An Experimental Evaluation on Proposing a Methodology for Assessment of Packing in DFIR of Ransomware Binaries

João Ribeiro[1,2] and Hajime Shimada[3]
[1]Graduate School of Informatics, Nagoya University, Aichi, Japan
[2]Criminalistics Institute, Civil Police of Federal District, Brazil
[3]Information Technology Center, Nagoya University, Aichi, Japan

assisribeiro@net.itc.nagoya-u.ac.jp
shimada@itc.nagoya-u.ac.jp

Abstract: When investigating ransomware incidents, DFIR (Digital Forensics and Incident Response) personnel and law enforcement agents are often tasked with performing Forensic Analysis and Reverse Engineering of malware to understand, evaluate and assess key features of the malicious executable to be able to establish authorship and materiality of the cyber-attack. In this light, there is often the challenge of dealing with packing of executable files, a feature that malware authors employ to hide malicious features, to avoid detection or to hinder reverse engineering. Although there are many options for malware analysts to deal with this issue, such as online sandbox services and platforms designed for automated, large-scale malware analysis of binaries, they might not be the suitable for DFIR personnel and law enforcement actors entrusted with the investigation of cyber incidents, because, amongst other factors, they might entail the submission of a live sample to a external website or platform, leading to a breach in the chain of custody and confidentiality. They may not output pertinent information of forensic value, act as black boxes, or they may not accurately or sufficiently replicate the environment or IT ecosystem present in each incident. They are often paid-for services or with often limited or inflexible resources and time constraints for free analysis options. Given this, we discuss some of the peculiarities of assessing the packing aspect of malware in the context of ransomware incidents, while carrying out an experimental evaluation of a methodology for assessing that feature in ransomware binaries. The main goal of this assessment is to determine whether a given ransomware sample unpacks itself and how, while also providing the analyst valuable insights about key characteristics of its unpacking process. The proposed methodology combines static and dynamic analysis indicators, in a dynamic multi-pass approach for increased robustness, while also adopting previously established metrics for measuring unpacking found in previous, generic malware research.

Keywords: Ransomware, Digital forensics, Malware analysis, Cybercrime, Packed malware

1. Introduction

In the investigation and DFIR procedures involving ransomware incidents, Malware Analysis and Reverse Engineering often plays a pivotal role. By analysing in-depth key characteristics of binaries, analysts can gain insight into the inner workings of one of the central pieces of software used to perpetuate an attack in a target computer system or IT infrastructure. One of those potential features is packing, in which malware authors employ to hide malicious features or to avoid detection. While there are many tools and resources available for dealing with that feature in the broader context of cyber security (e.g. prevention and detection), in Forensic Analysis settings there is a lack of dedicated tools and methodologies that can both adhere to chain of custody procedures and properly output detailed information invaluable to the analyst in the form of a forensic report.

In the literature, there is no precise and clear definition of packing, but an often accepted one is that of a binary executable that contains a piece of code present in the binary file but not when immediately loaded into memory, which is later decrypted, decompressed, decoded or otherwise transformed and executed in real-time according to Mantovani et al (2020). In ransomware incidents and more specifically those that rely on crypto ransomware, given the nature of the attack, the defining malicious behavioural characteristic is arguably that of data compromise. Amidst the actions taken by ransomware to achieve such result, the encryption of a victim's files is, although not the only one (e.g. deletion of Shadow Copies, stoppage of remote backup services, etc.), it is arguably the most fundamental part of it.

2. Related Work

Although there are in the literature extensive works on ransomware classification such as Aslan and Yilmaz (2021), Nurnoby and El-Alfy (2019), on general packing identification and classification Biondi et al (2019), Gao et al (2022) on packing taxonomy Muralidharan et al (2022), and on establishing static analysis frameworks Sharif et al (2008), the literature on in-depth Forensic Analysis in ransomware is rather scarce, considering that the main research focus in comprehensibly on detection and prevention. Sechel (2019) performs a comparative assessment of obfuscated ransomware detection methods using antivirus engines and dynamic

analysis with Cuckoo Sandbox, analysing 10 samples with a Windows 7 32-bit Cuckoo Sandbox environment. Kara and Aydos (2022) proposes a forensic analysis method alongside a case study but there is no reference to packing aspect of ransomware. Chayal et al (2022) performs an extensive literary review on Ransomware Forensic Analysis but there is also no mention of the packing (or unpacking) problem.

Considering the scarcity of research in Forensic Analysis of ransomware (and more specifically in the packing aspect) in the context of DFIR and law enforcement investigations (i.e. with limited legal and resource constraints) and also the challenge that packed malware proposes to Malware Static Analysis and Reverse Engineering, we prototype a methodology for in-depth assessing of packing in ransomware binaries, with also the consequential goal of serving as a preliminary analysis step in a latter to be implemented Ransomware Behavioral Analysis Methodology (Ribeiro et al, 2023).

3. Proposing a Methodology for Assessment of Packing in DFIR of Ransomware Binaries

We propose a prototype approach for in-depth, suitable for small-scale assessment of packing of ransomware binaries in Windows platforms (either 32 or 64-bit) with the following goals:

- Assuming a packed sample, to confirm or reject that the sample is packed or not according to our defined metric.
- Establishing how and when the ransomware sample unpacks itself with regards to the encrypting loop in the case of unpacking.
- Serving as a preliminary step for further analysis by providing the analyst insights into the malware behavior.

Figure 1 shows the flow of the proposal. A triage stage detects if the binary is packed or not. Then we obtained initial metrics from Static Analysis and after two steps of Dynamic Analysis. Finally, based on the calculated metric, we establish the final assessment result. The Dynamic Analysis is divided into two stages to increase the robustness of the analysis. In the first stage, the idea is to allow the sample to be executed with little or no intervention by the analyst, tracking and obtaining overall ransomware dynamic execution behavior indicators. Then, from these, identify those that have “potential unpacking interest” and track their execution in a second pass.

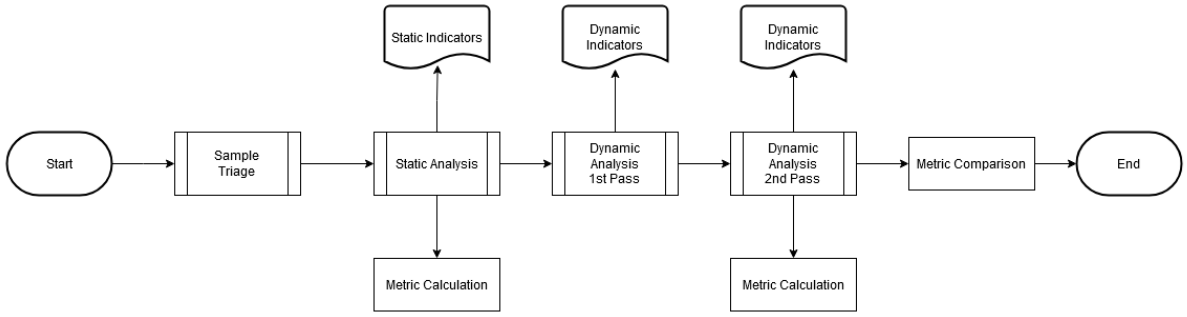

Figure 1: Main overview of Assessment of packing in DFIR of ransomware binaries

3.1 Proposed Analysis Environment Setup and Configuration

Our analysis environment mainly consisted of:

- An installation of VirtualBox on a Windows 10 64-bit host with an updated FLARE-VM installation (Windows 10 64-bit guest), which is a VM-based Reverse Engineering environment alongside an up to date 64-bit version of REMnux3, also a VM-based Linux toolkit for Malware Analysis. Both the machines were configured to be in a Host-only virtual network with static assigned IP addresses, managed by the Virtual Box DHCP local server;
- In the FLARE-VM installation, various Malware Analysis and Reverse Engineering tools were installed including, but not limited to:
    - Sysinternals Suite, containing tools such as procmon;
    - SystemInformer, a tool used for process monitoring (formerly known as Process Hacker);

- x64dbg Debugger, with anti-anti-debugging and anti-anti-analysis tools such as ScyllaHide;
- API Monitor, a Windows API Monitoring tool for 32-bit and 64-bit executables;
- Ghidra, a software reverse engineering suite;
- Various packing identifiers and PE file analysis tools such as Detect it Easy, ExeinfoPE and PEiD.

- In the REMnux virtual machine installation, INetSim, an internet simulator software, was set to simulate internet connectivity (namely HTTP/HTTPS/DNS responses) but configured to not deliver executable code given that our analysis is limited to the first layer of unpacking and that simple file droppers obtained from the internet can be analysed separately at a later time).

3.2 Sample Triage

Given that our scope is limited to crypto-ransomware, and considering that the tag "ransomware" is an umbrella term (i.e. many samples might be labelled as such due to being related to a ransomware incident but the binary itself might not be an actual crypto-ransomware binary), we felt it is necessary to perform a triage of the samples that abide to the following criteria:

- Were executable or loadable (in case of Dynamic Loaded Libraries);
- Were not .NET executables (even though those can be obfuscated, there is no original machine code to be unpacked, considering it is a form of dynamic generated code);
- Encrypted a file within 5 minutes of execution;
- Were identified as being packed by at least 1 out of 3 packing identifying tools (Detect It Easy, ExeinfoPE and PEiD). We also considered the case where any of the tools labelled the sample "crypted" as packed binaries.

For determining 1) above, appropriate checks and changes to PE headers (i.e. Machine Subsystem, Execution Flag, etc.) are made to account for modifications probably introduced by uploaders to prevent accidental execution of malicious samples. Also, for 3), appropriate measures would be taken in the case it was found that the binary employed anti-unpacking techniques according to Muralidharan et al (2022) (i.e. anti-debugging, anti-intercepting and anti-emulating) such as API Hooking and modification of PE Headers. For Dynamic Loaded Libraries files, execution point would be set at the DLL entry point.

3.3 Static Analysis

As an initial step of our analysis (step 2 of Figure 1) , we perform an initial metric evaluation of packing, based on entropy and code-to-data ratio (C2DR) which is explained in depth in Section 4, by loading the binary into Ghidra Headless Analyzer using a simple script that performs an estimation of C2DR, alongside the Detect-it-Easy tool for calculating the entropy of the sample.

Afterwards, we perform static analysis of the binary, identifying any potential "hint" of the packing algorithm that might be provided by the triage step. The idea is to check for common, off-shelf and not customized packing algorithms such as UPX, nPack, etc. Also, we search for sections with potential indicators of packing, that is, those that stand out with "unusual" names (i.e. not variants of .text, .data, .reloc etc.), with high-entropy (greater than 7), irregular size or containing potential embedded and encrypted containers and files.

If any of the tools identifies a packing algorithm (e.g. UPX), we perform OSINT to check if there is any available, dedicated tool to statically unpack it. If there is, we use it as a reference to the next step, if not, we "mark" the suspicious sections as of "potential unpacking interest" and proceed to the next step.

3.4 Dynamic Analysis

The Dynamic Analysis is divided in two steps (steps 3 and 4 of Figure 1), a first pass and a second pass. The approach in the first pass is to allow the ransomware to run with minimal intervention, tracking general API usage, process behavior and overall system resource usage. As for the second pass, the idea is to narrow down for key API calls and memory regions of potential "unpacking relevance", analysing it in a debugging environment while applying appropriate countermeasures for anti-unpacking techniques.

3.4.1 Dynamic analysis - first pass

Firstly, we load the binary in either the 32-bit or 64-bit version of API Monitor software, preferably setting the import option to "Remote Thread (Extended)", which experimentally we found to be the most effective general anti-debugging option. If the ransomware does not show typical behavior observed in the triage step or dies silently, we adjust the import option accordingly. We configured a filter for most common API's used inunpacking malware such as VirtualAllocEx, VirtualProtectEx, etc. according to Korczynski (2019). Our approach was to trace the memory allocations using the write-then-execute methodology, with a hands-on approach and leveraging process tracing (procmon), API calls monitoring (API Monitor) and overall system resource usage (SystemInformer).

The executable was set to run until:

- It reaches the end of encrypting loop: the interval between two files being encrypted is greater than 5 minutes and/or
- It exits or terminates within 2 hours or
- The virtual machine crashes or becomes unresponsive.

After this phase, initial Dynamic Indicators of unpacking are obtained, such as potentially relevant API calls, memory regions and program execution flow, which will feed into the second pass of analysis.

3.4.2 Dynamic analysis - second pass

From the previous step, we performed additional filters to the API calls and memory references obtained from the last step, identifying those that have showed unpacking behavior (write then execute new memory regions). We then loaded (or attached a debugger to) the binary setting API breakpoints at the API calls of reference and to hardware breakpoints (execute) on the referenced memory regions. In the same manner as last step, we adjusted the debugging import and/or attaching option on a case-by-case basis to account for anti-debugging and anti-unpacking techniques (e.g. using plugins such as ScyllaHide, with varying levels of countermeasures).

Also, depending on how the previous step of analysis unfolded, we also enabled a set of special breakpoints relative to program execution flow. Therefore, in addition to those defined by the first pass we also performed the following actions:

- *Terminated encrypting loop proceeded by program termination*. If the program exited the encrypting loop in a time less than 2 hours and terminate its execution (including child processes, remote threads, etc.) we set a breakpoint before the call to the API that caused the process to exit (e.g. ExitProcess);
- *Terminated encrypting loop. If the program exited the encrypting loop* (i.e. the time between the encryption of two files was greater than 5 minutes) but not the main program loop, we set a breakpoint before the first API call after the last interval between file encryption greater than 5 minutes;
- *Unterminated encrypting loop*. If the program did not exit the encrypting loop after 2 hours, we set a breakpoint on ExitThread, which later was used as a reference for finding the next closest instruction executed by the main thread of the ransomware binary and for performing the dump of the executable;
- *Machine crash or reboot*. If the virtual machine crashes or becomes unresponsive we set a breakpoint before the execution of the call that crashed the machine.

We then proceed to analyse each sample in a debugging environment up until it reaches the execution point established in the first pass, inspecting the memory regions of interest and performing the appropriate memory dump if it meets the established threshold of 800 bytes. This is to differentiate between code snippets (e.g. string decryption containing command line arguments for the deletion of Shadow Copy) and unpacked code. We then dump the memory region(s) of interest, accordingly, using the built-in dump function of x32dbg/x64dbg, or, in case of injected code, we dumped the process segment with SystemInformer. We repeat this step as necessary to handle evasion, Anti-Analysis and Anti-Debugging techniques employed by the ransomware binary. Finally, as a last step we dumped the PE file using xdbg Scylla plugin, in accordance with the state reached as result of the Dynamic Analysis First Pass, and, if necessary, also performing the "Fix Dump" and "PE Rebuild" options of Scylla (e.g. in the case of incorrect headers). For obtaining the dumps, we halted the execution of the ransomware in the debugger, stopped at the next instruction in user code and

suspended all the threads (typically encrypting loop threads). For obtaining the dumped and/or executable files, we enabled a virtual shared folder and copied the files to the host machine.

4. Results

4.1 Obtaining the Samples

We downloaded 1000 random files from MalwareBaazar that were tagged “ransomware” and “Windows”. From those samples, we filtered the first 20 samples (Table 1 of Appendix) that adhered to criteria established on Section 3.2.

4.2 Metric Results

Considering the challenge of defining a metric for establishing unpacking of malware in the wild where there is no ground truth (i.e. the original code cannot be obtained), Cheng et al (2023) and Sharif et al (2008) propose the use of empirical statistic heuristics to determine if an executable is packed or not. They are entropy difference and code-to-data ratio. Namely, if 1) a segment of memory with executable permissions is found in memory but not in executable form and 2) there is a change of entropy Δe greater than 0.4 for files with initial entropy e > 7 (High-entropy packed), or if code-to-data ratio C2DR is greater than 0.5 in the unpacked code for e < 7 according to Cheng et al (2018) (Low-entropy packed). For differentiating between unpacking and code snippets, Mantovani et al (2020) proposes setting the threshold at 800 bytes. Therefore, for calculating the entropy of the initial executable and the dumped code (or dumped PE file using Scylla) we used Detect it Easy in which entropy calculation is based on Bintropy, according to Lyda and Hamrock (2007). For calculating the code-to-data ratio (C2DR) we leveraged Ghidra Headless Analyzer after loading a binary using all default, non-prototype analyses. Even though code and data are often mingled in malicious executables, and that differentiating between code and data is a known hard problem, it is known, according to Sharif et al (2008) that disassemblers often mislabel data as code and not the other way around, so the provided value of code-to-data ratio tends to err on undervaluation of C2DR. In the case of process injection, we rebuilt the injected code into a PE file using appropriate tools. The raw results of our analysis with regards to entropy *e* and code-to-data ratio *C2DR* are presented in Table 1.

Table 1: Raw metric results for packing analysis

#	e_i	e_f	Δe	$C2DR_i$	$C2DR_f$	Final Result
1	7.03861	6.90401	0.13459	0.88052	0.87242	Not packed
2	7.14411	7.33128	-0.18717	0.88600	0.88596	Not packed
3	7.62705	5.49575	2.13123	0.38644	0.98730	Packed
4	7.87699	4.94529	2.93170	1.00000	0.52839	Packed
5	6.80623	6.64445	0.16178	0.39599	0.46050	Not packed
6	7.43978	7.34636	0.09342	0.89150	0.97236	Not packed
7	7.76664	7.70744	0.059199	0.97085	0.97085	Not packed
8	6.75859	7.31419	-0.55560	0.85068	0.88596	Not packed
9	7.03122	7.07820	-0.04698	0.32415	0.22610	Not packed
10	6.99935	6.72380	0.275549	0.55625	0.61568	Packed
11	6.55232	6.53141	0.02091	0.79891	0.78943	Not packed
12	6.57078	6.41059	0.16019	0.79768	0.78957	Not packed
13	7.11508	7.04976	0.06532	0.87069	0.97236	Not packed
14	7.05922	7.27143	-0.21221	0.88600	0.88596	Not packed
15	6.74297	6.79675	-0.05378	0.86194	0.82152	Not packed
16	6.59007	6.31463	0.27544	0.65938	0.63434	Not packed
17	7.08091	7.28813	-0.20722	0.88600	0.88596	Not packed
18	7.45796	7.47596	-0.01799	0.13945	0.65812	Not packed
19	6.50183	6.32686	0.17497	0.75796	0.74628	Not packed
20	7.75153	6.55561	1.19592	0.41639	0.97480	Packed

4.3 Indicators

The summarized results of each step of analysis (indicators) are presented in Table 2.

Table 2: Indicators obtained through each step of analysis

#	Family/ Signature	Arch		Packed?			Static Analysis Indicators	Dynamic Analysis Indicators	Dynamic Analysis End State	Final Result
		x32	x64	DIE	Exeinfo pe	PEiD				
1	Venus	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
2	Phobos	✓		✓	✓	✓	.cdata section	-	2	Not packed
3	Conti	✓		✓	✓	✓	UPX	-	2	Packed
4	Magniber		✓	✓	✓		.DLL obfuscation	Process Injection	2	Packed
5	Azov		✓	✓			Signed Binary	-	2	Not packed
6	Dharma	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
7	Dharma	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
8	Phobos	✓		✓		✓	Overlay	Multiple process creation	2	Not packed
9	Dharma	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
10	Azov		✓	✓			-	Dummy GUI Application	1	Packed
11	-	✓		✓			-	-	1	Not packed
12	-	✓		✓			-	-	2	Not packed
13	Dharma	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
14	Phobos	✓		✓	✓	✓	.cdata section	-	2	Not packed
15	DoNex	✓		✓	✓		-	-	4	Not packed
16	Spora	✓		✓		✓	.text section	-	1	Not packed
17	Phobos	✓		✓	✓	✓	.cdata section	-	2	Not packed
18	Dharma	✓		✓	✓	✓	-	Multiple process creation	2	Not packed
19	Mallox	✓		✓		✓	-	-	1	Not packed
20	Stop	✓		✓	✓	✓	.data section	Multiple process creation	2	Packed

4.4 Discussion

4.4.1 Unpacking results

Out of the 20 samples examined, only samples 3, 4, 10 and 20 were, according to the established metric, packed. Sample 3 was packed with UPX packer, Sample 4 was packed with a custom packer that employed process injection. Sample 10 was made of a dummy GUI application, where the actual ransomware binary was loaded in memory. As for Sample 20, it implemented a custom packer that spawned a different process. All the other samples were not packed and were mistakenly identified as packed by most tools. The reason for that can be partially explained by the theoretical result that the unpacking problem is undecidable, and also due to the fact that as its natural with ransom malware, they often contain ransom notes in form of text files and/or image file often encrypted, encoded or compressed. Because it is seldom possible to differentiate between encrypted/encoded/compressed data and packed code statically, it is left for dynamic analysis, however, static tools might mislabel then as packed due to obfuscation techniques employed by the malware.

4.4.2 Problems with current approach, possible mitigations

Write then execute and the debugging approach towards unpacking. The debugging approach towards the packing problem has some advantages to the analyst. More fire-grained control of program execution flow, the ability to provide countermeasures against anti-VM, anti-analysis, and anti-reversing are some of those, to name a few. Although these can come in handy for small scale, focused DFIR scenarios, they also present many challenges. Namely, anti-debugging subterfuges specifically designed to thwart this approach (Apostolopoulos et al), inefficiency, level of expertise required by the analyst, lack or limited support for automation and so on. With ransomware forensic analysis, also there is the need to safeguard the debugger files and executable from being encrypted or tampered with by the malware, which can be troublesome (Apostolopoulos et al (2021)) considering that they share the same level of execution privilege. To alleviate this issue, we ran the debugger on a dedicated folder with appropriate level of write permissions.

Automation. Adding to the natural challenges of automating ransomware analysis and reverse engineering for DFIR purposes (i.e. computational cost, robustness and reliability of analysis, etc.) our approach also relies, as previously mentioned, on the use of debugger in the Second Pass of Dynamic Analysis which hinders complete automation difficult. Also, there is the high memory demand of the logging and tracing tools, and the disk space demands for storing snapshots of virtual machines, which is large. We believe a natural step would be to decouple the debugging step from within the guest-OS and use hypervisor-assisted debugging.

5. Conclusion

We performed an experimental evaluation of a methodology for the assessment of packing in DFIR of ransomware binaries, using open source and/or freeware tools and adopting established literature metrics for determining if a ransomware binary Is packed or not, while also providing insights into the packing mechanism.

During our limited experimental evaluation, we were able to observe that most samples were not packed, which is understandable considering that malware authors typically want to avoid packing overhead which could lead to performance issues and therefore can hinder the success of the attack (i.e. compromising victim's data) in a ransomware cyberattack.

Although is arguable that general malware analysis tools might suffice for such assessment of packing in conventional scenarios of detection and prevention, we think that our proposed approach, after further refinement and improvement, can advance the field of DFIR and Reverse Engineering for those cases where, due to legal or technical constraints, general malware analysis solutions cannot be used.

Acknowledgements

This work was partially supported by JSPS KAKENHI Grant Numbers 23K28086.

References

Apostolopoulos, T., Katos, V., Choo, K. K. R., & Patsakis, C. (2021). Resurrecting anti-virtualization and anti-debugging: Unhooking your hooks. Future Generation Computer Systems, 116, 393-405.

Aslan ¨O, Yilmaz AA (2021) A new malware classification framework based on deep learning algorithms. Ieee Access 9:87936–87951

Biondi F, Enescu MA, Given-Wilson T, et al (2019) Effective, efficient, and robust packing detection and classification. Computers & Security 85:436–451

Chayal NM, Saxena A, Khan R (2022) A review on spreading and forensics analysis of windows-based ransomware. Annals of Data Science pp 1–22

Cheng B, Li P (2018) Bareunpack: generic unpacking on the bare-metal operating system. IEICE TRANSACTIONS on Information and Systems 101(12):3083–3091

Cheng B, Ming J, Fu J, et al (2018) Towards paving the way for large-scale windows malware analysis: Generic binary unpacking with orders-of-magnitude performance boost. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp 395–411

Cheng B, Leal EA, Zhang H, et al (2023) On the feasibility of malware unpacking via hardware-assisted loop profiling. In: 32nd USENIX Security Symposium (USENIX Security 23), pp 7481–7498

CuckooSandbox (2020) Creation of the virtual machine — cuckoo sandbox v2.0.7 book. URL https://cuckoo.readthedocs.io/en/latest/installation/guest/creation/?highlight=windows%207#creation-of-the-virtual-machine

D'ALESSIO S, MARIANI S (2015) Pindemonium: a dbi-based generic unpacker for windows executables

Devi D, Nandi S (2012) Detection of packed malware. In: Proceedings of the First International Conference on Security of Internet of Things, pp 22–26

Gao X, Hu C, Shan C, et al (2022) Malicage: A packed malware family classification framework based on dnn and gan. Journal of Information Security and Applications 68:103267

Kara I, Aydos M (2022) The rise of ransomware: Forensic analysis for windows based ransomware attacks. Expert Systems with Applications 190:116198

Korczynski D (2019) Precise system-wide concatic malware unpacking. arXiv preprintarXiv:190809204

Lyda R, Hamrock J (2007) Using entropy analysis to find encrypted and packed malware. IEEE Security & Privacy 5(2):40–45

Madani H, Ouerdi N, Boumesaoud A, et al (2022) Classification of ransomware using different types of neural networks. Scientific Reports 12(1):4770

Mantovani A, Aonzo S, Ugarte-Pedrero X, et al (2020) Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In: NDSS 2020, Network and Distributed System Security Symposium, 23-26 February 2020, San Diego, CA, USA, Internet Society

Medhat M, Essa M, Faisal H, et al (2020) Yaramon: A memory-based detection frame-work for ransomware families. In: 2020 15th International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, pp 1–6

Muralidharan T, Cohen A, Gerson N, et al (2022) File packing from the malware perspective: Techniques, analysis approaches, and directions for enhancements. ACM Computing Surveys 55(5):1–45

Nurnoby MF, El-Alfy ESM (2019) Overview and case study for ransomware classification using deep neural network. In: 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), IEEE, pp 1–6

PANDA (2024) Panda user manual. URL https://github.com/panda-re/panda/blob/dev/panda/docs/manual.md#emulation-details

Ribeiro J, Yukiko Y, Hasegawa H, et al (2023) Discussion about requirements gathering for proposing a forensic ransomware behavioral analysis methodology. In: Forum on Information Technology (FIT) 2023, L-021, pp. 183-186

Royal P, Halpin M, Dagon D, et al (2006) Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: 2006 22nd Annual Computer Security Applications Conference (ACSAC'06), IEEE, pp 289–300

Sechel S (2019) A comparative assessment of obfuscated ransomware detection methods. Informatica Economica 23(2):45–62

Sharif M, Yegneswaran V, Saidi H, et al (2008) Eureka: A framework for enabling static malware analysis. In: Computer Security-ESORICS 2008: 13th European Symposium on Research in Computer Security, M´alaga, Spain, October 6-8, 2008. Proceedings 13, Springer, pp 481–500

Appendix

Table 1: Analyzed samples

#	Sample hash (SHA-256)
1	0a4e5832841ffff9f8d27ce8216d655c8743b682fff0f90dee6bd3ea83dec028
2	0b997e8b0d0ff6cc4e6f1919c6c0f3080eaa0d08c8fccdf50f7648bf05cca446
3	05bbf1c653825b757ee73b59df45410070a28841819362462162d9547adb3d5a
4	06acd697bc0a41a6fa1098eba46ddd40d029a5fef3eb152fbf9d0d39e6f8673d
5	0733175b535b3c30160e804bac7f365b5b1d071a80537dc61342d9cfa436ba1c
6	0775f6c00ce22c822bb56ed45d658af3647e522d0e462d505e31dfc0f343adde
7	0b2467264b2544634a7252314e585b10b618d1e752b2aa7fd46c59210b9b93f6
8	0c0c9a19db1f89d94ddcd8af54fa631798e3ccc82743faae6d9818759f2dbcc1
9	30140a3a441d4d92bc78e6726d9de9a293e83f812c16658222c32ce408c453ba
10	0d654bd41f1aa5790624656e942f317e5984d139a3f17cb6f167544d713609a8

#	Sample hash (SHA-256)
11	6a207be6807f1ea51f0bdeeb89e3ea4f0560f48f9b7ed3ed4ea68a212e2714ca
12	8e974a3be94b7748f7971f278160a74d738d5cab2c3088b1492cfbbd05e83e22
13	a191d7d045dcf61582f2257bded2734b4ca424b1cf66ff519763c1888ec83190
14	c8d9a9758516d5a8936bd3bc01a9997fb677ed1dc54081caa985883935ff092b
15	0adde4246aaa9fb3964d1d6cf3c29b1b13074015b250eb8e5591339f92e1e3ca
16	1a247db9ae193938318c1935ebca3e258da2b1ba99902422066df28f4245a002
17	f25a3e2bbaa9bed0210adf5bff0bc5d76fbf44e09ae4bc22e40473814a6ebad1
18	e9bbcfb5d9f42ef0dd75eb435e78d5226087679593893e0c08977694e720cd7a
19	1f793f973fd906f9736aa483c613b82d5d2d7b0e270c5c903704f9665d9e1185
20	1ab5a24ccad1f77bc4a9c00c32671d19d8350d47bc8408693dbdbd3f30ea3bdd