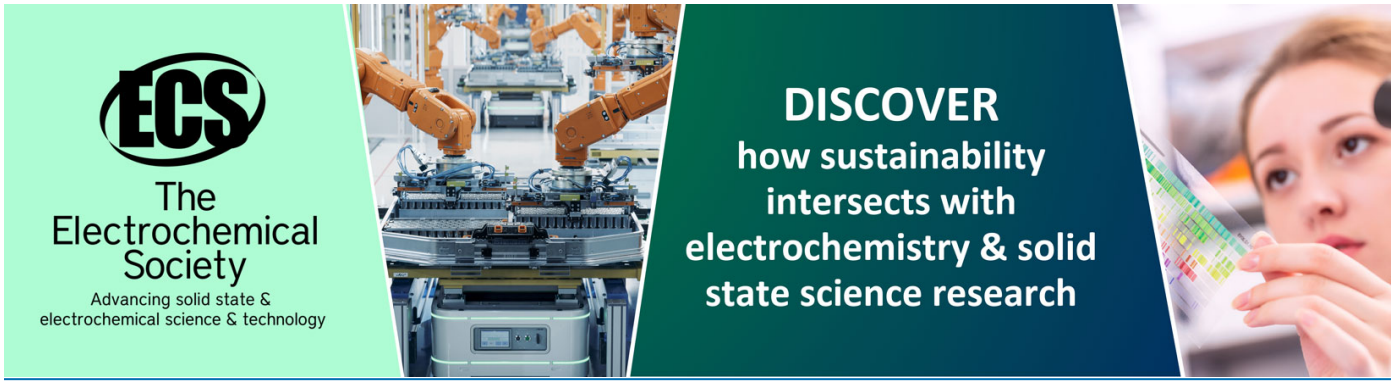
IOP Conference Series: Materials Science and Engineering

Mal-XT: Higher accuracy hidden-code extraction of packed binary executable



Mal-XT: Higher accuracy hidden-code extraction of packed binary executable

Charles Lim[12], Suryadi[3], Kalamullah Ramli[4], Suhandi[5]

[14]Department of Electrical Engineering, Universitas Indonesia, Kampus UI, Depok 16424, Indonesia
[25]Information Technology Department, Swiss German University, Kota Tangerang, Banten 15143, Indonesia
[3]Department of Mathematics, Universitas Indonesia, Kampus UI, Depok 16424, Indonesia

E-mail: charles.lim@sgu.ac.id

Abstract. Malware authors often use binary packers to hinder the malicious code from reverse-engineered by malware analyst. There have been many studies done on providing different approaches on unpacking the packed binary executable. Our previous works have successfully relied on the written memory section size as an indicator to extract hidden-code during the unpacking process. This paper enhances our previous work by locating executed instruction in the written memory section to provide a more precise memory location in extracting hidden code from the packed binary executable. The result of our experiments exhibits higher similarity result for all packers and benign applications compared to our previous works.

1. Introduction

Malicious software (malware) is one of the largest security threats to individuals and/or organizations that shared sensitive and valuable information. To persist in the computer systems, malware authors commonly use various evasion techniques from being detected by anti-malware systems. One of such techniques is code obfuscation, which includes software packer, polymorphism or metamorphism. Code obfuscation has been used to protect the software intellectual property, making it harder for software analyst to reverse-engineer it to obtain the original body of the software. In the case of software packer, it is used not only to decrease the size of binary executable but also to encrypt the original source code in such that the source code can be hidden or protected from being reverse-engineered.

With the exponential increased number of malware and 80% of these malwares in the wild are packed [1, 2], this condition makes the tasks of malware analyst even much more complex. A packed binary executable hides the original code and required libraries and with the possibility Import Address Table (IAT) being removed from the binary executable. Thus, commonly used static analysis method of analyzing malware becomes limited if not impossible [3]. Dynamic analysis (and combined with memory analysis) provides more promising alternative approach for analyzing malware. The original code from malware is usually loaded into memory sometime during the execution and obtaining the memory image at the point when the exposed hidden-code in memory becomes possible [4]. Our previous works, Mal-Xtract [5] proves that it is possible to extract hidden-code from packed binary executable by tracking the use of the consecutive memory address in a section to determine that the unpacking process is completed.

In this paper, we propose a method to detect the end of unpacking routine based on the written memory section [5] with the instruction inside the memory section being executed. We argue that the proposed method provides a better accuracy then Mal-Xtract since the executed instruction in one of the memory sections provides an indication that unpacking routine is completed. With the combined tracing of written memory section and executed instruction inside this section provides a more precise indicator of the completion of unpacking process, thus hidden-code can be extracted with higher accuracy.

The contributions of this paper can be listed as follows:

(i) We develop framework to extend the method presented in the previous work [5] to indicate a more precise end of unpacking by detecting the latest memory address executed in the memory section being observed.
(ii) We propose an enhanced memory analysis method that traces exact memory sections resulted from Mal-Xtract framework that have the first instruction being executed in the relevant memory sections.

The remainder of the paper is organized as follows: section 2 explains the related works of this research. Section 3 discusses our Mal-XT unpacking framework. Section 4 elaborates on the experiment setup and section 5 introduces our preliminary study to lay the foundation work of packed binary code analysis. Next,

section 6 presents the experiment results of this paper. Section 7 provides the analysis and discussion of experiments of our research. Finally, section 8 concludes our research and future works.

2. Related Works

Once the packed binary executable loaded into memory starts to unpack the original body of the code, Original Entry Point (OEP) pointed to by instruction pointer becomes the starting point of the execution of the hidden-code [6]. Isawa et al [6] uses the page extraction technique by tracking specific writing instructions, such as 'mov' and 'xchg', to detect written memory pages. Isawa et al considered the first executed instruction become the first candidate for the OEP.

PolyUnpack [7] uses static and dynamic integrated approach to compare the dynamically disassembled code in memory with the statically disassembled packed code. When the disassembled dynamic code is not found in the statically disassembled code, the dumped memory produces dynamically generated code. The disadvantage of this approach is the added disassembly step which complicates the process since many packers employ obfuscation that hinders the disassembly process [8]. Another weakness of PolyUnpack is that the whole process of disassembly will fail if there is one mistake during disassembly process.

Renovo [9] monitors a clean-dirty page (the written memory address is marked as dirty and the rest is clean) and specific jump instructions such as 'jmp' and 'je', to detect OEP of the packed binary executable. Similar to Renovo, OmniUnpack [10] utilizes page extraction technique with W⊕X (written then executed) policy, however OmniUnpack scans the newly generated memory page when there is a pre-defined dangerous system call is being executed. The disadvantages of OmniUnpack are the imprecision of page level tracking and continuous manual signature update is required to detect selected dangerous system calls being monitored. Justin [11] also uses the same W⊕X approach by intercepting NX bit exceptions to memory pages marking. However, the reliance on anti-virus signature database reduces the effectiveness of Justin in OEP detection.

SoK Deep Packer Inspection [12] sets out the goal of mapping packer complexity and it monitors all memory written frame and execution of each memory frame to track down possible OEP. SoK provides information on executed memory region which include memory type and address, size and number of unpacking frames. Basically, it will monitor all the execution trace and memory write process. For every new memory frame written, it monitors for the execution for each memory frame that newly created.

Mal-xtract [5] proposed a method to investigate the memory (written and re-written) section to determine the latest hidden-code that written in the memory section. In addition, it uses memory section threshold to determine which of the section are responsible for unpacking process. Our research is based on Mal-Xtract framework with additional monitoring on executed instruction in the selected memory section to determine the final OEP of the packed binary executable. Table 1 summarizes the comparison of Mal-XT framework and other related works.

Table 1. Mal-XT Framework and comparison with other approaches

Key Items	Renovo [9]	SoK [12]	Mal-Xtract [5]	Mal-XT
Memory Access Space	User & Kernel Space	User & Kernel space	User & Kernel Space	User & Kernel Space
Features	Runtime Memory Instruction	Executed Instruction	Memory Access Write & Read	Memory Access Write & Read, Executed Instruction Trace
End of Unpacking Detection	Specific memory Write instructions*	Latest frame & executed control flow	Written Memory Section threshold	Written Memory Section with executed instruction found in the memory section

*mov %eax, [%edi] and push %eax

3. Research Framework

Our research framework, Mal-XT, extends Mal-Xtract framework [5] by adding additional execution tracing on the consecutive address of written memory section monitored by Mal- Xtract. Hence, memory written section size that exceeds the pre-determined threshold is an indicator for end of unpacking process. The list of all qualified written memory sections is stored in the sections' data log, which the is used by Mal-XT to discover the potential executed instruction inside these memory sections. The execution trace performed by Mal-XT captures the following: memory address being executed, CR3 register value (indicating the related process ID being executed), and instruction counts. Mal-XT will use this information and the sections data log information to determine all the potential written memory

section and executed. Figure 1 shows Mal-XT framework with highlighted block that shows additional process to track executed instructions from written memory sections.

The algorithm that implements the execution trace is depicted in Algorithm. The correlation each line of memory section shown in line 4 and execution trace in line 5 is performed. When the content of the memory address being executed is within range of memory section then relevant memory section is marked executed as indicate in line 8 and relevant instruction count is also marked (line 9). From the list of marked memory sections with the relevant instruction count were processed and the first memory section executed is selected to provide the OEP of the hidden-code.

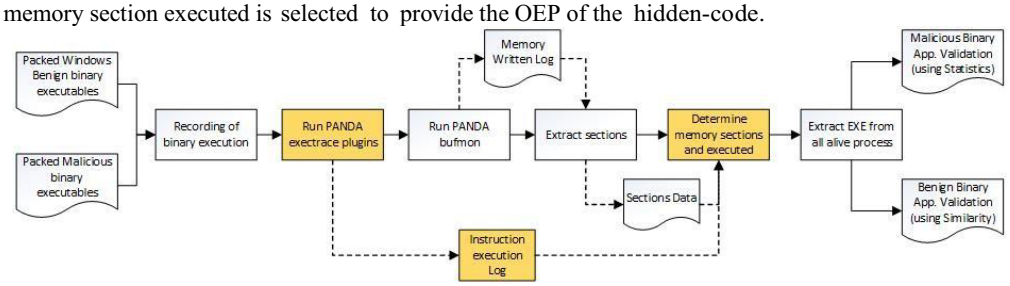


Figure 1. Mal-XT Research Framework

Algorithm Execution Trace Algorithm

```
1: procedure Instruction Trace Analysis > Corelate Instruction Log to the Memory section
2:     Replay the Recording using EXECTRACE > Get all executed Instruction Log from PID
3:     l = 1
4:     while Line(l) <> EndOfMemorySectionLogs do
5:         while Line2(m) <> EndofExectraceLog do
6:             if executed.address(m) > first.address.of.section(l) AND
7:               executed.address(m) <= last.address.of.section(l) then
8:               Mark Memory Section as executed
9:               Get Instruction Count number
10:            else
11:              Going for next Line2
12:            m++
13:    l++
```

Once the memory address with the relevant instruction count is determined, the physical memory dump can be performed to extract the relevant binary executable from memory. The extracted binary executable can then be validated: Similarity score [13] is used for measuring the extracted binary executable from packed benign binary executables and packed status of the extracted binary executable from packed malware executable. As explained by Lyda et al [14], entropy analysis alone is not enough for validating packed binary executable, additional statistical test is included, i.e. Arithmetic Mean Value, Chi-square and Serial Correlation to improve the accuracy in determining packed status of the binary executable being investigated.

4. Experiment Setup

All experiments are performed on HP Proliant DL380 G7, 256 GB RAM using VMWare ESXi 6.0 as hypervisor, Ubuntu 16.04 with 128 GB RAM and 4 Core Processor as guest OS. PANDA [15], dynamic analysis tool used in this research, requires QEMU to be installed inside Ubuntu VM and runs Windows 7 32bit SP1 (with 2 GB RAM for its QEMU VM Emulation). Finally, all binary executable samples are executed on Windows 7 32bit SP1.

The benign executable samples (a total of 26 benign applications) are collected from Windows 7 32bit SP1 System32 files. These benign executables are packed with a collection of third party either free or commercial binary packers, i.e. UPX, PECompact2, FSG, Armadillo, Molebox, WinUPack, Themida, VMProtect, ASPack, and ASProtect. Malware samples dataset are collected from VirusTotal [16] and VirusShare [17] which consist of 9 families, each family has a minimum of 30 malware samples. Malware families include Dynamer!rfn (2017), Awkolo.A (2017), Expiro.BA (2012), Expiro.BP (2013), Zonsterarch.U (2013), Zuepan.A (2017), Dorkbot.A (2011), Urelas.AA (2015) and Bicone!rfn (2016). All malware family samples are submitted to Packerinspector, an online service based on SoK [12] to ensure all malware are packed malware.

5. Preliminary Study

To validate our extracted binary executables, statistical analysis is performed to ensure that the binary executables already unpacked. A preliminary study is conducted to gather 42 benign Windows 7 32-bit binary applications and each of these files are packed with 16 common packers (a total of 672 files). The statistical analysis for Entropy, Chi Square, Arithmetic Mean Value, and Serial Correlation are calculated [18] to provide a final threshold for each. The summary of the results of the statistical threshold value calculation is shown on Table 2.

Table 2. Statistical Analysis Thresholds (packed & non-packed benign files)

Files	Qty	ENT	CQ	AM	SC (%)
Non-Packed EXE (AVG)	42	6.35	16,041,383.26	98.96	41.04
Packed EXE (AVG)	672	7.49	3,707,376.90	114.56	22.00
Threshold		6.92	**9,874,380.08**	**106.76**	**31.52**

AVG = Average; QTY = Sample Quantity; ENT = Entropy
CQ = Chi-Square; AM = Arithmetic Mean; SC = Serial Correlation

The binary executable is considered packed if 3 (three) of 4 (four) statistical is satisfied. If only 2 (two) of 4 (four) statistical criteria are met additional validation is required. Byte Histogram [19] is used to provide extra validation on this condition. Otherwise the binary executable is considered non-packed.

6. Experiment Results

We evaluate 3 (three) benign applications with 8 commonly used binary packers using our Mal-XT framework, the similarity results of the extracted binary executables is computed and compared with Mal-Xtract results, as shown in Table 3.

Table 3. Benign Executable Similarity (%) Results

Packers	Notepad			Helloworld			Cal		
	ENT	Xtrac	XT	ENT	Xtrac	XT	ENT	Xtrac	XT
UPX FSG	4.17	99.6	**100**	4.45	99.1	99.9	4.72	99.8	99.8
MOLEBOX	4.63	99.8	**100**	5.15	99.8	**100**	0.26	99.8	99.9
PEC2	5.32	99.6	99.9	4.14	99.0	99.6	5.53	99.8	**100**
ARMADILLO	4.27	99.6	99.9	4.25	99.2	**100**	0.92	99.8	**100**
VMPROTECT	1.75	99.6	99.9	3.46	99.0	99.6	2.52	98.8	99.9
WINUPACK	5.38	97.5	97.7	5.47	97.3	97.5	4.23	97.2	97.4
THEMIDA	4.31	99.6	99.9	5.43	99.6	99.9	5.33	99.8	98.9

ENT = Entropy; Xtract = Mal-Xtract; XT = Mal-XT

The similarity results demonstrated that our extraction method using Mal-XT successfully provide higher similarity results for all packers and benign applications compared with Mal- Xtract results. Even though some similarity reach 100%, some packers such as Themida and VMProtect show a lower similarity results due to many protection layers performed by these packers as confirmed by Mal-Xtract [5].

Table 4 shows statistical validation results of extracted hidden-code from packed malware samples. Detail malware samples for the relevant malware family can be found at Table 5 in

the Appendix. Statistical Analysis is used to confirm the status of the binary executable (packed or non-packed) and Table 4 confirms that each of extracted binary executables is a non-packed binary executable.

Table 4. Statistical Analysis Result on Extracted Binary Executable (Malware)

Malware Family	Samples	ENT	CQ	AM	SC (%)	Packed?
Urausy	A	6.35	28,703,970.10	95.53	36.48	NO
	B	6.34	30,305,721.51	95.89	36.84	NO
Sality	A	1.51	18,342,494.34	16.19	69.80	NO
	B	1.85	87,303,712.78	18.36	67.41	NO
Awkolo.A	A	5.78	4,539,318.59	67.91	29.90	NO
	B	5.78	4,539,255.01	67.88	29.85	NO
Expiro.BA	A	4.11	155,528,902.88	55.16	63.84	NO
	B	4.41	141,833,604.19	61.51	66.66	NO
Expiro.BP	A	4.25	165,406,348.60	57.16	61.21	NO
	B	4.24	165,512,763.06	58.13	62.17	NO
Urelas.AA	A	6.66	6,157,657.67	99.93	44.46	NO
	B	2.71	82,099,075.91	36.13	69.96	NO
Zuepan.A	A	6.30	2,163,801.11	93.76	43.27	NO
	B	5.86	3,045,047.62	86.22	47.41	NO
Crypto	A	6.39	1,977,198.45	104.80	26.62	NO

AVG = Average; ENT = Entropy; CQ = Chi-Square; AM = Arithmetic Mean; SC = Serial Correlation

7. Discussion

The experiment results show that by tracking the last instruction executed after the written memory section provide higher similarity results compared with just using written memory section alone. The lowest similarity result is 97.4% (Themida) and the highest similarity reach 100% (UPX, FSG, Molebox, PEC2). For the extracted binary executable from malware samples also have been shown to be unpacked and the extracted binary executables when uploaded to VirusTotal were categorized as the same family malware.

8. Conclusion

In this paper, we introduce Mal-XT framework that monitors executed instruction found in the written memory section as an indicator that the unpacking process is completed. The experiment results demonstrate a higher similarity result for all packers and benign applications compared to our previous works. We believe our method of hidden-code extraction will help malware analyst in gaining insight to malicious code of packed malware. We hope to improve our research in the future to differentiate between code and data found in malware.

9. Acknowledgements

We would like to thank the Honeynet Project and VirusTotal for providing various malware family samples used in our research. This article's publication is supported by Universitas Indonesia's PITTA 2018 Grant with contract no 2463/UN2.R3.1/HKP.05.00/2018.

10. Appendices

Below are the list of malware family and samples used in this research.

Table 5. Malware Families and Samples List

Malware Family	Samples	MD5 Value
Urausy	A	2ef3d3f52146c2581466cece68f6e2f3
	B	0e6deb746d1a85d38cb7a5a411d629fd
Sality	A	1eabb49197516e3e294a413f83b8ba68
	B	a69930834b1f48df9483739dad487775
Awkolo.A	A	c6c954d5a5c3100c08308c4c77f0f405
	B	abe0999bafdec20fa31fdfa689d8c683
Expiro.BA	A	0a040bd5acb8a542f3ef2120a406e192
	B	1b3ab3b1c9d465b4e15852fc03cf120c
Expiro.BP	A	ec9c91ec990013e4fc20e1207c7c5951
	B	d4d5098cbb965d228bef7d38a98a9cc0
Urelas.AA	A	d834e474e7ae0bb4c60eda8a3dd0c90a
	B	e7c0e0e3c218dc4a393cb4736555102f
Zuepan.A	A	c3f153d6847ba1c70f3d97f72251d973
	B	e2eb2b673751d176e8ebe487c6ecb238
Crypto	A	53c85399809a8ffc399bfd82d6145ced

References

[1]. Osaghae E O 2016 International Journal of Information Technology and Electrical Engineering 20 19 URL http://www.iteejournal.org/Download_dec16_pdf_4.pdf

[2]. Ugarte-Pedrero X, Santos I, Sanz B, Laorden C and Bringas P G 2012 Countering entropy measure attacks on packed software detection IEEE Consumer Communications and Networking Conference (CCNC) 2012 (IEEE) pp 164–168 URL http://dx.doi.org/10.1109%2FCCNC.2012.6181079

[3]. Moser A, Kruegel C and Kirda E 2007 Limits of static analysis for malware detection Twenty-third Annual Computer security applications conference (ACSAC) 2007 (IEEE) pp 421–430 URL https://doi.org/10.1109/ACSAC.2007.21

[4]. Babar K and Khalid F 2009 Generic unpacking techniques Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on (IEEE) pp 1–6

[5]. Lim C, Kotualubun Y S, Suryadi and Ramli K 2017 Journal of Physics: Conference Series 801 012058 URL https://doi.org/10.1088/1742-6596/801/1/012058

[6]. Isawa R, Inoue D and Nakao K 2015 IEICE TRANSACTIONS on Information and Systems 98 883–893

[7]. Royal P, Halpin M, Dagon D, Edmonds R and Lee W 2006 Polyunpack: Automating the hidden-code extraction of unpack-executing malware Computer Security Applications Conference, 2006. ACSAC'06.22nd Annual (IEEE) pp 289–300 URL http://dx.doi.org/10.1109/ACSAC.2006.38

[8]. Roundy K A and Miller B P 2013 ACM Computing Surveys (CSUR) 46 4 URL http://dx.doi.org/10.1145/2522968.2522972

[9]. Kang M G, Poosankam P and Yin H 2007 Renovo: A hidden code extractor for packed executables Proceedings of the 2007 ACM workshop on Recurring malcode (ACM) pp 46–53 URL https://doi.org/10.1145/1314389.1314399

[10]. Martignoni L, Christodorescu M and Jha S 2007 Omniunpack: Fast, generic, and safe unpacking of Malware Twenty-Third Annual Computer Security Applications Conference (ACSAC) 2007 (IEEE) pp 431–441 URL http://dx.doi.org/10.1109/ACSAC.2007.15

[11]. Guo F, Ferrie P and Chiueh T C 2008 A study of the packer problem and its solutions Recent Advances In Intrusion Detection (Springer) pp 98–115 URL https://doi.org/10.1007/978-3-540-87403-4_6
[12]. Ugarte-Pedrero X, Balzarotti D, Santos I and Bringas P G 2015 Sok: deep packer inspection: a longitudinal study of the complexity of run-time packers IEEE Symposium on Security and Privacy (SP) 2015 (IEEE) pp 659–673
[13]. Lueker G S 2009 Journal of the ACM (JACM) 56 17
[14]. Lyda R and Hamrock J 2007 IEEE Security & Privacy 5 40–45 URL https://doi.org/10.1109/MSP.2007.48
[15]. Dolan-Gavitt B F, Hodosh J, Hulin P, Leek T and Whelan R 2014 Repeatable reverse engineering for the greater good with panda Tech. rep. Department of Computer Science, Columbia University URL http://dx.doi.org/10.7916/D8WM1C1P
[16]. VirusTotal 2018 Virustotal - analyze suspicious files and urls to detect types of malware including viruses, worms, and trojans. https://www.virustotal.com
[17]. Virusshare 2017 Virusshare http://http://virusshare.com/
[18]. Jacob G, Comparetti P M, Neugschwandtner M, Kruegel C and Vigna G 2012 A static, packer-agnostic filter to detect similar malware samples International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (Springer) pp 102–122 URL https://doi.org/10.1007/978-3-642-37300-8_6
[19]. Wojner C 2016 Bytehist URL https://cert.at/downloads/software/bytehist_en.html