International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

Anti-Virus Tempering Methodologies

Dharati Dholariya[1], Dixit Panchal[2], Dr.Pallavi Singhal[3]

[1]Research Scholar - Maulana Azad University, Jodhpur, Rajasthan, India.

[2]Security Researcher, Quick Heal Technologies Pvt.Ltd, India.

[3] Professor, Maulana Azad University, Jodhpur, Rajasthan, India.

Emails: *dharati111@gmail.com[1], panchaldixit877@gmail.com[2], pratap. pallavigmail.com[3]*

Abstract

In this paper, we will discuss antivirus tampering methodologies and various techniques to bypass them, focusing specifically on static and dynamic engine bypasses and their respective methods. The study is centered around antivirus and malware tampering methodologies, with the main objective being to research various aspects of antivirus evasion and bypassing methods. The emphasis lies in understanding how antivirus software operates, exploiting its limitations, and overcoming these restrictions.

Keywords: *Antivirus tempering, Antivirus bypassing, Antivirus evasion methods and techniques.*

1. Introduction

The fundamentals Antivirus software is designed to find and stop the spread of harmful operating system files and processes, protecting the endpoint from running them. Antivirus engines have evolved over time, becoming more intelligent and sophisticated, but most products still use the same basic technology. Some of the still following same methods to detect threat and malware like one-to-one detection method which is totally based on hash value of file. In defence in depth, antivirus software is working on different techniques to identify potential threat and viruses [1-4]. Upon more investigation, we have found out some techniques that antivirus uses to detect malware.

- Static engine (one to one detection)
- Dynamic engine (includes the sandbox engine)
- Heuristic engine
- Unpacking engine

1.1 Methods to Detect Threat / Malware Using Av

One to One Detection (Static Engine Detection): During a scan to identify malware, the antivirus program's static engine compares the current files in the operating system to a database of signatures. Because every change to a malware file might make it reject a particular static signature or even the static engine entirely, static signatures can't truly identify every piece of malware that exists. Figure 1 shown in Static Engine detection

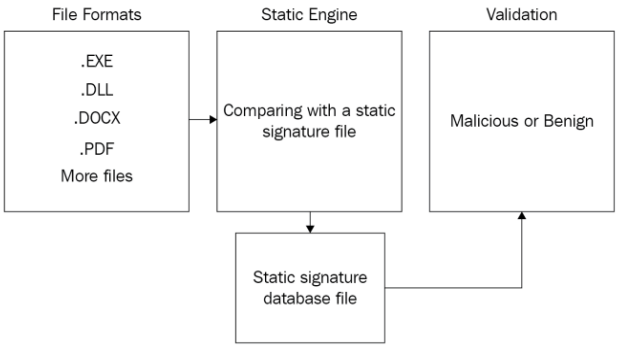


Figure 1 Static Engine detection

Antivirus deals with different file format and with the help of those format it will able to detect it based on static signatures.

- Executable files like .exe, .dll, .msi,.com, .pif, .cpl, .elf , .ocx, .sys, .scr.
- Documents files like .doc, .xls, .ppt, .pdf,.rtf, .chm, .hlp.

These systems look for character strings, file extensions, hash value, some keyword related to viruses that are known to appear in particular malware components in executables and other

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

documents [5]. A file will be identified as harmful if it includes the exact same string as one in the antivirus database; otherwise, it won't. from detailed study we have identified that every day more than 20K+ new malwares are appearing. A static engine detection antivirus needs to have knowledge of every single strain released in order to correctly detect all of these strains, which is a nearly impossible undertaking. Of course, some viruses will slip through.

2. Dynamic Engine Base Detection (Generic detection)

Utilizing a dynamic engine raises the level of sophistication of antivirus software. When malware is actively being used by the system, this type of engine can identify it. The dynamic engine is a little more advanced than the static engine and checks the file in real time using a number of methods. The first method is API monitoring, which seeks to intercept harmful operating system API requests. APIs are tracked by system hooks. A sandbox is a virtual environment that is separate from the memory of the actual host machine. This enables the identification and analysis of malicious software by running malicious software in a virtual environment rather than directly on the memory of the actual machine. Sandboxed malware will be successful against it, especially if it is unsigned and is not recognized by the static engine of the antivirus Programme. One of the big drawbacks of such a sandbox engine is that malware is executed only for a limited time. Security researchers and threat actors can learn what period of time the malware is executing in a sandbox for, suspend the malicious activity for this limited period of time, and only then run its designated malicious functionality. Figure 2 represent Dynamic Engine Detection flow.

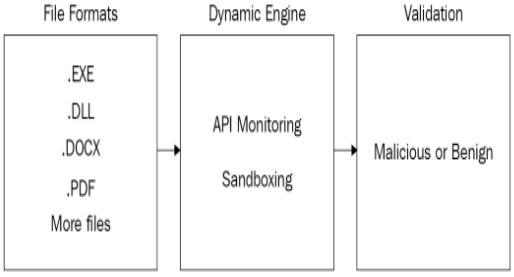


Figure 2 Dynamic Engine Detection

3. Heuristic Engine Detection Method

Using a **Heuristic Engine**, antivirus software becomes even more advanced. With the help of this engine AV can able to detect advance threat using cloud base detection and monitoring the behaviour of source code. This type of engine determines a score for each file by conducting a statistical analysis that combines the static and dynamic engine methodologies. Figure 3 shown in Heuristic Engine Detection Method Formats. Heuristic-based detection is a method, that based on pre-defined behavioural rules, can detect potentially malicious behaviour of running processes. Examples of such rules can be the following:

- If a process tries to interact with the LSASS.exe process that contains users' NTLM hashes, Kerberos tickets, and more.
- If a process that is not signed by a reputable vendor tries to write itself into a persistent location.
- If a process opens a listening port and waits to receive commands from a Command and Control (C2) server.

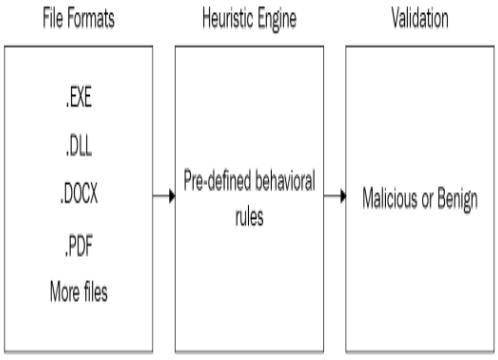


Figure 3 Heuristic Engine Detection Method

In order to prevent alerting on the existence of code that, although being often utilized by malware, is also present in valid software, this sort of method is particularly reliant on having in-depth knowledge of both the contents of genuine programmes as well as the contents of historical malware [6-9]. Because it examines all of the program's contents rather than just the code that would be executed during a single programme invocation, static heuristics, in contrast to dynamic heuristics, have the ability to examine multiple potential programme execution paths.

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

3.1 Antivirus Bypassing Techniques

You have a greater possibility of re-encoding malware to avoid detection by antivirus programmes if you have more possibilities for doing so. Since we want to evade antivirus detection, we must stay away from anything that antivirus software would deem suspect, such as packaged programmes applications with many sections containing executable code.

3.2 Antivirus Bypass Using Obfuscation

Obfuscation is a straightforward method for altering a type of code, including source code and byte code, to make it less understandable. An app developer will employ an obfuscation approach to secure the code and render it unreadable since they do not want unauthorized people to access their code.

Rename obfuscation: With this method, the variable names within the code are primarily obscured. It is challenging to read and comprehend the code using this method, as well as to comprehend variable names and their context inside the code. The variable name after obfuscation might consist of letters like "A," "B," "C," and "D," numerals like 0,1,2 , unprintable characters, and may be it will be converted like 0 - o. Figure 4 Shown as Rename Obfuscation

```
def func(arg1, arg2, arg3='arg3', *, arg4, **kwargs):
    print('arg1', arg1)
    print('arg2', arg2)
    print('arg3', arg3)
    print('arg4', arg4)
    print('kwargs', kwargs)

func('a', 'b', arg3='c', arg4='d')
```

```
def func (000000000000000000 ,000000000000000000 ,000000000000000000 ='arg3',*,arg4 ,**000000000000
    print ('arg1',000000000000000000 )#line:2:print('arg1', arg1)
    print ('arg2',000000000000000000 )#line:3:print('arg2', arg2)
    print ('arg3',000000000000000000 )#line:4:print('arg3', arg3)
    print ('arg4',arg4 )#line:5:print('arg4', arg4)
    print ('kwargs',000000000000000000 )#line:6:print('kwargs', kwargs)
func ('a','b',arg3 ='c',arg4 ='d')#line:8:func('a', 'b', arg3='c', arg4='d')
```

Figure 4 Rename Obfuscation

As you can see we have written simple code like creating function and we have converted it in to 0 and o's formate using rename obfscution technique.

Control Flow Obfuscation: Control-flow obfuscation converts original source code to complicated, unreadable, and unclear code. In other words, control-flow obfuscation turns simple code into spaghetti code!

3.3 AV Bypassing using

Table 1 Program for ("Hello, World!)

#include <stdio.h>
 	
int main() {
printf("Hello, World!\n");
return 0;
	5+tPBV+lF6boJTbKvFPhAHR9Lxw5DyBv6Cg98cYRg9S4bV2Md1cm33ArGyiheJWv9qD6RjUtVRjdB0V8IexJw0keoy9W0KZ3udJr1ZGVzvo=

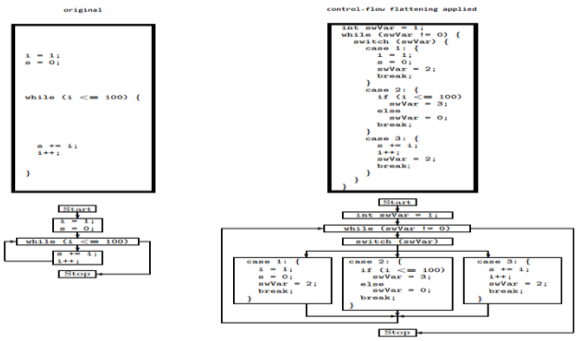


Figure 5 AV Bypassing using

Figure 5 represent the code of AV Bypassing using The encryption of the code, which is one of the most effective methods to hide the source code (Table 1), is one of the simplest ways to employ a bypass. The harmful functionality of the malware might appear to be an innocent piece of code or even completely irrelevant by utilizing encryption, which allows

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

antivirus software to ignore it and allow the malware to successfully infect the system. Malware must first decode its code in runtime memory before it can begin to carry out its harmful behavior. The virus won't be ready to start performing its harmful deeds until it has finished decrypting itself. Figure 6 Shown as Encryption Mechanism

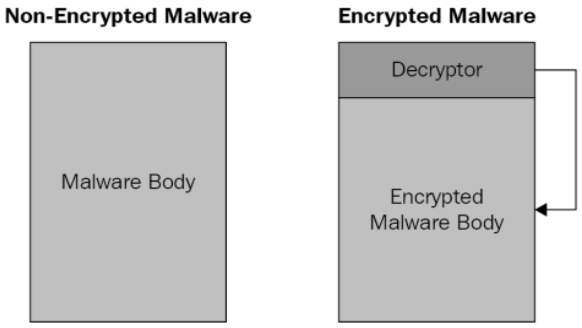


Figure 6 Encryption Mechanism

3.4 Antivirus Evasion using Morphism

Oligomorphic Code: Oligomorphic malware code keeps its basic functionality while exhibiting little change or mutation. Malware writers typically employ this technique to avoid being detected by antivirus software. Traditional antivirus software often uses signature-based detection, which contrasts known malware signatures or patterns against files or processes, to identify harmful software. Malware writers often change their code to produce variations that can evade detection and these signatures. The Oligomorphic Code Source code in below Figure 7 & Flow in Figure 8.

Original Code:

```
function maliciousFunction()
{
    // Malicious actions
    ...
}
function legitimateFunction()
{
    // Legitimate actions
    ...
}
maliciousFunction();
```

Oligomorphic code:

```
function legitFunction() {
    // Legitimate actions
    ...
}

function maliciousCode() {
    // Malicious actions
    ...
}

legitFunction();
```

Figure 7 Oligomorphic Code

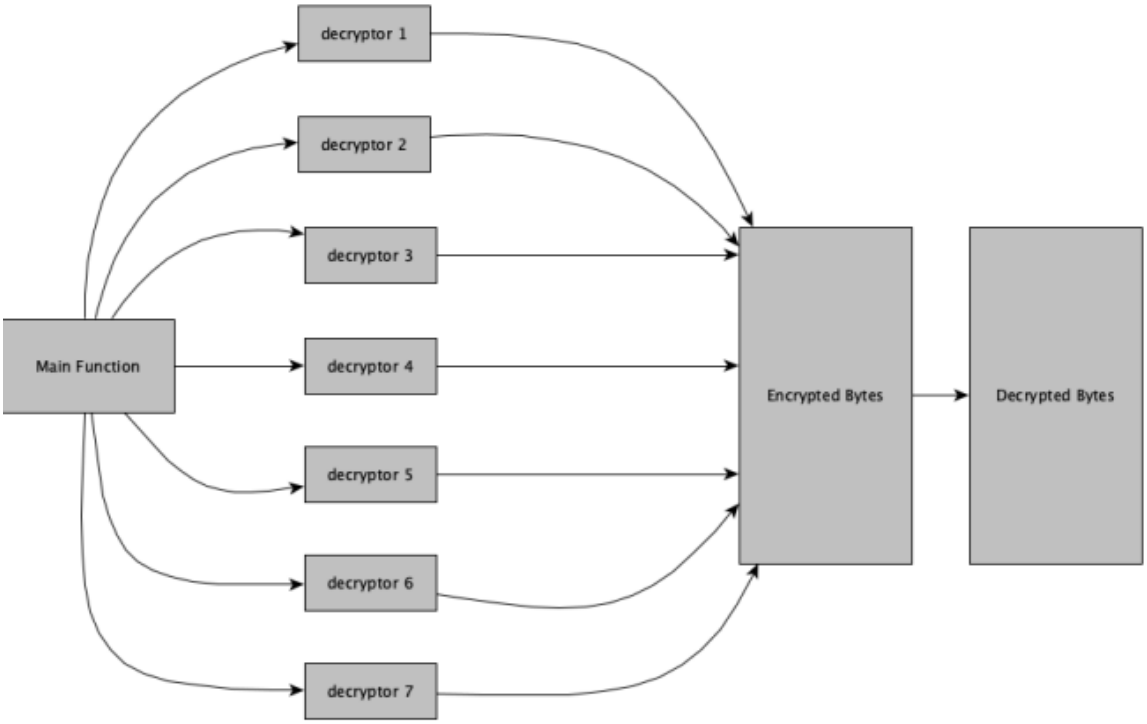


Figure 8 Oligomorphic flow

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

Poliymorphic Code: Malware that is polymorphic typically has a code generator or mutation engine that can produce various iterations of itself. The code is changed in a variety of ways by the generator or engine, including variable names being changed, the sequence of instructions being executed changed, meaningless or junk code being inserted, and portions of the code being encrypted. These modifications result in new malware variants with distinct byte-level signatures from earlier iterations.

Original Code:	Polymorphic Variant 1:	Polymorphic Variant 2:

function maliciousFunction() {
// Malicious actions
...
}
 	
function legitimateFunction() {
// Legitimate actions
...
}
 	
maliciousFunction();
	function _0xE7425C04() {
	// Malicious actions
	...
	}
	
	function _0x4FD3A1B8() {
	// Legitimate actions
	...
	}
	
	_0xE7425C04();
		function _0x982DE24C() {
		// Malicious actions
		...
		}
		
		function _0xF8130D91() {
		// Legitimate actions
		...
		}
		
		_0x982DE24C();

Figure 9 Poliymorphic Code

In this Figure 9 simplified example, the original code contains a malicious Function that performs malicious actions and a legitimate Function that performs legitimate actions. In the first polymorphic variant, the function names are transformed into obfuscated names, such as _0xE7425C04 and _0x4FD3A1B8. The same applies to the function names in the second polymorphic variant, which become _0x982DE24C and _0xF8130D91. Polymorphic Code Flow shown in Figure 10.

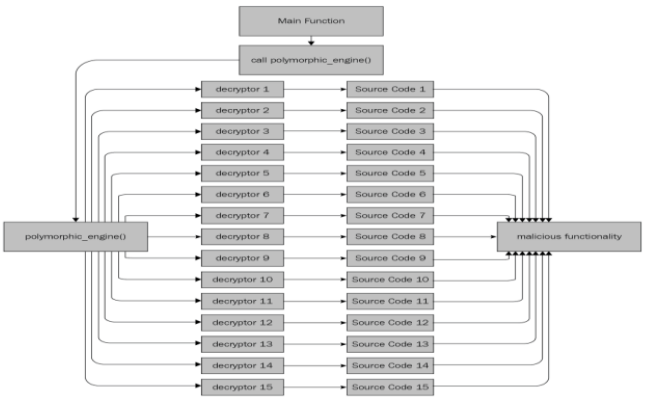


Figure 10 Polymorphic Code Flow

4. Antivirus Tempering Using Packing

To explain how packers work, we will run a simple "Hello World.exe" file through one packer, Ultimate Packer for executables (UPX). general, packers work by taking an EXE file and obfuscating and compressing the code section (".text" section) using a predefined algorithm [10]. The OEP is the entry point that was originally defined as the start of program execution before packing took place. The main goal of antivirus software is to detect which type of packer has been used, unpack the sample using the appropriate techniques for each packer using its unpacking engine, and then classify the unpacked file as either "malicious" or "benign.".

UPX – Ultimate Packer for Executables: This packer is widely used by legitimate software and malware authors alike. First, we will pack our sample Hello World.exe file, and then we will unpack it using the -d argument built into UPX. Finally, we will conduct the unpacking process manually to understand some of the inner workings of this packer. Before we pack the sample, we first put the Hello World.exe executable into a tool called DiE (short for Detect It Easy). The following Figure 11 tells us that the executable has been compiled with C/C++ and that there is no sign of any "protection" mechanism:

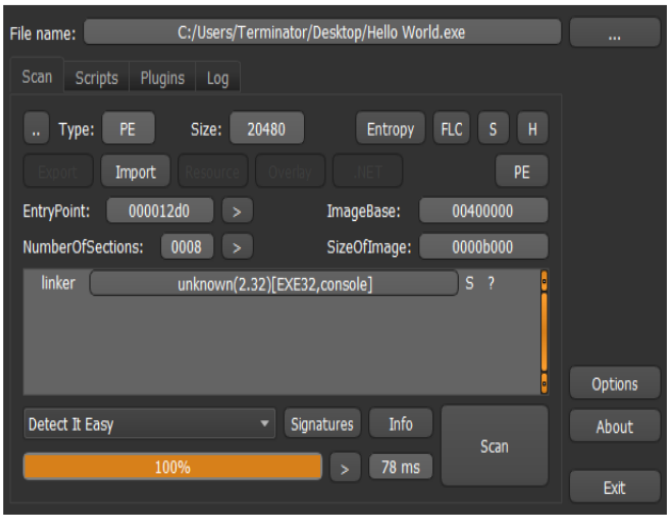


Figure 11 UPX Packer with "Hello World!"

We then check the entropy of the file. Entropy is a measurement of randomness in a given set of values or, in this case, when we check whether the file is packed or not. If the entropy of the any file which is

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

greater than 7 then most probably file is packed. In the following Figure 12 screenshot, we can see that the entropy value is not high (less than 7.0), which tells us that the executable is not packed yet:

Figure 12 Entropy Without Packing File

Another great indicator of a packed file is the function imports that the file includes, which are small compared to a non-packed executable. The following Figure 13 screenshot shows a normal number of imported DLLs and API functions used by the executable using the PE-bear tool.

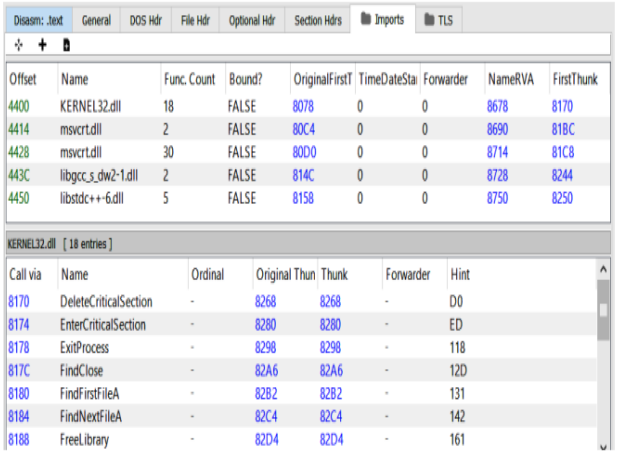


Figure 13 Imported API's

In addition, in the following Figure 14 screenshot, we can see that the entry point (EP) of this program is 0x12D0, which is the address where this executable needs to begin its execution:

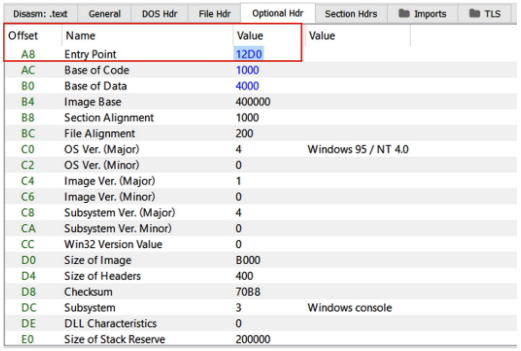


Figure 14 Entry Point for Execution

Now that we understand what a regular file looks like before packing takes place, we can pack the Hello World.exe executable using UPX, with the following command shown in Figure 15.

UPX.exe <File Name> -o <Output file name>

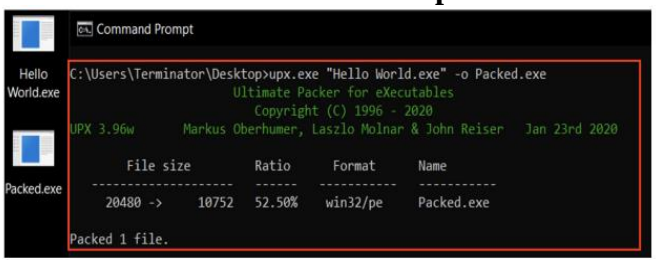


Figure 15 Packing Hello World Exe File Using UPX

Now, testing the packed Hello World.exe executable in the DiE tool reveals very different results, as shown Figure 16.

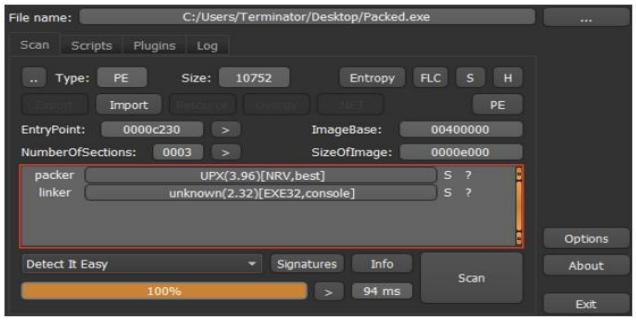


Figure 16 Detected Packer UPX

And as you can see, the executable is successfully detected as a UPX-packed binary. The entropy and the section names support this conclusion, as seen in the following Figure 17 screenshot.

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

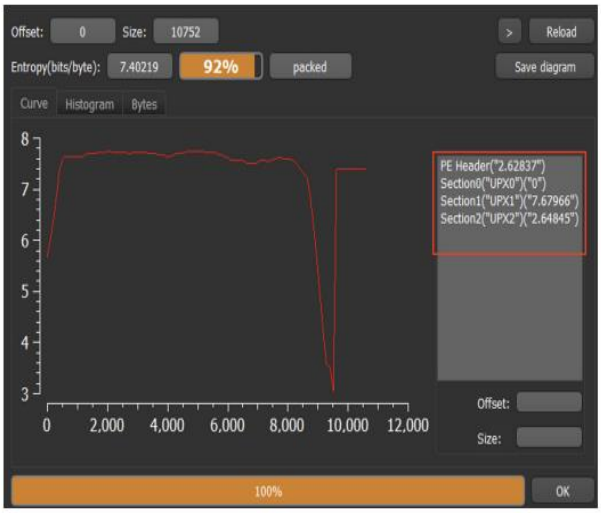
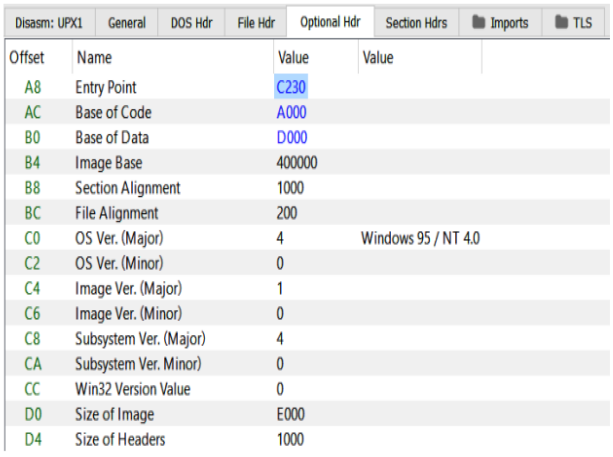


Figure 17 Entropy of Packed File

In addition, using the PE-bear tool again, we can see here that the entry point of this packed version of Hello World.exe has also been changed to 0xC230:



Disasm: UPX1 | General | DOS Hdr | File Hdr | Optional Hdr | Section Hdrs | Imports | TLS

Offset	Name	Value	Value
A8	Entry Point	C230	
AC	Base of Code	A000	
B0	Base of Data	D000	
B4	Image Base	400000	
B8	Section Alignment	1000	
BC	File Alignment	200	
C0	OS Ver. (Major)	4	Windows 95 / NT 4.0
C2	OS Ver. (Minor)	0	
C4	Image Ver. (Major)	1	
C6	Image Ver. (Minor)	0	
C8	Subsystem Ver. (Major)	4	
CA	Subsystem Ver. Minor)	0	
CC	Win32 Version Value	0	
D0	Size of Image	E000	
D4	Size of Headers	1000	

Figure 18 Entry Point Changed After Packing

As Conclusion you can see that after Packing the file using UPX packer we get some of the indicators that file is packed using entry point and some tools like PE-Detective, PE-Bear, EXEinfo etc using these tools you can easily identified packed files. Figure 18 shown as Entry Point Changed After Packing

4.1 Some Defences Against Potential Threats

Run never in administrator mode. It is a golden rule that may prevent 99% of viruses without an AV. Utilise the very robust security capabilities in Windows 10 to harden your computers. Spend money on network intrusion detection systems and keep an eye on it. Malware infestations are frequently not found on the victim's PC, although odd NIDS or firewall logs might help. Utilise a number of AV equipment from various suppliers. One product can make up for another's flaws.

Conclusion

In this paper I have Determined the different AV Evasion techniques to evade any antivirus software and how to prevent this evading throughout the paper. the techniques of finding loop wholes in antivirus software. [11] Bypassing antivirus is simple when you exploit their weakness .it requires some knowledge of windows system internals and required deep understanding of how AV software can detect threat. Antivirus is useful in detecting millions of wild Threats which are already in database, also they are useful for system recovery.

References

[1]. Samociuk, D. (2023). Antivirus Evasion Methods in Modern Operating Systems. Applied Sciences, 13(8), 5083. https://doi.org/10.3390/app13085083

[2]. 91-95. (n.d.). Scribd. https://www.scribd.com/document/629597533/91-95#

[3]. Devglan. https://www.devglan.com/online-tools/text-encryption-decryption

[4]. C/C++ Obfuscator - Obfuscate your C/C++ source code for free and online. (n.d.). https://picheta.me/obfuscator

[5]. What is a “control-flow flattening” obfuscation technique? (n.d.). Reverse Engineering Stack Exchange. https://reverseengineering.stackexchange.com/questions/2221/what-is-a-control-flow-flattening-obfuscation-technique.

[6]. Citu, A. (2023, January 23). Book Review: Antivirus Bypass Techniques. Adventures in the Programming Jungle. https://adriancitu.com/2023/01/18/book-review-antivirus-bypass-techniques/

[7]. JoelGMSec. (n.d.). GitHub - JoelGMSec/Invoke-Stealth: Simple & Powerful PowerShell Script Obfuscator.

International Research Journal on Advanced Engineering and Management
https://goldncloudpublications.com
https://doi.org/10.47392/IRJAEM.2024.0166
e ISSN: 2584-2854
Volume: 02
Issue: 04 April 2024
Page No: 1232-1239

GitHub. https://github.com/JoelGMSec/Invoke-Stealth

[8]. Pérez-Sánchez, A. M., & Palacios, R. (2022). Evaluation of Local Security Event Management System vs. Standard Antivirus Software. Applied Sciences, 12(3), 1076. https://doi.org/10.3390/app12031076

[9]. DevGlan. (n.d.-b). Encrypt and Decrypt Text Online. Devglan. https://www.devglan.com/online-tools/text-encryption-decryption

[10]. Dan.com. (n.d.). antivirusware.com - Domain Name For Sale | Dan.com. https://antivirusware.com/

[11]. Grant, A. (2023, June 14). Help Net Security - Cybersecurity News. Help Net Security. https://www.helpnetsecurity.com/