Generating Antivirus Evasive Executables Using Code Mutation

Stefan Sabin NICULA
The Bucharest University of Economic Studies, Romania
niculastefan21@gmail.com

The paper is focus around developing a utility tool based on a python component and a C++ stub in order to compile executable Windows files that are capable of staying undetected to Antivirus solutions. The research process was focused around Antivirus software's ability to detect a malicious file and methods of bypassing the identified techniques. Dependencies and auxiliary links of the project are defined as inputs from the user as well as support software and frameworks designed to provide malicious payload with listeners and handlers for the generated shellcode. Overall, the utility tool is able to receive shellcode and one encryption key as input and generate malware in the shape of a Windows executable file that is able to successfully run and bypass Antivirus detection.
Keywords: *Antivirus bypass, Antivirus evasion, Executable malware, Shellcode execution, Undetected virus, Obfuscation techniques*

1 Introduction

The Windows operating system is heavily based on Portable Executable file format also known as PE files. This specific file format defines the encapsulation structure of the data for executable files, object code and dynamic link-libraries. This paper will focus on compile-time mutations for executables in the form of .exe file which are an extension of the PE format. Other extensions of the PE file format are .dll, .sys, .ocx, .efi, .scr.

The most common executable file .exe is also the most used one for creating and executing malware or malicious code. For the purpose of controlling the execution of such files or other corrupted files, antivirus solutions were developed. These solutions are specifically built in order to identify not only PE corrupted or compromised files but also other infected types of files from the system as well.

Even though these solution were created, the area of cover for trying to control all the possibilities for malicious or infected files has almost infinite proportions. However, there are certain patterns and templates, behaviors and actions that an antivirus solution can look for in order to try and fingerprint a specific files as bad intended or not.

Regarding the intentions of the files, the antivirus solution should also focus on trying not to interfere with the normal flow of the system, specifically with the normal files of the system. We can define a normal file, a file that has not been created or modified to cause harm, data loss or to have any negative impact over the system and its data. This knowledge, however, can actually be used and is actively used by attackers in order to forge malwares that are behaving like a normal legit executable but actually have malicious purposes and cause damage to the system.

When speaking of attackers, they can be defined as malicious actors from outside the target that are interfering with the system in an unauthorized manner or entities and people that are specifically employed or contracted in order to simulate or replicate these kind of malicious activities with corresponding authorization letter. Certain projects are specially designed in order to test the mentioned antivirus solutions and create real malware attack simulations in the means to improve future protection mechanism and asses the overall strength of the entity's protection and prevention actions against these types of attacks.

These types of projects have many different forms and names and can be part of a bigger activity that challenges the organization's overall protection methods and effectiveness. They can be found as Purple Teaming, phishing attacks, Red Team engagement or most commonly known as Penetration Testing project. A Penetration Testing engagement,

DOI: 10.12948/issn14531305/22.2.2018.08

from the attacker's perspective in terms of hacking context, is defined as a group or a team of white hat hackers that are attacking the organization's informatics systems and whole infrastructure in terms of simulating a full fledge cyber-security attack. From these simulations and attacks, we can note the activities and efforts in trying to successfully evade any antivirus solution that the organization may be placed for its systems.

2 Problem Formulation

There is a big percentage of situations when the team members who are executing a Penetration Testing engagement are faced with the challenge of successfully bypassing an antivirus protection held in-place. These situations can vary from social engineering efforts, fake web-pages and phishing campaigns to planted USB sticks with PE executables or other kind of malicious files.

Most of the time, the malicious files and mail attachments can be in the form of macro-enabled documents designed for exploiting the macro functionalities of the Office suite or .hta files that are executed by the browser. However, there are specific situations when there is a need for a PE executable to be delivered for the target in scope. In this case, certain steps may be required in order to successfully try and evade an antivirus protection. There are certain unwritten rules that an attacker must follow in order to try and evade multiple detection and prevention mechanism that an antivirus solution is presenting. These rules are strongly related to the defense mechanism that an antivirus solution is using in order to successfully identify and remove any potential threat.

2.1 Detection techniques

Among the most common used techniques, we can note two different approaches in which they are grouped by: static analysis and behavior or dynamic analysis.

The main difference between these two groups is that the static analysis is usually made by the antivirus solution on a suspicious file by scanning the file with no actual execution or before the execution takes place. This includes hashing and fingerprinting the potential malware, detecting the obfuscation of packing level a conducting analysis of the portable executable structure owned by the program. Further analysis can take place in the form of analyzing against strings, dynamically linked libraries and functions as well as other disassembler actions.

On the other hand, the behavior analysis takes place when the file is being executed. In that time, the antivirus uses hook functions in order to link with the malware and hijack its execution flow. Among other techniques, the scanning process is observing the outbound and inbound network traffic, any suspicious of powerful Windows API function invoke or other dangerous actions and activities that the executable may imply.

While trying to successfully bypass an antivirus protection with a specially forged malware, an attacker should bear in mind these two defined main techniques when developing his solution. From different experiences, it is fair to say that a lot more effort requires and should be placed in trying to evade dynamic analysis as this type of technique is a more powerful one in terms of detection. [7]

2.2 Evasion techniques

From an attacker's perspective, with all the in mind, a crafted malicious PE file should not only be statically unique every compile time but it should also be able to modify its behavior affectively in order to bypass the dynamic and static scanning of the antivirus solution all together.

Regarding the static analysis, the hash of the executable should always be different from another former version of itself, even if the inputs are the same. In this way, the most common fingerprint technique can be avoided. One thing to look closely when trying to avoid static detection is making a stub being undetected in empty mode. We can define the stub as a program or a sequence of code that processes and finally executes an arbitrary given payload or shellcode. We refer as the empty mode to the state of the stub not being initialized with the future shellcode.

DOI: 10.12948/issn14531305/22.2.2018.08

If the stub is not being detected then further actions should be taken in order to keep the malware undetected. Before inserting the shellcode inside the stub, a proper encryption and obfuscation technique should be made for the payload. This will greatly help in keeping the stub and the whole program under the radar.

On the other side, against the behavior analysis, it is always a good choice to encrypt the outbound connections if the malware is focused on executing a reverse shell. Most antivirus solutions have started to fingerprint different common RAT utilities by fingerprinting the transport layer starting sequence. For example, Windows Defender can successfully identify the Metasploit meterpreter RAT by looking into the first sequence of the transport layer and certificate negotiation. To avoid getting the reverse shell interrupted, another step should be made in terms of configuring the encryption method, more specifically enabling the certificate usage.

Regarding the certificates, another common detection technique is based on fingerprinting the default certificates for common RAT utilities [1]. This can be easily turned in the attacker's favor by changing the certificate with a custom one.

3 Problem Solution

For the implementation of the defined concepts, I used a combination of two programming languages packed in two different components and further combined with one another. The first main component is a python-based script that is taking care of the file manipulation processes along with shellcode injection, manipulation and stub compilation.

For the stub template, I used a C++ program defined as the second part of the utility tool. The stub main purpose is to host and execute the given payload further defined as a sequence of obfuscated shellcode.

The choice for a python main script, among others, was the modularity and portability of the language. The usability of the programming language makes it a good choice for working with cross-compilation, payload preparation and other mechanisms used for the proof of concept project.

On the other hand, the C++ part is great for low level programming, more specifically for having access to memory allocation function, working directly with pointers and overall having a deeper access to Windows API functions.

In order to link both components of the tool, I used a Linux based system that is capable of cross-compiling the C++ stub. For the cross-compiling function, a good choice is *mingw64* program. The main difference between compiling from Linux and compiling directly from Windows host is the code weight and dependencies that Windows is adding to the stub compilation. Of course, the *mingw* program can also be ported to Windows system as well but if Visual Studio is used, for example, it will compile the C++ source code with more dynamic libraries added and thus will make the overall executable more heavy weighted in terms of dependencies and size.

3.1 Stub obfuscation

The first action that should be taken for the goal of the project is to make a strong stub that is capable of staying undetected with or without the payload. This is, in fact, the definition of the malware executable that will be ultimately compiled and executed on the antivirus protected host. First off, the C++ template should contain functions that are DE obfuscating the shellcode, loading the shellcode in-memory and finally executing it. These functions and methods should have a layer of obfuscation. This action can be implemented in different ways. For example, there can be dummy functions that are calling other dummy functions and in the end, will call one of the main functionalities of the stub. Or the obfuscation techniques could allow partial functionalities to be completed and later in another method continued with the execution flow.

Another ground rule that must be taken is to run and load everything related to shellcode in memory. Even if the antivirus solution does have higher local privileges on the machine in

DOI: 10.12948/issn14531305/22.2.2018.08

order to make Windows API hooks, it does not have the necessary rights to bypass kernel protection against reading and modifying another process's memory. If the shellcode or malicious code is touching the disk by any means, it will endanger the potential execution of the program and has a very high chance of detection.

Another step to further obfuscate the C++ source code is to implement encryption and decryption mechanisms for packing and unpacking the shellcode received as input. Finally, the stub should delay the shellcode execution as much as it can, given a relatively decent time. This can be achieved by either adding extra dummy code that is loading random bytes in memory, making dummy operations with files, decrypting and encrypting arbitrary chosen data and other techniques that are trying to mimic the true behavior of the program without actually affecting other processes and make external contact. All of these operations must be chosen carefully in order not to tamper with the actual execution flow of the program and to prevent the original shellcode alteration.

3.2 Python program functionality

As the main utility program of the set-tool, the python acts like a controller. The main responsibility is to obtain the shellcode, encrypt it, pass it to the stub and finally compile the template into an executable file. For this matter, the python script can be made simple and clean and no further obfuscation action should be undertaken. However, the script must contain a strong algorithm that will obtain the shellcode and properly encrypt it. By the time it reaches the stub, the payload should be encrypted and not decipherable. This action will greatly increase in protecting the executable against static analysis.

Another important task is compiling the whole stub with the payload injected into an executable file. This can be achieved by directly calling the *mingw64* program [2] with the help of system commands. The python program is receiving as input from the user the file where the shellcode is placed and the encryption key (currently defined as one length key) that will be used for encrypting the shellcode inside the python program and later injected inside the C++ stub.

3.3 Shellcode generation and listener

The main input received from the user of the evasion utility tool is the shellcode that will be primarily used to execute the user's intentions. The second part of the input is defined as a custom encryption key currently measuring just one key length. The length of the key was chosen in order to provide little encryption effort and actually highlight the overall mechanisms of evasion and not the effort placed in them.

There are a couple of open-source solutions for shellcode generation and some of them even come with a listener that will catch the probably upcoming reverse shell. These solutions can be found in many shapes and sizes. Some of the most popular ones and two of the favorites are the *Metasploit* framework [3] along with its *msfvenom* shellcode generation tool and the *pupy* framework [4] which is actually python based unlike the first example which has ruby as main programming language.

If the user does not want to specifically use any of these open-source tools, he can opt for a custom made implementation of the shellcode generator and handler or use other common tools. For example, there are publicly known websites that are hosting and actively creating shellcodes (for example, *exploit-db* [5]) and the user's task remains to locally run a custom handler by using, for example, the *ncat* binary. [6]

4 Conclusion and future work

As it can be noticed, the efforts for obfuscating the payload and patching the overall malware process and executable are not that high. The project's purpose is to also reveal that with a minimal effort and by respecting some key aspects of the evasion process, a decent undetected executable can be forged.

In other words, the project was focused in trying to highlight key aspects and techniques for successfully making code mutations and

DOI: 10.12948/issn14531305/22.2.2018.08

processes in order to bypass certain antivirus protections. Indeed, there are a lot of improvements that can be added to the developed utility tool, however, its whole purpose is to intentionally use little effort but smart and calculated choices in order to create undetectable malicious executable.

For future improvements, the overall python code and C++ can be greatly enhanced. First, a quick and valuable improvement can be made in the encryption mechanism of the shellcode received as input. Currently, the algorithm used is a simple exclusive or (xor) function that is made via a relatively simple key. The encryption key has a very low entropy and offers very little entropy on the overall executable file however, it resulted that it is a decent choice in matter of hiding the payload. Nonetheless, a good encryption algorithm with a strong key can be used in further encrypting the payload.

Regarding the payload delivery, the obfuscation and junk code can be added for extra behavior analysis protection. Junk code can be created dynamically and decryption methods can be implemented between the dummy codes inserted. In this way, the process of decryption will be less linear and thus results in a lower suspicious rate.

Acknowledgment:

This work is based on my dissertation thesis from Bucharest University Economic Studies, IT&C Security Master program, which has not been traditionally published.

References

[1] *Remote access Trojan*. Available at: https://en.wikipedia.org/wiki/Remote_access_trojan (Accessed: 21 December 2016).

[2] *MinGW Wikipedia page*. Available at: https://en.wikipedia.org/wiki/MinGW (Accessed: 4 January 2017).

[3] *Metasploit framework*. Available at: https://metasploit.help.rapid7.com/docs/msf-overview (Accessed: 17 May 2017).

[4] *pupy framework*. Available at: https://github.com/n1nj4sec/pupy (Accessed: 10 February 2018).

[5] *exploit-db website*. Available at: https://www.exploit-db.com/shellcode/ (Accessed: 17 May 2017).

[6] *ncat binary*. Available at: https://nmap.org/ncat/ (Accessed: 3 October 2016).

[7] Elias Bachaalany and Joxean Koret, *The Antivirus Hacker's Handbook*, Wiley, 2015

DOI: 10.12948/issn14531305/22.2.2018.08

Stefan Sabin NICULA has graduated the Faculty of Economic Cybernetics, Statistics and Informatics in 2016. He holds a master's degree in Information Security from 2018 obtained at the IT&C Security master program. Passionate about mobile application security, binary exploitation, IoT and web application security, presented at the international security conference Defcamp 2017 held in Romania. His research work focuses on the analysis of kernel exploitation techniques, binary analysis and reverse engineering. Other research efforts he made targeted Wireless attacking techniques combined with hardware engineering and malware development process in Red Team engagements.

DOI: 10.12948/issn14531305/22.2.2018.08