Malware, Master Thesis in Computer Science, HTWK Leipzig
23. GitHub: fireeye/flare-floss (2019). https://github.com/fireeye/flare-floss/blob/master/doc/theory.md
24. Blackhat.com (2019). https://www.blackhat.com/docs/us-15/materials/us-15-Choi-API-Deobfuscator-Resolving-Obfuscated-API-Functions-In-Modern-Packers.pdf
25. Docs.microsoft.com. (2019). _PEB. https://docs.microsoft.com/en-us/windows/desktop/api/winternl/ns-winternl-_peb
26. Detours: Binary interception of Win32 functions. In: Hunt, G., Brubacher, D. (eds.) Third USENIX Windows NT Symposium. USENIX, July 1999
27. Marhusin, M.F., Larkin, H., Lokan, C., Cornforth, D.: An evaluation of API calls hooking performance. In: 2008 International Conference on Computational Intelligence and Security, Suzhou, pp. 315–319 (2008)
28. Mansfield-Devine, S.: Fileless attacks: compromising targets without malware. Netw. Secur. **2017**(4), 7–11 (2017)
29. Chan, K.T., Lenard, C., Mills, T.: An Introduction to Markov Chains (2012)
30. Yaml.org: The Official YAML Web Site (2019). https://yaml.org/
31. Sikorski, M., Honig, A.: Practical malware analysis. San Francisco (California, EEUU) (2012)
32. Easyhook.github.io (2019). EasyHook. https://easyhook.github.io/
33. GitHub: jbeder/yaml-cpp (2019). https://github.com/jbeder/yaml-cpp
34. Docs.microsoft.com: PsSetLoadImageNotifyRoutine function (ntddk.h) - Windows drivers (2019). https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetloadimagenotifyroutine
35. Docs.microsoft.com: Antimalware Scan Interface (AMSI) - Windows applications (2019). https://docs.microsoft.com/en-us/windows/desktop/amsi/antimalware-scan-interface-portal
36. Blog, Z.: How to bypass AMSI and execute ANY malicious Powershell code, zc00l blog (2019). https://0x00-0x00.github.io/research/2018/10/28/How-to-bypass-AMSI-and-Execute-ANY-malicious-powershell-code.html
37. Blog.gentilkiwi.com: mimikatz| Blog de Gentil Kiwi (2019). http://blog.gentilkiwi.com/mimikatz. Accessed 10 May 2019
38. Attack.mitre.org: Technique: Process Hollowing - MITRE ATT&CK™ (2019). https://attack.mitre.org/techniques/T1093