DOI: https://doi.org/10.52651/sam.a.2022.1.14-22

EVASION OF ANTIVIRUS WITH THE HELP OF PACKERS

Andrej FEDÁK, Jozef ŠTULRAJTER

Abstract: Nowadays, almost every malware file comes obfuscated and prepacked preferably with an unknown algorithm. Antivirus programs are taught to deal with these kinds of obstacles with the help of signature databases and heuristic engines. AV systems and their tools are professionally and carefully developed by experts; however, they are not flawless either. They tend to react to any threats that are identified by already-known malicious patterns and bad behaviours. Therefore, malware has to evolve and use new methods to pass these defences. In this paper, the internal components of AV programs and well-known packing techniques are briefly explained while in addition they are tested against each other. This work provides an initial insight into the complex subject of antivirus protection.

Keywords: Antivirus detection; Malware evasion; Scanner; Signature; Heuristic engine; Packer; Obfuscation; Compressor; Crypter; Protector; Portable executable.

1 INTRODUCTION

First things first, in order to evade antivirus protection, researchers or attackers must know how an antivirus product works. Therefore, the aim of this work is to describe the basic components that create the whole structure of the AV program. Individual AV components are introduced together with their internal processes that help in the detection and removal of malicious files. As the attack patterns become more sophisticated, antivirus engines have to adapt and improve their capabilities to identify new potential threats. Malware authors always try to be one step ahead of the competition and develop new methods to mask their activities and hide any traces of their malicious code. One of the many obfuscation techniques is the use of packer programs that are capable to create a new protection layer around the bad executable file. Dozens, nay hundreds of unique packer programs are in circulation and many more to come. In this paper, the effectiveness of the well-known packers is tested as well as some of their shielding features are explained.

2 DEFINITION OF ANTIVIRUS SOFTWARE

Antivirus software is special security software created for the purpose to protect your computer and prevent computer infections by detecting malware. In the vast majority of cases, it is used as a preventive measure. Even such specified software solutions are not perfect and in the case of an uncaught intrusion and infection, they are furthermore designed to completely remove the malicious software and disinfect the computer [1].

Nowadays, several security features come usually built-in within the operating systems such as Windows (Defender) or Mac OS X (XProtect). There is still a vast number of companies (for example Bitdefender, Norton, Kaspersky, Avast, ESET, and many others) that solely focus all of their resources to create special security software that aims to give better protection than that offered by the operating system.

The main feature of AV software is to find known malicious behaviours and patterns in programs, documents, web pages, or network packets. The detection capabilities of AV products are primarily based on experience with previously known malware patterns. Simply, an AV software is not able to identify new unknown threats unless they are based on old known behavioural or static patterns [1].

3 STRUCTURE OF ANTIVIRUS SOFTWARE

The main part of an AV system is called the core or the kernel, which coordinates tasks between all the other components such as the scanning engine (command-line scanner, GUI scanner), daemons or system services, file system filter drivers, network filter drivers, plugins, kernel AV components (signature database, decompressors, emulators, supported file formats). The AV product suite may also include other additional support utilities like browsers, browser toolbars, drivers for self-protection, firewalls, and so on. As you can see, the product is the whole software package the AV company ships to the customer [1].

3.1 Kernel

A kernel forms the core of an AV product. All the routines for unpacking executable programs, compressors, crypters, protectors, and so on are in the kernel's libraries. Hence the kernel must support all the code for opening a very long list of file formats in order to iterate through all the streams in a file, analyse them, and catch malicious exploits embedded in the files. Some file formats (excluding compressors and archives) that need to be supported are OLE2 containers (Word or Excel documents); HTML pages, XML documents, and PDF files; CHM help files; PE, ELF, and MachO executables; JPG, PNG, GIF, and TIFF image file formats; ICO and CUR icon formats; MP3, MP4, AVI, and MOV video and audio

file formats; and so on. Furthermore, the kernel is frequently used by the scanner engine, by the AV resident (or daemon), or by other programs and libraries. Developing an AV kernel is very complex because enormous time and effort are required to support mentioned features [1].

3.2 Scanners

Another common feature of AV products is the scanner, which may be a GUI or command-line on-demand scanner. Such tools are used to scan whenever the user decides to check a set of files, directories, or the system's memory. There are also on-access scanners, more typically called residents or real-time scanners. The resident analyses files that are accessed, created, modified, or executed by the operating system or other programs (like web browsers). It does this to prevent the infection of document and program files or to prevent known malware files from executing. However, the resident is one of the most interesting components to attack. For example, a security bug in the parser of Microsoft Word documents can expose the resident to the execution of a malicious code after a Word document is downloaded (even if the user doesn't open the file). Or a similar approach can be applied to the AV's parser code handling new email messages and their attachments. These bugs can be used to perform a denial-of-service attack on an AV program, which makes it crash or loop forever, thus disarming the antivirus temporarily or permanently [1].

3.3 Signatures

The scanner of any AV product searches files or packets using a set of signatures to determine if the files are malicious. The signatures are the known patterns of malicious files. Some typical signatures are based on the simplest pattern-matching techniques (searching for a specific string, or byte-stream), Cyclic Redundancy Check algorithms (error-detection code that calculates output hash in form of CRC checksums), or MD5 and SHA1 hashes. Relying on cryptographic hashes, like MD5, works only for a specific file (as a cryptographic hash tries to identify just that one whole file), while other fuzzy logic-based signatures, as CRC algorithm applied on specific parts of data, can identify various bad files. AV products usually have different types of signatures which range from simple CRCs to rather complex heuristics patterns based on many PE header properties, the complexity of the code at the entry point of the executable file, the entropy of a section or the whole executable file, and so on [1].

Each kind of signature has advantages and disadvantages. For example, some signatures are very specific and less likely to be prone to a false positive (when a healthy file is flagged as malware) – cryptographic hashes, while others are very risky and can generate a large list of false positives – CRCs. For example, imagine a signature that finds the word "Microsoft" anywhere in a file. This would cause a large list of false positives, regardless of whether it was discovered in malware. Stricter pattern description avoids any false positive detections [1].

3.4 Decompressors and unpackers

Another key part of every AV kernel is the support for compressed or archived file formats like ZIP, TGZ, 7z, RAR, XAR, and so on. AVs must be able to decompress and navigate through all the files inside any compressed or archived file, as well as compressed streams in PDF files and other file formats. Because AV kernels must support so many different file formats, vulnerabilities are often found in the code that deals with this variety of input [1].

An unpacker is a routine or set of routines developed for unpacking protected or compressed executable files. Malware in the form of executables is commonly packed using freely available compressors and protectors or proprietary packers. Some packer tools, like UPX (Ultimate Packer for Executables), just apply simple compression, and unpacking such samples is an easy and straightforward matter. On the other hand, more complex software packers and protectors may in addition transform the code into bytecode and run it with its own virtual machine. Some packers can be unpacked using the CPU emulator of the AV, another by static means, and the rest, more complex ones, using both techniques. The emulator is used up to some specific layer and then a static routine executes when some specific values are known such as the size of the encrypted data, the algorithm used, the key, and so on [1].

As with compressors and archives, unpackers are a very common area to explore when you are looking for vulnerabilities in AV software. The list of packers to be supported is immense, even larger than the number of compressors and archives, and it is still growing. Some of them are used only during a specific malware campaign, so there is ever-growing emergence of new packers hiding the logic of new malware [1].

3.5 Emulators

Most AV cores on the market offer support for a number of emulators such as the most common Intel x86 emulator, AMD64, or ARM emulators. Emulators are not limited to regular CPUs. There are also emulators for some virtual machines that are aimed at inspecting Java bytecode, Android DEX bytecode, JavaScript, and even VBScript or Adobe ActionScript. Usually, files that trigger the emulators are EXE crypters or packers that are too complex to decrypt statically, so the antivirus engineers decided to decrypt them using the emulator.

Nowadays, fingerprinting or bypassing emulators and VMs used in AV products is very common and quite an easy procedure. It is almost impossible that the developers of the AV emulators would implement all of the instructions supported by to-be-emulated real CPUs. For executable ELF or PE files, it is even less likely that the developers would implement the whole operating system environment. Therefore, it is really easy to discover many different ways to fool emulators and to fingerprint them [1].

3.6 Heuristics engines

Another common component in antivirus software that detects malicious code is the heuristic engine. The AV heuristic engines make decisions based on general evidence instead of universal detections or typical signature-based methods. They rely on detection procedures that assess evidence and behaviour as collected from analysing the code statically or dynamically. On the other hand, they do not rely on specific signatures to try to catch a certain family of malware or malware that shares similar properties. Heuristic engines implement a set of algorithms that emulate the decision-making strategy of a human analyst [1].

There are three different types of heuristic engines namely static, dynamic, and hybrid, which uses both strategies. Most often, static heuristic engines are considered true heuristic engines, while dynamic heuristic engines are called Host Intrusion Prevention Systems (HIPS). Static heuristic engines try to discover malicious software by finding evidence statically by disassembling or carefully analysing the file headers. Dynamic heuristic engines try to assess the file or program based on its behaviour by hooking (intercepting) API calls or executing the program in an emulated environment. Learning about various heuristic engines can get some insights into how attackers are evading AV detection [1].

3.7 Static heuristic engine

Static heuristic engines are implemented in many different ways depending on the deployment target. For example, it is common to use heuristic engines that are based on machine learning algorithms (such as Bayesian networks, genetic algorithms, or expert systems) to reveal information about similarities between families by focusing on the biggest malware clusters created by the heuristic engines. Those heuristic engines are deployed and acceptable only in malware research labs because they can cause a large number of false positives and consume a lot of resources. For desktop antivirus products, a much better choice is an expert system that implements a set of algorithms simulating the decision-making process of a human analyst [1].

A human malware analyst can determine that an executable file appears malicious, without actually observing its behaviour, by briefly analysing the file structure and taking a quick look at its disassembled code. The analyst would evaluate several indicators as a whole before labeling the file as a malicious one. Some of the suspicious features could be an uncommon file structure, uncommon characteristics in a PE header, the obfuscated code, compressed or somehow protected program, file packed multiple times, corrupted file, any anti-debugging tricks, change in the icon of the PE file to the different one (used for image files, documents, etc.), dual extension (common in malware that disguises an executable file as a video, picture, document, ZIP file, or other types), and so on. If some of the mentioned features are true, a human analyst would suspect that the file is malicious or at least that it is trying to hide its logic and needs to be closely analysed. He would also compare that sample with some sort of list of known false positives. Such human-like behaviour, when implemented in a heuristic engine, is called an expert system [1].

3.8 Dynamic heuristic engine

Another analytical technique is known as dynamic heuristics. When researchers want to analyse a suspicious code without endangering running systems, they contain the sample in a controlled environment (like a secure lab) and perform a variety of tests. Like this, it isolates the program or piece of code inside a specialized virtual machine or sandbox and gives the AV program a chance to test the code and simulate what would happen if the suspicious file was allowed to run. It examines each command that's executed and looks for any suspicious behaviours, such as self-replication, overwriting files and registry entries, and other actions that are common to malware [2].

Dynamic heuristic engines base their detections on the behaviour of the file or program by hooking API calls or executing the program under an emulation framework. The former approach is more reliable because it involves actually looking at the true runtime behaviour, while the latter is more error-prone because it largely depends on the quality of the CPU emulator engine and the quality of the emulated operating system APIs. It is quite an easy task to bypass heuristic engines based on emulators and virtual execution environments. Malware may execute a code that is not fully supported by the emulators to fingerprint the AV software and change its own behaviour accordingly with the intention of avoiding detection. Bypassing heuristic engines based on hooks, like the typical Host Intrusion Prevention Systems (HIPS), is not complex either and depends on which layer the API hooks are installed (userland or kernel-land hooks) in order to monitor the behaviour of a program [1].

Userland hooks work by detouring or intercepting some APIs to monitor and control the execution of

those APIs. To bypass userland hooks, attackers could read the original prologue of the hooked functions from the disk, execute those bytes, and afterward continue executing the not-hooked part of the function past the prologue bytes. Another simple approach is to unload the hooking library, which will subsequently remove the respective hooks. Kernel-land hooks rely on registering call-backs that monitor the creation of processes and access to the system registry and monitoring real-time file activity. Similarly, kernel-land hooks might be bypassed and uninstalled by malicious code running in the kernel [1].

4 MALWARE DETECTION WITH VIRUSTOTAL

VirusTotal is an online service that allows you to upload a file, which will be subsequently inspected with over 70 antivirus scanners. It can be useful in detecting malicious content and also in identifying false positives (harmless items detected as malicious by one or more scanners). Upon submitting a file, scanning reports are shared with the submitter, and also the public VirusTotal community. As a result, the contributors are raising the global IT security level and helping cybersecurity professionals and security product developers discover harmful files samples for further study, analyse emerging cyber threats, and create new defences [3].

VirusTotal's aggregated data is the output of many different antivirus heuristic engines, known-bad signatures, website scanners, metadata extraction, file and URL analysis tools, many user contributions, etc. Since the end of 2017, it is also integrating a malware analysis system in order to contribute behavioural analysis reports. Thus, its tools are able to comprehensively analyse samples from both static information and dynamic behaviours, trigger and capture behaviours of the samples in the sandbox, and output the results in various formats [3],[4].

Sometimes, the main advantage of using VirusTotal could be also its drawback since all the uploaded files automatically become public. This is not productive if you are researching AV evasion techniques or when doing penetration testing. In the first case, malware creators are also searching through public databases to find out if their malware has already been discovered. If so, they could alter the behaviour of malicious samples or stop using some of its services to hide any tracks. In the second situation, using VirusTotal can be a bad idea if you want to keep your testing payloads private to ensure they evade antivirus products for a longer period of time. Therefore, you need to use a private VirusTotal-like tool and this is where your own offline MultiAV solutions come into play [1].

The actual usefulness of virus scanners to discover new threats is being disputed, but they are able to detect *well-known* threats quite well. In this work, we are mainly interested in the changes in the detection results after applying wrappers on the malicious files. Hence the output provided by VirusTotal is more than sufficient.

5 MALWARE EVASION WITH PACKERS

AV software uses various techniques to identify malicious software, which often self-protects. Today's malware may use many obscure techniques in order to persist by staying hidden during infection and operation and to prevent detection, analysis, and removal. Malware achieves this by adding code that is not strictly malicious but only intended to hide the malicious code in an operating system (see the visualization in Figure 1).

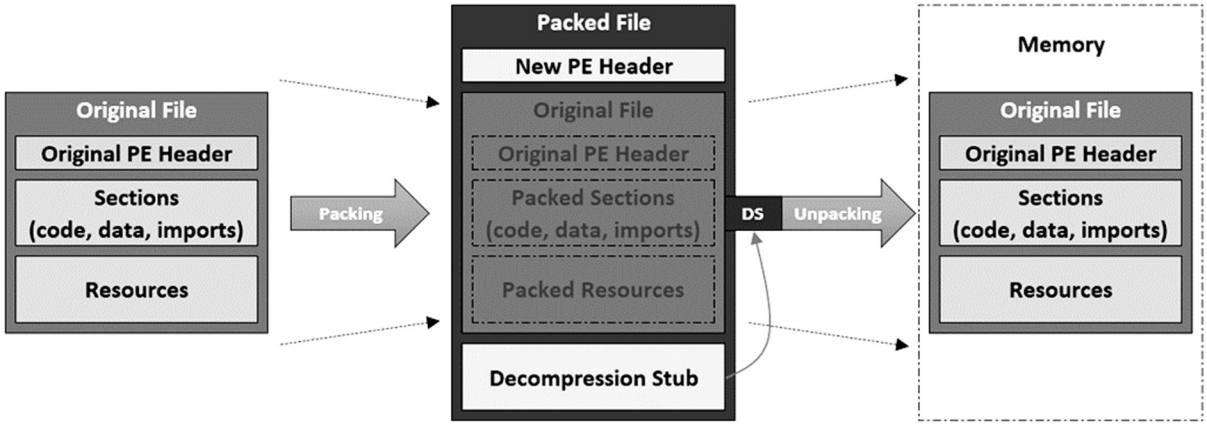

Fig. 1 Life cycle of packed PE (portable executable) file
Source: authors.

According to the used layer of protection, the obfuscation techniques can be divided into three main categories: packers, crypters, and protectors. Definitions for these categories are not carved in stone, differences between them are sometimes blurred, they all have overlap and there are exceptions to the rules [1], [5], [6].

ROT5

Plaintext (ASCII)	R	o	T	a	T	e	M	E
Plaintext (8-bit)	01010010	01101111	01010100	01100001	01010100	01100101	01001101	01000101
Add 5 (8-bit)	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101
Ciphertext (8-bit)	01010111	01110100	01011001	01100110	01011001	01101010	01010010	01001010
Ciphertext (ASCII)	W	t	Y	f	Y	j	R	J

XOR operation

Plaintext (ACSII)	h	e	l	l	o	m	a	n	A	B	A ⊕ B
Plaintext (8-bit)	01101000	01100101	01101100	01101100	01101111	01101101	01100001	01101110	0	0	0
XOR key (8-bit)	01010011	01010011	01010011	01010011	01010011	01010011	01010011	01010011	0	1	1
Ciphertext (8-bit)	00111011	00110110	00111111	00111111	00111100	00111110	00110010	00111101	1	0	1
Ciphertext (ASCII)	;	6	?	?	<	>	2	=	1	1	0

Base64 encoding

Plaintext (ASCII)	S								U								B							
Plaintext (8-bit)	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	0
Ciphertext (6-bit)	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1	0
Ciphertext (Base64)	U						1						V				C							

Fig. 2 Trivial examples of encryption techniques
Source: authors.

5.1 Compressors, original packers

In a lot of cases, the entire malware program is obfuscated using what's known as a packer program (for example UPX, PESpin, MPRESS, ASPack, even WinRAR, and dozens of others). Simply, a runtime packer compresses the original malware file, thus making all the original code and data unreadable. This software prevents anybody from directly viewing the malware's code until it decompresses itself at runtime in the memory where the "packed file" is executed thus revealing the program's original code. Sometimes this technique is also known as "self-extracting archives" or "executable compression" [5], [6].

In the past, this type of compression has been used for legitimate purposes, some of which include protecting against piracy and making executable files smaller because of the then size of portable media and internet speeds. Nowadays, this application became unnecessary, so when you see some packers being used, it is almost always for malicious purposes. They help conceal vital program components to prevent less-experienced reverse engineers from unpacking the malware's contents. The creation of new custom packers defeats modern unpacking scripts and forces reversers to manually unpack the file. Sometimes malware authors will pack their files two times, with a commercial packer and then with their own custom solution [5], [6].

Fortunately, there are many programs available that identify commercial packers, and also advise on how to unpack these files. Some of the file scanners are for example Exeinfo PE, PEID, Detect-It-Easy, or any signature-based database checker [6].

5.2 Crypters

The crudest technique utilized by malware authors to hide malware's internals is called obfuscation which can be commonly seen in scripts. Obfuscation is a technique that at first sight makes binary and textual data (for example malicious URLs or registry keys) unreadable and hard to understand. Its implementation can be as simple as a few bit manipulations and advanced as cryptographic standards (i.e. DES, AES, etc). Thus, a more complex method is actual encryption. A crypter is a type of software that can encrypt, obfuscate, and manipulate malware, to make the hidden executable as hard to detect by security programs as possible [5], [6].

Perhaps the simplest technique is **ROT** which is an ASM instruction for "rotate", hence, for example ROT13 would mean "rotate 13". ROT13 uses simple letter substitution to achieve obfuscated output. The **XOR** operation is probably the most common method of obfuscation. With the simple XOR cipher, a string of text can be encrypted by applying the bitwise XOR operator to every character using a given key. Even without the XOR key, decryption programs are able to cycle through every possible single-byte XOR value in search of a particular string (i.e. "MZ" or "PE"). To make the obfuscation more bulletproof, malware authors might implement a two-cycle

approach (performing two XOR encryptions with different values) or increment the XOR, ROT value in a loop. Furthermore, **Base64** encoding has been used for a long time to transfer binary data (machine code) over a system that only handles text. Its encoding alphabet is commonly used in malware to disguise text strings. Because Base64 encoding is typically easy to identify by its padding character (equal sign "=") and then overcome, malware authors may adjust the order of the alphabet, which breaks standard Base64 decoders. The basic principles of aforementioned cryptographic techniques are illustrated in the Figure 2 [6].

5.3 Protectors

A protector (for example Enigma, Themida, VMProtect, and so on) is software created to keep an attacker from directly inspecting or modifying a compiled application to change its behaviour. It could be described as a shield that keeps an application encrypted and protected against reverse engineering (refer to Figure 3). The obfuscation techniques used by the protectors usually include the best of both packing and encrypting (hybrid). That combination together with some added features builds several protective layers around the payload that a researcher has to face. For example, when a protected application is going to be run, the software protector will first check for possible cracking tools (dissemblers or de-compilers) that may be running on the operating system. If everything is safe the software protector will then proceed to decrypt the protected application and allow it to be executed [5], [7].

Another approach of protectors is code virtualization, which uses a customized and different virtual instruction set every time you use it to protect your application. Such professional protectors are used in the gaming industry against piracy, yet this technique has also made its way into malware, more specifically ransomware. The protection is so efficient that there is no need for the encryption key to be obtained from the command-and-control server, but it can be hardcoded right into the ransomware. Unpacking samples protected by a virtualization packer could be highly time-consuming and sometimes even impossible for researchers to restore the sample into its original code. Detection of these packed samples is extremely difficult with traditional AV unpacking technology [5].

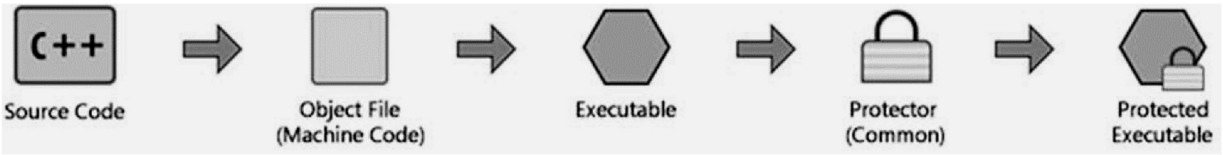

Fig. 3 Philosophy of common software protectors
Source: [7].

The main advantage of using a software protector is to protect an application (in our case, malware) against piracy and reverse engineering, however, that doesn't mean the protected application is unbreakable. That's because software protection is very different from data protection. Even if a software protector encrypts the protected application with the most robust cryptographic algorithm (like RSA, Elliptic curves, and AES), sooner or later the protected application needs to be decrypted part by part in order to be run by the CPU. It is in this phase that most attackers will start their work by dumping the decrypted application from memory to disk thus not having to deal with the cryptographic algorithm and reconstruction of the original application [7].

6 THE EFFECTIVENESS OF COMMON PACKERS

Using obfuscation of any kind can be beneficial for the reuse and recycling of old malware solutions. In this exercise, we will be presented with the results of how effective the usage of common packers in evading AV detection can be. Static evasion techniques are achieved by modifying the contents of the input file, with the help of packer programs, so its hash or checksum is changed and can no longer be detected using signature-based detections. Packed malicious executable files, preferably well-known ones, will face dozens of scanning engines provided by VirusTotal online service. The work of signature checkers and their databases together with static heuristic engines will be adequately tested. However, in real-life situations, there is a high chance that the packed malware will be discovered by the dynamic heuristic engines, therefore it would need more anti-AV modifications. In the conducted small-scale exercise (see Figure 4), it is noticeable that packers have an impact on the AV detection rates (for example 60/69 means how many AV engines found the file malicious out of the whole AV software pool). In this case, statistical deviation could be quite large, since rather a smaller pool of samples was part of the experiment, but it is adequate for illustrative demonstration. Still, the differences in detection capabilities could be seen, even though well-known

packer programs and malware files have been tested against the latest and time-tested signatures and static heuristic engines. Perhaps, the results would be slightly different if other than basic settings of the packers were applied. As only free trial and demo versions of software products have been tried out, some strong security features were not accessible. Almost every packer program offers dozens of options which as a result would produce unique output files each time the different option is checked. In this case, when ten software packers are used and each possible feature would be tested, there might be more than thousands of different outputs. Trying this would be hugely time-consuming and the results might be probably better, but not that significantly different. It also needs to be mentioned that the malware files were already obfuscated with several techniques including packing which was probably used even more times.

A few interesting things could be observed in Figure 4. Harmless Python executable file "py.exe" was marked as a highly suspicious file or even malware by several AV products after the packers were applied. Sometimes packed malware and regular software were evaluated as malicious by the same or similar amount of AV engines. On this note, many of the AV heuristic engines detected suspicious patterns in packed software not because of the malicious internals, but because of the unusual obfuscation and protection layer provided by tested compressors and protectors. Therefore, excessive protection measures may trigger an alarm of some AV products even if the software authors have good intentions, which in the end may prove counter-productive in practical terms. Such software solution would need to be included in the AV white list.

Samples	Original file	Compressors					Crypters / Protectors				
		MEW v1.2 [8]	MPRESS v2.19 [9]	PElite v2.4 [10]	UPX v3.94 [11]	WinRAR SFX v6.10 [12]	Enigma v7.00 [13]	Obsidium v1.7.4 [14]	PELock v2.11 [15]	Themida v3.1.1 [7]	VMProtect v3.5.1 [16]
py.exe	0/68	17/66	8/69	23/69	2/67	3/66	16/69	13/67	24/68	18/69	12/69
	0,0%	25,8%	11,6%	33,3%	3,0%	4,5%	23,2%	19,4%	35,3%	26,1%	17,4%
Artemis	29/68	23/67	14/69	15/69	ERR	1/67	18/67	19/69	11/69	22/69	12/67
	42,6%	34,3%	20,3%	21,7%	ERR	1,5%	26,9%	27,5%	15,9%	31,9%	17,9%
CryptoLocker	57/66	39/66	31/69	36/67	49/71	13/66	16/69	16/69	30/69	26/69	20/69
	86,4%	59,1%	44,9%	53,7%	69,0%	19,7%	23,2%	23,2%	43,5%	37,7%	29,0%
Kovter	60/69	42/69	ERR	36/69	ERR	12/66	21/70	14/69	26/69	26/68	30/68
	87,0%	60,9%	ERR	52,2%	ERR	18,2%	30,0%	20,3%	37,7%	38,2%	44,1%
Slammer	31/68	31/68	ERR	32/69	22/69	10/67	20/69	21/68	28/69	29/70	28/66
	45,6%	45,6%	ERR	46,4%	31,9%	14,9%	29,0%	30,9%	40,6%	41,4%	42,4%
WannaCry	64/70	45/69	ERR	40/70	ERR	10/67	36/70	30/69	36/69	33/69	30/66
	91,4%	65,2%	ERR	57,1%	ERR	14,9%	51,4%	43,5%	52,2%	47,8%	45,5%
Improvement	/	22,0%	50,2%	32,5%	25,1%	80,7%	51,2%	54,0%	44,6%	38,9%	46,2%

Fig. 4 Impact of packers on AV detection rates (tested by using VirusTotal online service)
Source: authors.

Another interesting result was achieved with the use of WinRAR compression, encryption and SFX features when the only common AV desktop solution, that caught compressed malware, was Bitdefender. To explain the process of obfuscation, the malware file was firstly compressed and encrypted with a password. Then the encrypted malware file together with a decryption script were the main parts of the executable SFX (self-extracting) archive. When the SFX archive was executed, it was set to immediately run the decryption script and afterward the decrypted malware. However, this process would be most likely intercepted by AV software at a time when the known malware file reveals itself in the memory. Despite that unfair approach, it is a demonstration of how unusually regular software can be misused for a bad purpose.

7 CONCLUSION

Even governments participate in writing malware in the form of spying on rebels or sabotaging other countries' infrastructures to protect their own interests. Whatever someone's intentions are, malware authors use several different techniques to achieve the ultimate goal which is being undetectable by any security vendor also known as FUD (Fully Undetectable). The first step is usually to encrypt malware with a strong and resilient protector (preferably with the perpetrator's unknown software solution). Then, malware authors will privately scan hundreds of unique copies of their malware with multiple AV security products (similar to VirusTotal) and choose only copies that can bypass all of them. And finally, they use zero-day exploits and cyber-attack techniques to increase the chance of a successful infection [17].

Signatures checkers and static heuristic engines are sometimes prone to mark a good file as malware. For example, when one or two VirusTotal scanning engines out of 70 identify suspicious files as a threat and dynamic analysis doesn't sound the alarm, it is most likely a false positive case. If an experienced user is sure that the file is false positive, packers could be quite helpful when you use it as a form of hiding the false positive files from your sensitive AV software, however, the new layer of obfuscation can again falsely trigger other AV solutions. Besides that, these protection tools are handy in terms of keeping your proprietary software away from prying eyes, but they are also often misused by malware authors as a form of obfuscation technique. Keep in mind that the use of a protector might result in an unwanted false positive detection, which is not acceptable during the wide distribution of your own software solutions.

References

[1] KORET, J. and E. BACHAALANY. *The Antivirus Hacker's Handbook.* John Wiley & Sons, Inc., Indianapolis, 2015. ISBN 978-1-119-02875-8. s. 360.

[2] *What is Heuristic Analysis?* [online]. [accessed 10. February 2022]. Available at: https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis

[3] *How it works?* [online]. [accessed 10. February 2022]. Available at: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

[4] *Malware analysis sandbox aggregation: Welcome Tencent HABO!* [online]. [accessed 10. February 2022]. Available at: https://virustotal10.rssing.com/chan-4742985/article104-live.html

[5] *Explained: Packer, Crypter, and Protector.* [online]. [accessed 10. February 2022]. Available at: https://blog.malwarebytes.com/cybercrime/malware/2017/03/explained-packer-crypter-and-protector/

[6] *Obfuscation: Malware's best friend.* [online]. [accessed 10. February 2022]. Available at: https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/

[7] *Themida.* [online]. [accessed 10. February 2022]. Available at: https://www.oreans.com/Themida.php

[8] *MEW.* [online]. [accessed 15. February 2022]. Available at: https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/MEW-SE.shtml

[9] *MPRESS.* [online]. [accessed 15. February 2022]. Available at: https://www.autohotkey.com/mpress/mpress_web.htm

[10] *PEtite Win32 Executable Compressor.* [Online]. [accessed 15. February 2022]. Available at: https://www.un4seen.com/petite/

[11] *UPX.* [online]. [accessed 15. February 2022]. Available at: https://upx.github.io

[12] *WinRAR.* [online]. [accessed 15. February 2022]. Available at: https://www.win-rar.com

[13] *Enigma Protector.* [online]. [accessed 15. February 2022]. Available at: https://enigmaprotector.com/

[14] *About Obsidium.* [online]. [accessed 15. February 2022]. Available at: https://www.obsidium.de/home

[15] *Software protection system.* [online]. [accessed 15. February 2022]. Available at: PELock Software Protection & Software License Key System

[16] *VMProtect software.* [online]. [accessed 15. February 2022]. Available at: https://vmpsoft.com

[17] *Fully UnDetectable (FUD).* [online]. [accessed 15. February 2022]. Available at: https://www.neushield.com/learn/fully-undetectable-fud/

1st Lt. Dipl. Eng. Andrej **FEDÁK** (PhD. student)
Armed Forces Academy of General M. R. Štefánik
Department of Computer Science
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: andrejfedak@gmail.com

Prof. Dipl. Eng. Jozef **ŠTULRAJTER**, CSc.
Armed Forces Academy of General M. R. Štefánik
Department of Computer Science
Demänová 393
031 01 Liptovský Mikuláš
Slovak Republic
E-mail: jozef.stulrajter@aos.sk

Andrej Fedák was born in Žiar nad Hronom in 1994. He received his engineering degree from the Armed Forces Academy of General M. R. Štefánik in Liptovský Mikuláš in the field of Military Communication and Information Systems. He is currently an officer of aeronautical ground information systems - Air Force Headquarters. His research is focused on computer networks, information systems, information and cyber security.

Jozef Štulrajter works as a professor at the Department of Informatics, Armed Forces Academy of General M. R. Štefánik in Liptovský Mikuláš. He graduated (Ing.) at the Military Technical College in 1974. He obtained the degree of CSc. diploma in Theoretical Electrical Engineering - Theory of Circuits and Systems of the Military Academy in Liptovský Mikuláš in 1992. His research interests include Information and Communication Technology (ICTs), computer architectures, image coding, computer security.