Collection of the main Anti-Virus detection and bypass techniques

Jérémy Donadio[1,3], Guillaume Guerard[1], and Soufian Ben Amor[2]

[1] Léonard de Vinci Pôle Universitaire, Research Center, 92 916 Paris La Défense, France
[2] LI-PARAD Laboratory EA 7432, Versailles University, 55 Avenue de Paris, 78035 Versailles, France
[3] Research student
f_name.l_name@devinci.fr
f_name.l_name@uvsq.fr

Abstract. A large amount a new threats, technologies and business models have emerged in the cybersecurity area through the COVID-19 pandemic. The remote work involved unplanned cloud migrations and swift procurement of IT products and services the remote landscape. In this context, the role of anti-viruses is crucial for the private life and work. In this paper, we study the workings of anti-viruses as to understand how to avoid them. We created a collection of the main bypass techniques whilst analyzing their respective advantages and drawbacks. We show that it is possible to avoid both static and emulation analyses, while enunciating the techniques and approaches being used.

Keywords: Malware Bypass · Anti Virus · Static Analysis · Dynamic Analysis

1 Introduction

More than ever, the Covid-19 pandemic revealed that both our public and private computerized systems were vulnerable to computer attacks [8,19,26]. Anyone inclusing hospitals [1], private companies [3] or an individual for example as energy consumers [25] are the victims of such attacks like the well-know WannaCry malware.

The sole fact that 90% of all data breaches were initiated by a human error showcases the urgency to detect them. The objective of the black hats[4] is to achieve monetary gain through stealing or destroying data. It is worthwhile to note that the most common ransomwares are: Lockergoga, Katyusha, WannaCry, Jigsaw, Pewcrypt, Ryuk, Dharma, Gandcrab, Revil, Samsam [6]. we recommend the following books for a deep understanding of cybersecurity, noting attacks and issues [30,31].

[4] The term *black hat* designates a hacker who breaches into a computerized system or network with malevolent intentions. Such motives usually are a monetary gain.

Cybercrime statistics estimate the economic impact of the ransomware phenomenon to be of around 600 billion dollars which is more profitable than the drug market, estimated to a 400 billion dollars value[5].

To introduce the problem of cybercrime, we refer to the following statistics from ENISA[6], and specifically cite the following:

- 10.1 billion euros were paid in ransom in 2019 (compared to 3.3 billion in 2018).
- 66% of health organizations were victim of at least one ransomware attack[7] in 2019.
- 45% of targets pay the ransom.
- 28% of security incidents within businesses have been attributed to malwares.
- Ransomware detection within businesses augmented by 365% in 2019.

According to the same source, 88% of companies that have more than one million folders unknowingly have more than 100,000 folders that are publicly accessible. Furthermore, according to Varonis Systems' reports[8], 30% of companies have more than 1,000 sensitive folders accessible publicly. Additionally, 57% of companies have more than 1,000 folders with inadequate access permissions.

We therefore formulate the following hypothesis: the existence of a business implies the existence of a risk; the more this business is important, the higher the risk is. This paper will explore the workings of Anti-Virus software, as well as means to avoid them. It will provide a global yet detailed view of the working and possible bypass techniques of said software.

Strengthening anti-virus implementations is therefore a critical subject for our society, Kaspersky provides a real time map showing the cyberattacks[9]. The aim of this paper is to focus on breaches in these software, as to diminish the risk faced by the stakeholders making up our society. We will first study the workings of anti-viruses, then we will list the ways in which one can avoid them. This will shed light onto some of the flaws of anti-malware software, which developers can then patch.

In the second section, we define the terms of malware and anti-virus, as well as detailing the ways in which malwares are detected. The third section will showcase bypass techniques to the previously listed detection methods. The last section will enumerate the results from the experiments that we conducted.

[5] Nick Galov. *40 Worrisome Hacking Statistics that Concern Us All in 2020.* In february 18, 2021.

[6] The European Union Agency for Cybersecurity. *https://www.enisa.europa.eu/publications/ransomware*

[7] Ransomware: program that first encrypts all a computer's data, then asks for a ransom in return of the data being decrypted.

[8] Varonis. *2018 Global Data Risk Report From The Varonis Data Lab.* In 2019.

[9] https://cybermap.kaspersky.com/stats

2 Definition and categorization of Malwares and Anti-Malwares

There exist multiple types of malevolent programs such as viruses, rootkits, keyloggers, stealers, ransomwares, trojans, Remote Access Tools (R.A.T.), worms and spywares [32].

To keep the discussion simple, we will use the general appellation of *malware* to designate any malevolent program whose purpose is to spy on or corrupt a computer. This encompasses all programs aiming to control one or more peripherals (keyboard, mouse, camera, ...), but also the overarching system (visualization, copy, alteration, or deletion of files), or encrypt folders (ransomware).

Simplifying the term malware is important as this study focuses on their modes of action, detection, and bypass. As to protect the computers of businesses and individuals, some cyber-protection parties started to develop software dedicated to detecting and protect against malwares. Such softwares are known as anti-malwares, commonly referred to as anti-viruses.

We can identify two main functionalities of anti-virus softwares, being the detection and neutralization of threats. Focusing on the detection step, we can see that it is usually broken-down into the following three steps:

1. Static analysis: analysis without executing the program.
2. Virtual Machine (VM) analysis: analyzing program execution within a virtual machine.
3. Real time (proactive) analysis: executing the program natively and analyzing it in real time.

2.1 Static analysis

Static analysis aims to analyze the code of a program without executing it, which is comparable to analyzing *inert code*. By this method, one can detect a malware without executing it and therefore corrupt the computer system. In practice, the anti-virus software analyzes the binary code of the program to determine its degree of malevolence. We will then describe some of the commonly used detection methods: the signature, pattern analysis, spectral analysis, by machine learning, by checksum, by using additional data.

The signature – Every anti-virus possesses what is referred to as a dictionary. Such dictionaries are large databases composed of short byte sequences, or signatures[10] from malevolent programs. When analyzing a program, the anti-virus cross-checks the database with the program's binary code, looking for instances of known signatures. If such a match is found, then the program is classified as potentially dangerous as shown in Figure 1.

[10] Joxean Koret and Elias Bachaalany. The Antivirus Hacker's Handbook. September 28, 2015.

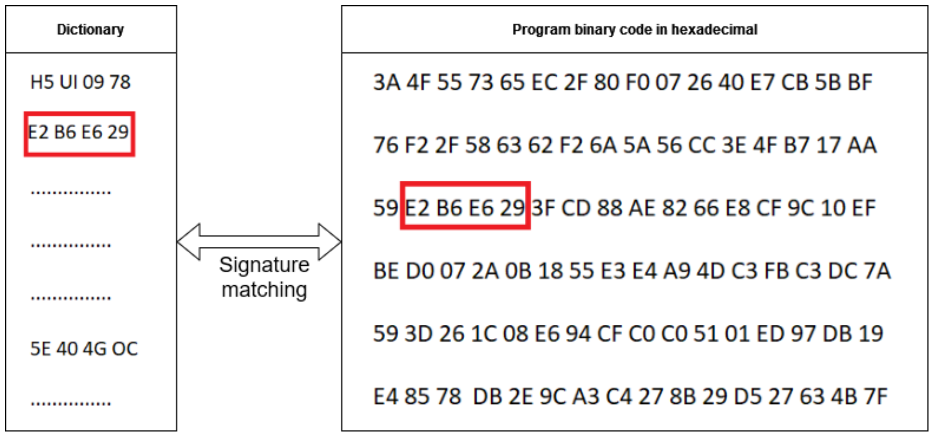

Dictionary
H5 UI 09 78
E2 B6 E6 29
5E 40 4G OC
Signature matching
Program binary code in hexadecimal
3A 4F 55 73 65 EC 2F 80 F0 07 26 40 E7 CB 5B BF
76 F2 2F 58 63 62 F2 6A 5A 56 CC 3E 4F B7 17 AA
59 E2 B6 E6 29 3F CD 88 AE 82 66 E8 CF 9C 10 EF
BE D0 07 2A 0B 18 55 E3 E4 A9 4D C3 FB C3 DC 7A
59 3D 26 1C 08 E6 94 CF C0 C0 51 01 ED 97 DB 19
E4 85 78 DB 2E 9C A3 C4 27 8B 29 D5 27 63 4B 7F

Fig. 1. Example of detection by signature.

It is also possible to apply this method with program hashes[11]. In this instance, the anti-virus possesses a dictionary of known malevolent program hashes. Therefore, when analyzing a new program, the anti-virus computes its hash and cross-checks it with its dictionary, looking for a match indicating the program is potentially malevolent.

This malware detection method is one of the first approaches ever invented and remains one of the most commonly used. Its weak point is that one needs to already know a malware to detect it. This method is therefore less and less efficient due to the creation of new types of malwares as polymorphic malwares and because the overall number of malwares ever created augments exponentially.

Pattern analysis and disassembly – It is possible to deduct a program's behavior from analyzing its instruction sequences. Looking at a program's behavioral pattern, one can infer whether or not it is malevolent as shown in Figure 2. There are numerous ways to identify and understand such patterns [17]. One can analyze the PE (.EXE) format of the program, disassemble it, and possibly apply machine learning methods [5].

To summarize, the aim is to understand the behavior of some of the program's functions, rather than search for a ”precise byte sequence” within the program's binary code.

Spectral analysis – Spectral analysis entails analyzing the number of occurrences of all instructions within the program. The number of instructions or possible states attainable by the processor (CPU) being determined, one can

[11] Hash: a sequence of numbers and letters generated by a cryptographic hashing function.

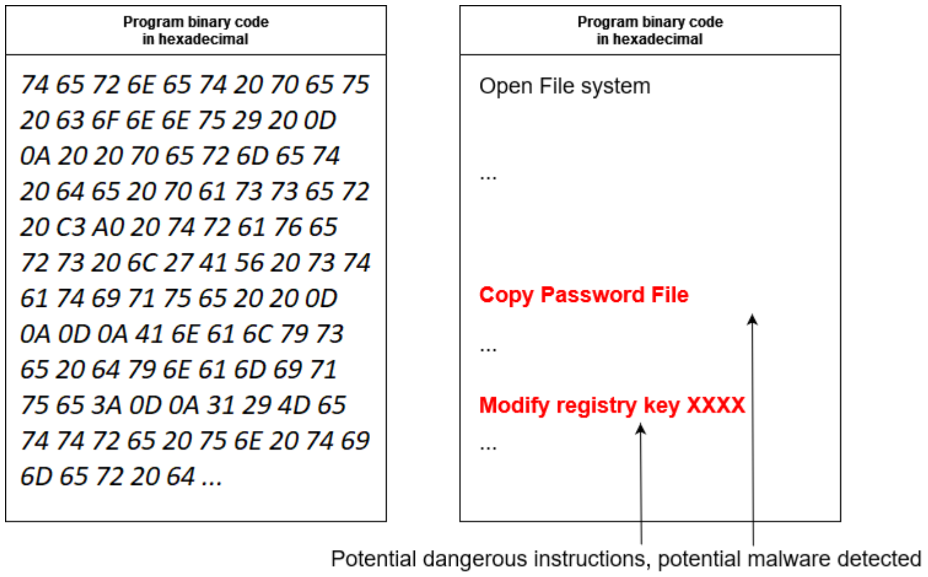

Program binary code
in hexadecimal
74 65 72 6E 65 74 20 70 65 75
20 63 6F 6E 6E 75 29 20 0D
0A 20 20 70 65 72 6D 65 74
20 64 65 20 70 61 73 73 65 72
20 C3 A0 20 74 72 61 76 65
72 73 20 6C 27 41 56 20 73 74
61 74 69 71 75 65 20 20 0D
0A 0D 0A 41 6E 61 6C 79 73
65 20 64 79 6E 61 6D 69 71
75 65 3A 0D 0A 31 29 4D 65
74 74 72 65 20 75 6E 20 74 69
6D 65 72 20 64 ...
Program binary code
in hexadecimal
Open File system
...
Copy Password File
...
Modify registry key XXXX
...
Potential dangerous instructions, potential malware detected

Fig. 2. Simplified example of a behavioral analysis.

compute the number of times, the frequency, the CPU is in each of these states. A program spectrum corresponds to a graph of the all the possible CPU instructions, against the frequency of each instruction.

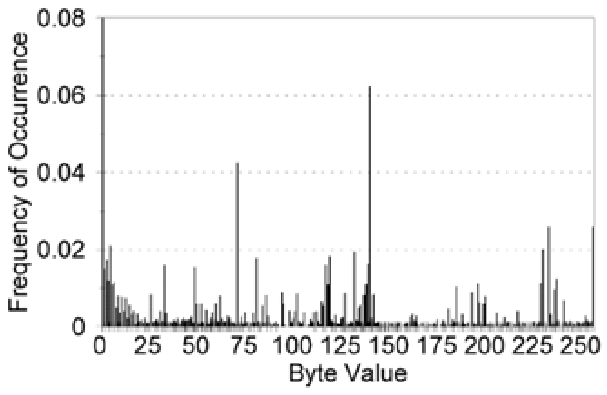


Fig. 3. Example of a normal program spectrum [18].

This spectrum in Figure 3 depicts a simplified representation of reality, where there are 255 possible byte values with their corresponding frequency. For this analysis to be useful, on needs to analyze the actual instruction rather than the byte values. We now show the spectrum of an encrypted program in Figure 4.

This spectrum reveals that all the instructions associated to the program's binary code are biased due to the program being encrypted. It is also possible to use spectral analysis to detect polymorphic malwares[12], by computing a similarity index based on the programs' instructions, for which we will observe that their specters are identical.

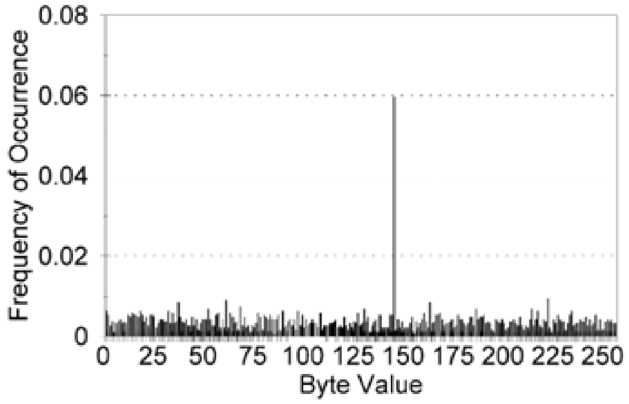


Fig. 4. Example of a secured program spectrum [18].

Using machine learning in static analysis – It is possible to use machine learning techniques to achieve a more accurate program analysis, thus augmenting the odds of detecting malevolant programs. Shijo et Al. [29] presents a method consisting in integrating a large number of known malwares and inoffensive programs in the training set. Features are then obtained by extracting Printable String Information [11,12] contained within the program's binary code. This data is then used as inputs for different classification algorithms.

Integrity checksum Analysis – Checksum analysis consists in gathering multiple pieces of information regarding a file, such as its name, size, last-modified date, hash, as well as other parameters as presented in Figure 5. Once the data has been acquired, the anti-virus will attempt to sum them as to obtain a unique value. Any attempt from the malware to duplicate itself within the file will result in an automatic change in file size, hash, last-modified date, and other variables.

[12] malware having the capacity to modify its signature every time it is generated.

The resulting checksum will then change, and if the value is deemed too different from the original one, then it is possible that the program is a malware. On the other hand, if one manually modifies or uses the file, then the checksum will not change drastically, and no alert will be raised. This technique therefore creates an important number of false positives, which is why it is important to use it alongside other analysis methods.

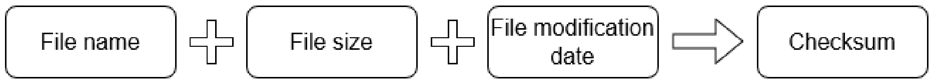


Fig. 5. Analysis by integrity checksum.

Using additional data – It is possible to use other data elements to gauge how dangerous is a program [15]. For instance, one can verify the libraries or Application Programming Interfaces (API) being used to determine whether they are documented [24]. One can also verify interrupt vectors, analyze the file format, etc. It is also important to note that there exists a large number of other static analysis techniques such as integrity verification [7], and that a lot of them now use machine learning techniques [9]. Further discussion of such statistical analysis techniques has been omitted due to their large similarity to the aforementioned techniques.

2.2 Emulation Analysis

VM analysis is used to detect a malware in the case where it succeeds in going through static analysis undetected. As previously stated, static analysis has for aim to analyze the code without running it. However, it is possible for this analysis to not yield any results, making executing the code a necessary step. Because this kind of testing can cause damage to the computer system, one usually runs it in an insulated *sandbox*. Such an environment is comprised of a virtual machine separated from the main system in which the program is executed. This way, any harm that is caused by the malware is contained to within the virtual machine, thus not affecting the main system's integrity.

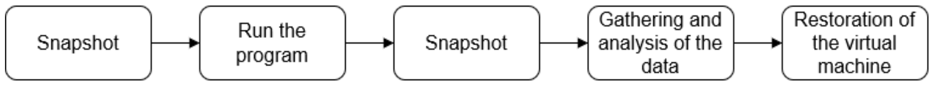

Snapshot
Run the program
Snapshot
Gathering and analysis of the data
Restoration of the virtual machine

Fig. 6. Analysis of a program by emulation.

The different steps carried during VM analysis are (also see Figure 6):

1. Creating the virtual machine.
2. Save the VM (in the event when we want to go back to the save point to execute the program once again).
3. Run the program.
4. Second save (in the event when one wants to re scan the program's execution result - the saves are there to gain time in the event when one wants to jump to a specific step in the analysis).
5. Gathering and analysis of the data.
6. Restoration of the virtual machine.

The objective is therefore to execute the program in a secure environment and analyze its behavior to determine whether or not it is malevolent. Such an analysis is based upon many axes, some of the main ones being:

– Registry.
– File system.
– Network analysis.

We will detail each of these axes below.

The registry – The registry is used to determine which programs are launched upon starting up the system. The registry also manages these program's access permissions and configurations. A malware will be forced to interact with the registry if it attempts to present itself to the system as a legitimate program or tried to become persistent (be present upon restarting the computer). Therefore, registry analysis is a primordial step in guaranteeing the system's security.

The file system – Analyzing the file system is essential as the malware will likely attempt to modify a file or a process, warranting particular attention.

The network analysis – In the event where the malware succeeds in finding the targeted information, it will likely create a network connection with a third party like a hacker as to send over the stolen data. Network analysis is therefore a major step in guaranteeing system reliability.

There exist other axes that are being monitored, and the list provided in this paper is not exhaustive.

2.3 Real-time analysis (dynamic/proactive)

In the event where previously mentioned analysis techniques fail, one can use proactive analysis [10]. Proactive analysis consists in a real-time analysis of the system where each anti-virus uses multiple *agents* to protect the system. An agent is a program whose purpose is to guard against a specific type of intrusion. Such intrusions include deletion of files from another program, transfer of packets between the user and the outside world through a network, mail, web, etc. Each agent is therefore tasked with analyzing a specific part of the main system.

3 Anti-virus Bypass and tests

We will detail the different methods that can be used to bypass each type of analysis. In order to put into practice the theoretical methods discussed in this paper, we have developed a keylogger as well as a ransomware, while testing most of the aforementioned bypassing methods.

Here are the complementary information about the operating system and the installed anti-virus:

- Operating system used: Windows 10 family on version 19042.928.
- Anti-Virus tested: Avast premium 21.2.2455 with 1.0.644 interface.
- Viral database: Avast premium 210609-10.

3.1 Bypassing static analysis

In this section, we will detail the main static analysis bypassing techniques.

Writing your own malware – Static analysis' main pitfall is that it requires the anti-virus to have encountered the malware at least once in order to detect it. This implies that if one writes their own malware, the chances of it being detected by an anti-virus are negligible.

Changing signatures – Another way to make it through static analysis is to change a malware's signature. If the malware's signature is known from the anti-virus, it is possible to change it manually by cutting the program in small pieces using a tool such as *UK splitter*, then analyze such pieces to locate which ones are being flagged as malevolent, then change them using a hexadecimal editor [28]. It is also possible to create a program that has the ability to change its signature at each instantiated code. Such malevolent programs are called *polymorphic malwares*.

Malware encryption – We previously mentioned that static analysis was based upon comprehension of the program's binary code. It is therefore sufficient to encrypt the program's content, thus its instruction, to make it illegible and undecryptable by the anti-virus. The program is thus fully encrypted, bypassing all static analysis methods. Once the program is executed, it can decrypt itself and execute its code. Such *crypters* are downloadable on the internet and enable encryption of one's program as to bypass anti-viruses.

```
Int main()
    {
     DecryptCode();
     ExecuteShellCode();
     Return 0;
    }
```

Packers – The primary function of a *packer* is to compress a program without altering its function. Packers are used by software/game developers to make the produced executable illegible as to prevent attempts at reverse-engineering or piracy. Packers have the ability to package, compress and encrypt the executable. Such abilities are very useful to conceal a malware. There exist multiple popular packers such as UPX [23] or Themida [16].

Supplementary techniques – As stated in the *malware encryption* part, it is possible to carry out multiple actions to bypass the analysis of a program [21]. Such actions include disabling the Firewall, the Task Manager, the Command Prompt, the User Account Control, etc. This list is non-exhaustive and there exist multiple methods to bypass static analysis [4], such as the use of concealers [2]. However, the techniques remain the same:

1. Encrypt the program.
2. Make it unreadable.
3. Write the malware from scratch so that its signature is unknown.

3.2 Static analysis bypass tests

Changing signatures manually works. We have been able to make a malware undetectable from anti-viruses by modifying its signature. It is important to note than this method can be time consuming, and that the malware may become dysfunctional. However, after changing the signature, our malware remained undetectable and functional. You will find two screenshots in the annex, showing the detection of the malware before and after execution of this method.

Changing signatures using a polymorphic malware works as well. It is important to note than this method will only work if the polymorphic malware is well programmed, and that it drastically changes signatures frequently enough. Otherwise, it may be detected by various machine learning or spectral analysis methods.

We also wrote a keylogger and a ransomware, none of which were detected by statistical analysis despite their potential dangerousity.

Using free packers did not yield successful results. However, we believe that the creation of a complex packer unknown to the anti-virus would work. We were not able to test paid packers but it would be logical that they yield a higher success rate compared to the free ones. Nevertheless, we think that they may not work against all anti-viruses.

The testing of malware encryption is currently in progress. We have set the objective of writing our own program capable of encrypting code. We have not been able to test already existing implementations, but we think that this method should work in light of our previous test cases.

3.3 Emulation analysis bypass

The main issue with emulation analysis is the time it takes to analyze the program, which is easily detectable. Indeed, VM analysis starts when a program is executed, and it is only once a program has been analyzed that it is truly run for the user. This signifies that the longest the analysis lasts, the longer the user will wait. In order to provide an enjoyable user experience, it is important that the analysis time remains short. The second problem is that the VM requires a certain number of resources to run (RAM, CPU), making the VM startup detectable by the malware. The primary methods to bypass emulation analysis are the following:

– Wait for a certain lapse of time before executing the malware.
– Detect the VM and remain inactive while it is analyzing.
– Prevent the VM from starting.

We now showcase an illustration of some of the aforementioned methods with some results. There exist other methods allowing to bypass emulation analysis. In order to learn more, Nasi et Al.'s article [22] is oftentimes used in this section. The other techniques being used are all analogous to those already discussed (wasting time, detect or block the VM) which is why we omitted further discussion of said methods.

Excessive Increment – The aim is to make the program stall for a certain lapse of time before executing it. This effect is achieved by using a for loop that is tasked with waiting for an excessive amount of time before executing the remainder of the program. The idea is that the program does not start during the analysis. The VM analysis only observes a useless increment and will rule the program as benevolent. Here is a working code example:

```
Void LoopFud()
    {
     ShowWindow(GetConsoleWindow(), SW\_HIDE);
     for (int i = 0; i < 4000; i++) {Sleep(1);}
    }
```

Timer – Similar to the previous function, except that one uses a timer rather than a for loop.

```
Void Timer()
    {
     ShowWindow(GetConsoleWindow(), SW_HIDE);
     int total_time = 60;
     time_t current_time = time(0);
     while(time(0) - current_time < total_time){Sleep(1);}
    }
```

VM detection – As previously stated, setting up a VM necessitates a certain quantity of resources (RAM, CPU). It is therefore possible to detect the VM by monitoring program memory allocations as well as running processes. Once detected, the program should be paused while the VM is running on the system. The bypassing method is summarized with the following pseudo-code:

1. If a lot of memory is allocated or a process asks for a certain quantity of resources, then the malware is most likely contained within a VM.
2. While the malware is contained within the VM : do not execute the program.

It is also possible to detect the VM with respect to its process name.

Too much memory – It is also possible to prevent the emulation analysis from starting entirely. Indeed, a VM needs to allocate memory to run the program when starting up. However, the program will not be able to run if the program requests too much memory.

Say My Name – When the code is emulated within the VM, no other process bearing the same name can be started. The program can then conclude that as long as its name does not appear in another process, then it is contained within a VM. It can therefore choose to execute if and only if its name appears in another process.

3.4 Emulation analysis bypass tests

The Excessive Increment method (for loop) proved successful. The code used in the description of this method is the code that has been used to successfully bypass the emulation analysis.

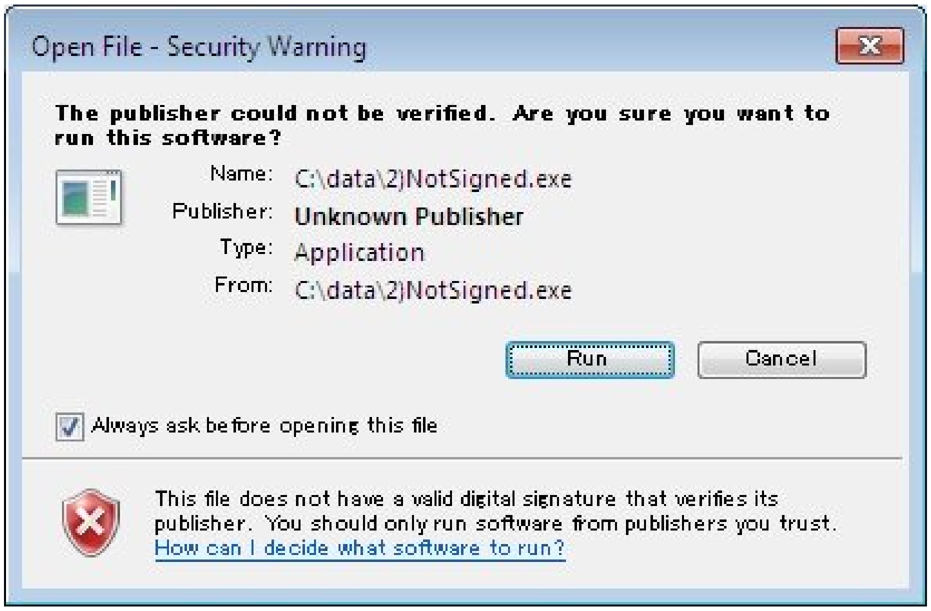
The Timer method also proved successful using the code in the description.

The Say My Name method seems to be working. We do have access to the programs' process name, and we can verify whether it is within a VM. However, we do not have access to other processes' names. This detail is irrelevant for this part but is a potential source for issues in the Dynamic Analysis Bypass part.

3.5 Dynamic or real-time analysis bypass

The dynamic analysis is the most complex one to bypass due to its technicity. There exist two main methods to bypass this analysis:

1. Make the anti-virus believe that we have the credentials to perform an admin task (erase a file, modify a file, etc).
2. Insert oneself within or impersonate a known legitimate program to carry out the desired task.



Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: C:\data\2)NotSigned.exe
Publisher: Unknown Publisher
Type: Application
From: C:\data\2)NotSigned.exe

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust.
How can I decide what software to run?

Fig. 7. Example of a window displaying the code signature certificate.

Legitimacy gain – The most commonly used method to gain legitimacy is to use a code signature certificate as shown in Figure 7. When you execute a program on your computer, it will display a window mentioning the type of file being executed, the name of the publisher, and the file location. Here is an example of a window appearing when executing a program.

Here, we can see that the program is unsigned because the publisher's name is unknown. However, if you install trustworthy software such as Word, you will have a publisher name like Microsoft, indicating that the file is safe. One can purchase such a certificate for prices between 200 and 1000 euros in many websites, allowing malwares to bypass dynamic analysis.

Insertion within a legitimate program – In this instance, the malware attempts to impersonate a legitimate program already existing on the computer. It will therefore try to insert itself in a running process. Such an endeavor is called an injection [27], which happen at the memory or DLL level most of the time. Other injection methods include Shellcode Injections, Inline Hooking and others [20].

3.6 Targeting the core of the operating system

The methods that have been discussed above work at the application layer of the operating system. However, it is possible to create malwares acting upon the kernel or bios layers, making them undetectable. Such malwares operating on low level layers are also referred to as rootkits, and their main specificity is that they are undetectable from anti-viruses. Among all the methods used to bypass anti-viruses, the *Blue Pills* method [14], developped by Joanna Rutkowska, will never be able to be stopped or detected on consumer operating systems (windows, linux, unix, etc), making it one of the most powerful yet elegant methods to date.

The malware will download an operating system on one of the computer's storage devices (flash drive, hard dirk, etc). Once downloaded, the operating system will behave like a VM in which the native operating system will run. In other words, launching Windows will result in it executing within another operating system [13]. This other operating system will be able to spy on your computer and render all anti-viruses inefficient as they are limited by the bounds of the native operating system and therefore will be oblivious to the other operating system's existence. The process of the Blue Pills' attack is shown in Figure 8.

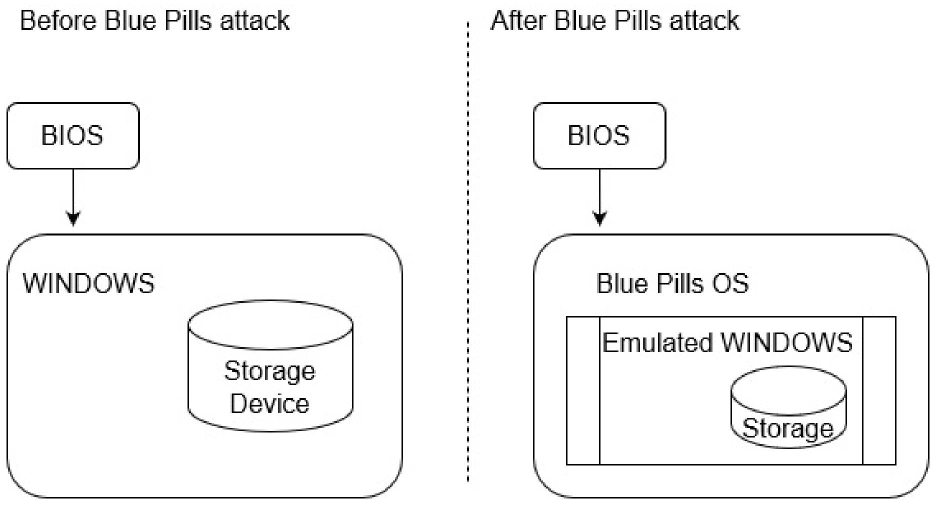


Fig. 8. Blue pills' attack.

You now understand that the name Blue Pills is a reference to The movie "The Matrix" because it illustrates a machine running within another machine. It is also important to note that the researcher who created the Blue PILLS attack also created an operating system known as *Qube OS*, which is able to prevent Blue Pills attacks.

4 Conclusion

To conclude, we underline in this paper that static analysis is based on a program's binary, without running it. However, this method can be insufficient in case where the program is eligible or if the anti-virus never encountered the malware.

With respect to emulation analysis (VM) which aims at studying the behavior of a program by making it run within a sandbox, we can consider that it will become obsolete because it is easy to detect or block the VM, as well as make our program inactive during the analysis timeframe.

The most efficient and hard to bypass analysis is the dynamic one. There exist however multiple ways to pass through, and multiple of these methods, such as by injection, will be tested in future work.

In the future, our work and research will focus on the functioning and bypassing of dynamic analysis, especially methods by injection.

References

1. Akselrod, H.: Crisis standards of care: Cyber attack during a pandemic. Annals of Internal Medicine **174**(5), 713–714 (2021)
2. Alazab, M., Venkatraman, S., Watters, P., Alazab, M., Alazab, A.: Cybercrime: the case of obfuscated malware. In: Global security, safety and sustainability & e-Democracy, pp. 204–211. Springer (2011)
3. Alladi, T., Chamola, V., Zeadally, S.: Industrial control systems: Cyberattack trends and countermeasures. Computer Communications **155**, 1–8 (2020)
4. Anderson, H.S., Kharkar, A., Filar, B., Roth, P.: Evading machine learning malware detection. Black Hat (2017)
5. Aslan, Ö.A., Samet, R.: A comprehensive review on malware detection approaches. IEEE Access **8**, 6249–6271 (2020)
6. Aurangzeb, S., Aleem, M., Iqbal, M.A., Islam, M.A., et al.: Ransomware: a survey and trends. Journal of Information Assurance & Security **6**(2), 48–58 (2017)
7. Calderon, P., Miavril, V., Présent, P.: Contournement d'analyse dynamique de code viral
8. Fontanilla, M.V.: Cybercrime pandemic. Eubios Journal of Asian and International Bioethics **30**(4), 161–165 (2020)
9. Gandotra, E., Bansal, D., Sofat, S.: Malware analysis and classification: A survey. Journal of Information Security **2014** (2014)
10. Idika, N., Mathur, A.P.: A survey of malware detection techniques. Purdue University **48**, 2007–2 (2007)
11. Islam, R., Tian, R., Batten, L., Versteeg, S.: Classification of malware based on string and function feature selection. In: 2010 Second Cybercrime and Trustworthy Computing Workshop. pp. 9–17. IEEE (2010)
12. Islam, R., Tian, R., Batten, L.M., Versteeg, S.: Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications **36**(2), 646–656 (2013)
13. King, S.T., Chen, P.M.: Subvirt: Implementing malware with virtual machines. In: 2006 IEEE Symposium on Security and Privacy (S&P'06). pp. 14–pp. IEEE (2006)

14. King, S., Chen, P.: Subvirt: implementing malware with virtual machines. In: 2006 IEEE Symposium on Security and Privacy (S P'06). pp. 14 pp.–327 (2006). https://doi.org/10.1109/SP.2006.38
15. Lagadec, P.: Dynamic malware analysis for dummies. In: Symposium Sur la Sécurité des Technologies de l'information et des Communications, SSTIC (2008)
16. Lee, J.h., Han, J., Lee, M.w., Choi, J.m., Baek, H., Lee, S.j.: A study on api wrapping in themida and unpacking technique. Journal of the Korea Institute of Information Security & Cryptology **27**(1), 67–77 (2017)
17. Liu, W., Ren, P., Liu, K., Duan, H.x.: Behavior-based malware analysis and detection. In: 2011 first international workshop on complexity and data mining. pp. 39–42. IEEE (2011)
18. Ludwig, M.A.: The giant black book of computer viruses. American Eagle Publications (1998)
19. Macdonald, S., Jarvis, L., Lavis, S.M.: Cyberterrorism today? findings from a follow-on survey of researchers. Studies in Conflict & Terrorism pp. 1–26 (2019)
20. Mohanta, A., Saldanha, A.: Code injection, process hollowing, and api hooking. In: Malware Analysis and Detection Engineering, pp. 267–329. Springer (2020)
21. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). pp. 421–430. IEEE (2007)
22. Nasi, E.: Bypass antivirus dynamic analysis. Limitations of the AV model and how to (2014)
23. Oberhumer, M.F.: Upx the ultimate packer for executables. http://upx. sourceforge. net/ (2004)
24. Osorio, F.C.C., Qiu, H., Arrott, A.: Segmented sandboxing-a novel approach to malware polymorphism detection. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). pp. 59–68. IEEE (2015)
25. Plėta, T., Tvaronavičienė, M., Casa, S.D., Agafonov, K.: Cyber-attacks to critical energy infrastructure and management issues: Overview of selected cases (2020)
26. Ramadan, R.A., Aboshosha, B.W., Alshudukhi, J.S., Alzahrani, A.J., El-Sayed, A., Dessouky, M.M.: Cybersecurity and countermeasures at the time of pandemic. Journal of Advanced Transportation **2021** (2021)
27. Ray, D., Ligatti, J.: Defining code-injection attacks. Acm Sigplan Notices **47**(1), 179–190 (2012)
28. Scott, J.: Signature based malware detection is dead. Institute for Critical Infrastructure Technology (2017)
29. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Procedia Computer Science **46**, 804–811 (2015)
30. Sikorski, M., Honig, A.: Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press (2012)
31. Singer, P.W., Friedman, A.: Cybersecurity: What everyone needs to know. oup usa (2014)
32. Tahir, R.: A study on malware and malware detection techniques. International Journal of Education and Management Engineering **8**(2), 20 (2018)