Proceedings of the Federated Conference on Computer Science and Information Systems pp. 237–246
DOI: 10.15439/2019F244
ISSN 2300-5963 ACSIS, Vol. 18

Malicious and Harmless Software in the Domain of System Utilities

Jana Šťastná
Technical university of Košice
Letná 9, 042 00 Košice, Slovakia
Email: jana.stastna@tuke.sk

***Abstract*—The focus of malware research is often directed on behaviour and features of malicious samples that stand out the most. However, our previous research led us to see that some features typical for malware may occur in harmless software as well. That finding guided us to direct more attention towards harmless samples and more detailed comparisons of malware and harmless software properties. To eliminate variables that may influence the results, we narrowed down our research study to specific software domain - system maintenance and utility tools. We analysed 100 malicious and 100 harmless samples from this domain and statistically evaluated how they differ regarding packing, program sections and their entropies, amount of code outside common sections and we also looked at differences in behaviour from the high-level view.**

I. Introduction

WHEN studying research papers in the field of malware research, one may get the impression that harmless software is somehow neglected in research studies. The idea we have specifically in mind is that presence or absence of specific features or differences in their qualities in malware with comparison to harmless software is seldom examined. However, such studies would be of great help.

Many research works use harmless software only as a resource for demonstrating detection rate of new presented detection method. However, selection of harmless samples may considerably influence detection results and thus detection rates. When programs that are part of the default system installation are used as a control group in published research works, they secure lower false-positive ratios, but they do not form complete representative set of harmless software, since many different software products are available and some of them may even resemble malware in some of their features. This idea initiated our first experiments targeted at packing and related properties of programs [1][2]. We discovered that harmless software shares occurrence of packing with malware, together with other related properties. This led us to deeply examine malicious and harmless samples and search for hidden relations.

Our experiment presented in this paper is unique by means of samples selection focused on specific software domain – system utilities and maintenance tools. By this rather narrow selection we aim to eliminate the influence of usage domain of software which may play important role in exhibited behaviour and properties. In this way we can better compare malicious and harmless samples and look for features that may help in distinguishing them.

This work has been supported by the Slovak Research and Development Agency under the contract No. APVV-15-0055 and by project KEGA no. 079TUKE-4/2017.

II. Background of the Experiment

Detecting that a program is packed is the first step towards its in-depth analysis [3]. Execution of packed program is often inevitable for recovering original program's code and unveiling its behaviour. Dynamic inspection of malware poses a risk that it may escape from analytic environment and spread on more systems, therefore strict security precautions need to be taken when executing malware.

A. Packing as Analysis and Detection Prevention

Packers employ compression for reduction of program's size and encryption to obstruct program's reverse-engineering [4]. The resulting file comprises an unpacking routine and packed data blocks. When packed program is executed, the unpacking routine recovers the original program code into memory and directs the execution flow to execute it. Program code can be retrieved by virtual machine monitors or emulators [5] but researchers also look for static techniques to distinguish packed malware from goodware [6][3]. Packers are popular for hindering signature-based detection and static analysis.

A general belief is that packing, together with obfuscation, is a common trait in malicious programs and this idea is repeated among malware analysts and researchers [7][8]. Despite that, it is not easy to find current and accurate rates of packer detections. According to Cisco Blog, they estimate around 70-80% of malware is packed and only around 5% of harmless software is modified by packers [1]. Considering the year the article was published (2010) the rates are now outdated, however, a more recent blog article on Malwarebytes from October 31, 2017 states that "*over the last quarter, we've seen an increase in malware using packers, crypters, and protectors*" and "*the growing number of malware authors using these protective packers has triggered an interest in alternative methods for malware analysis*"[9]. As it seems, packers are not on the decline yet, so investigating their occurrence in harmless software may lead to interesting insights.

[1]https://blogs.cisco.com/security/malware_validation_techniques

B. Introducing Research Hypotheses

Packing is deemed typical for malicious software but results that would confirm and explain reliability of this assumption are not present. We will try to shed some light on this problem and see if assumptions regarding packing match the reality. Our research is evaluated with statistical tests of null hypothesis and two alternative hypotheses for each of analysed features: amount of detected packers, amount of program's sections, entropy of section *.text*, entropy of section *.rest*, and percentage of program's code in section *.rest*.

The null hypothesis reflects the assumption that values detected – low or high – are not related to malicious or harmless origin of samples, so no significant difference in values will be observed in data sets:

Hypothesis 0 (H0): The difference of values measured for analysed feature in harmless software and malicious software is small and insignificant.

Alternative hypotheses reflect the expectation of higher or lower values in malicious samples:

Hypothesis 1 (H1): Analysed feature has higher values in malicious software when comparing to harmless software.

Hypothesis 2 (H2): Analysed feature has lower values in malicious software when comparing to harmless software.

According to hypothesis H2, packing and related features considered typical for malicious software may be detected in harmless software with higher values. Proving the hypothesis for these features may unveil hidden complications in detection mechanisms based on typical malware features.

For statistical analysis, we used two-sample Wilcoxon rank sum test (U test) for comparing the data sets with confidence level 95% ($\alpha = 0.05$).

III. Research Data and Methods

A. Experimental Sets

The set of experimental samples consisted of utility software distributed on the internet for free. Parsons and Oja explain that utility software is a kind of software purposed to assist with monitoring and configuring a computer system and its software [10]. Utility software covers various maintenance tasks, e.g. deleting temporary files, broken links removal, searching for duplicate files, tasks management, memory optimization or personal files encryption. We targeted this specific group of software because of operations that these programs are designated to perform.

We assume that software which legitimately accesses registry entries, processes, file system, etc., may be a promising target for malware writers which create malicious imitations of the original harmless software. With this in mind we aimed at comparing harmless system maintenance tools and their fake malicious counterparts.

We assembled two experimental sets: One containing legitimate applications in a form of executable files (.exe) downloaded from the internet and another containing 100 samples verified as malicious, collected from malware analytic services. We could not obtain all malware .exe files to analyse them on our own, therefore to unify our resources we used reports from analysis of samples, which were available for both sets. Also, we could not obtain exact malicious counterparts of all harmless programs, but nevertheless we preserved the domain of malicious samples in utility software. We obtained malware in the domain of utility software by looking at application name, e.g. "win defrag", which hints on the intended purpose of the sample. We also focused on high amount of detections by anti-virus (AV) engines and manually selected samples which met our criteria. Data fields and their extraction are described in Section III-C.

B. Analytic Tools

We performed our experiments with various kinds of tools:

PEiD is a tool that allows to identify packers which have been used on programs' code [2]. Packing and encrypting libraries are often used by malware for concealment of suspicious parts of a program and evasion of detection based on malware signatures. PEiD performs the search based on signature-like definition of several hundreds of known packers. While it is reliable to detect commonly used packers, it may fail on custom-made packers whose signatures are yet unknown. The original web page of the tool is discontinued and according to reports it may have been taken over by malicious actors. We obtained our copy of the tool with REMnux [3] distribution for malware analysis.

UPX is a packing tool for executable files [4] which is used in the experiment to check whether tested file is packed or not, and to unpack files that are packed. As Davis *et al.* state [11] in their book, numerous computer viruses use specifically UPX packer. A recent case of its malicious usage is presented in blog article by Nick Biasini *et al.* in which cryptocurrency-mining malware *Dark Test* uses UPX as one of its hiding techniques [12]. The packing problem is discussed also in the work of Guo, Ferrie and Chiueh [7]. Therefore, detection of UPX packer being used on analysed sample arises suspicion.

VirusTotal is online malware analysis service. We used it to obtain properties and behaviour of malicious and harmless set of samples [5]. In case of harmless samples it was used for safe and reliable analysis of samples that we collected. In case of malware, since we did not possess original malicious files, we used the service to search for reports from analysis by hash codes of samples that we collected beforehand. Reports generated from analysis contain various information, regarding our experiment e.g. scan results form over 50 anti-virus (AV) solutions, detection of packers by analytic tools F-PROT and PEiD, information from PE header, PE sections with their names and properties, and behavioural information with executed system calls.

Tools for static analysis – PEiD, UPX – run as terminal applications which accept arguments that modify settings and set input and output of analysis. This allowed us to create a

[2]PEiD tool: https://www.aldeid.com/wiki/PEiD
[3]REMnux: https://remnux.org/
[4]UPX: http://upx.sourceforge.net/
[5]VirusTotal: https://www.virustotal.com

helper program which utilised these features for automation of analysis.

C. Experimental Procedure

The experiment was performed in two stages: First we analysed harmless set of programs and evaluated results. In the second stage we proceeded with examining malicious samples. The procedure differed for harmless and malicious samples, mainly because original malicious samples were not available, therefore, the second stage leaned on data provided by analytic reports produced by VirusTotal.

In the first stage we employed our helper program which was developed prior to the experiment for automating the usage of analytic tools. Each harmless sample was analysed by PEiD and UPX to check whether it is packed, and if yes, to identify used packers. Results were collected and summarised in a table. Analysis by VirusTotal followed. We collected produced reports and extracted information of interest.

The second stage regarded malicious samples and employed data extracted from reports of analysis obtained by VirusTotal.

Data obtained from analytic reports comprise the following:

- *Detection results of malware scanners.* In case of positive detection, we obtained name of detection signature, for each scanner separately. We summarised the data as quantity of detections per sample.
- *Detection of packers applied to pack analysed sample.* We acquired names of detected packers and summarised the data as quantity of detected packers per sample.
- *Names of program's sections.* We counted the amount of sections and also stored their names for further manual research. Too few or too many sections comprising the executable file suggest that it was packed, encrypted or otherwise modified in order to disguise original code structure [6].
- *Entropy of program's sections* .text *and* .rest. A section usually named as *.text* or *.code* contains program's instructions. In some occasions parts of code may occur out of usual sections, in so-called *.rest*. Presence of this quasi-section is characteristic for programs modified with some packer. Entropy of these sections may show whether they were modified by packing or encryption, which typically cause entropy to be very high. Values are measured in the interval $< 0,8 >$. Bytes of program's code have some non-random distribution and therefore low entropy. The higher the value, the more random distribution of bytes, suggesting uncommon modifications.
- *The amount of program's code in section* .rest. We calculated percentage amount of bytes in this section. Large portions of code in this section are typical for packed programs.

To objectively evaluate differences between data of malicious and harmless samples we used statistical analysis, specifically two-sample *Wilcoxon rank sum test (U-test)* with confidence level 95% ($\alpha = 0.05$).

[6]More information regarding PE file sections: https://docs.microsoft.com/en-us/windows/desktop/Debug/pe-format

CI	code injection	HG	HTTP GET request
DLL	runtime DLL	HP	HTTP POST request
DNS	DNS request	MC	mutex created
FC	file created	MO	mutex opened
FD	file deleted	PC	process created
FM	file moved	REG	registry entry
FO	file opened	SS	service started
FR	file read	SW	searched window
FW	file written	TCP	TCP data flow
HOOK	hooking activiy	UDP	UDP data flow

TABLE I
BEHAVIOURAL CATEGORIES AND THEIR ABBREVIATIONS.

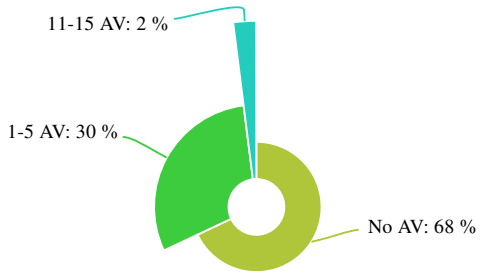

Fig. 1. Pie chart with amount of *harmless samples* that were positively detected by given amount of AV scanners.

From reports provided by VirusTotal we were able to obtain high-level information about behaviour of analysed samples as a list of executed system calls. We used the data in summative way as quantities of operations in behavioural categories listed in Table I. Each sample was then described by 20 numerical values representing occurrences of 20 types of behaviours.

IV. Results and Observations of the Experiment

The following sections present results regarding analysed features and statistical analysis (end of this section, Table II).

A. Detection Results of Malware Scanners

During our experiment VirusTotal employed usually 56 AV scanners but in some cases, for unknown reasons, few of them were unavailable in the report.

Pie charts in Fig. 1 and 2 show amounts of AV scanners (height of a slice) that positively detected analysed samples (their amount as width of a slice), grouped into ranges for improved visual clarity. Among harmless samples no detection (Fig. 1) prevails, but some were detected as threat nevertheless. Among malware samples (Fig. 2) only one had no detection and the rest of them was detected by multitude of AV scanners.

B. Detection of Packer Usage

Pie charts in Fig. 3 and 4 show the amounts of detected packers. Height of a slice represents the amount of packers detected in single sample and width shows the amount of samples detected to be packed with given amount of packers.

Only 20% of *harmless samples* were detected as not packed, the rest was modified by packers ranging from 1 to 7.

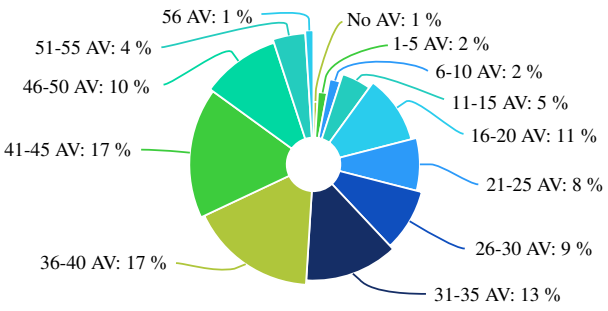

Fig. 2. Pie chart with amount of *malicious samples* that were positively detected by given amount of AV scanners.

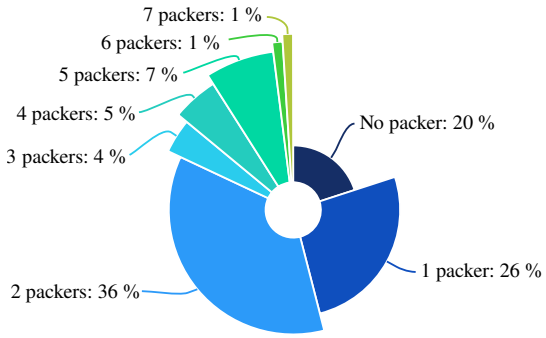

Fig. 3. Pie chart with amount of detected packers in *harmless samples*.

Data from malware samples present more surprises. One malware sample was packed 46 times – that is extreme. However, majority of samples was detected as not packed at all – another surprise, since usual expectations are that malware will be packed massively, not the opposite.

Regarding distribution of amount of packers detected, we can see in Fig. 5 that for *malware* the value of median matches the lower (first) quartile (value 0), and in case of *harmless software* median matches the upper (third) quartile (value 2). From this we can deduce that harmless samples are more prone to being packed, at least with packers that are detectable by available tools. Outliers (extreme values) are not shown in order to prevent plot deformation.

Regarding the hypothesis H1 saying that values of analysed feature – occurrence of packers – is higher in malicious software than in harmless software (Table II, row *Packers amount*, alternative *Higher*), the U-test resulted with p-value > 0.99 which by far exceeds the significance level. As a result, we fail to reject the null hypothesis for this case.

For alternative hypothesis H2 suggesting that occurrence of packers is lower in malicious software, thus prevails in harmless software (Table II, row *Packers amount*, alternative *Lower*), the U-test resulted with p-value 7.3124×10^{-12} which is far below the significance level. As a result, we reject the null hypothesis and accept the alternative hypothesis H2.

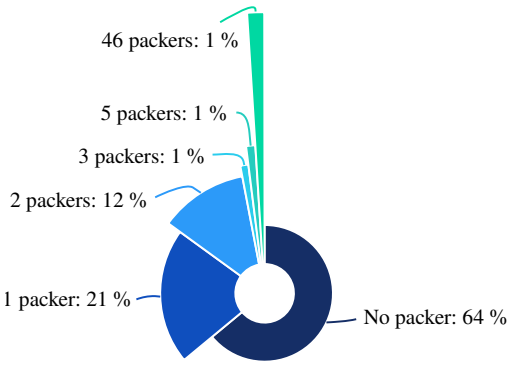

Fig. 4. Pie chart with amount of detected packers in *malicious samples*.

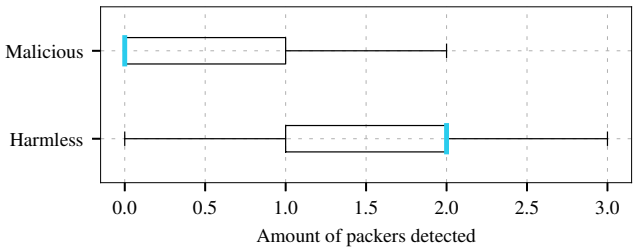

Fig. 5. Boxplots for amounts of packers detected in *harmless* and *malicious* samples. Outliers are not shown in the figure.

C. Amount of Program's Sections

Pie charts in Fig. 6 and 7 show the amounts of detected sections in analysed programs. Height of a slice represents the amount of sections detected in single sample and width shows the amount of samples with given amount of sections. The amount of sections in majority of harmless samples is quite high – 8 – but the amount in malware is lower.

For *harmless samples*, the interquartile range is higher (Fig. 8) – the box is much wider, in comparison to *malware samples*. Both sets of samples match on the first quartile with value 4. There are 6 *harmless samples* with no section detected. This may be caused by different actual file format than PE so the section table could not be retrieved. While several outliers among *malware samples* may make the impression that the amount of sections is high in malware, values of median clearly show that *harmless samples* tend to have higher amount of sections.

For alternative hypothesis H1 saying that amount of sections is higher in malware than in goodware (Table II, row *Sections amount*, alternative *Higher*), the U-test resulted with p-value > 0.99 which by far exceeds the significance level. As a result, we fail to reject the null hypothesis for this case.

For alternative hypothesis H2 saying that amount of sections is lower in malware, so prevails in harmless software (Table II, row *Sections amount*, alternative *Lower*), the U-test resulted with p-value 1.9375×10^{-6} which is far below the significance level. As a result, we reject the null hypothesis and accept the alternative hypothesis H2 – sections prevailing in goodware.

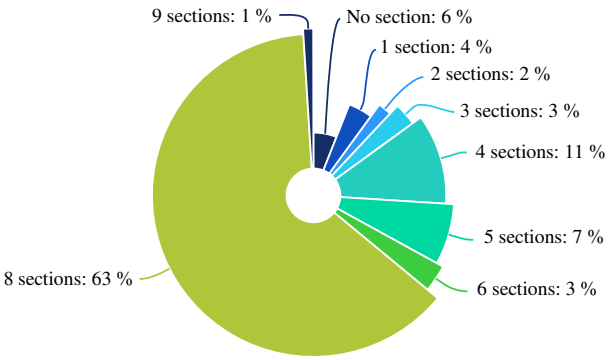

Fig. 6. Pie chart with the amount of detected sections in *harmless samples*.

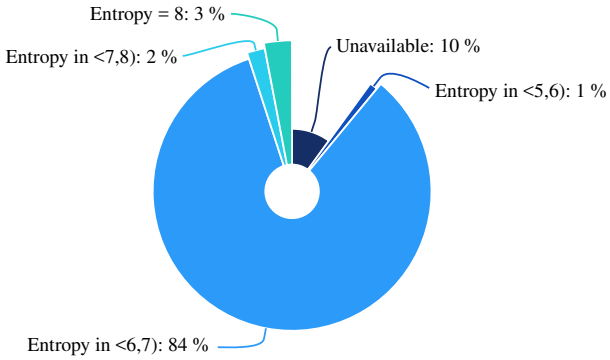

Fig. 9. Pie chart with entropy of section *.text* in *harmless samples*.

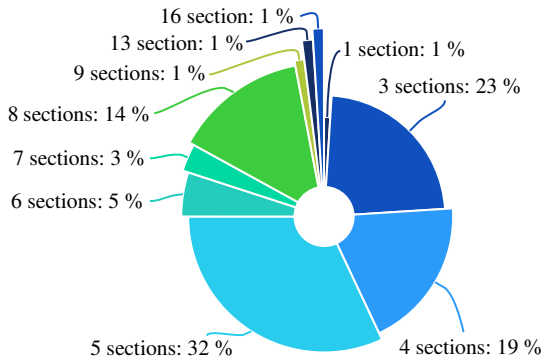

Fig. 7. Pie chart with the amount of detected sections in *malware samples*.

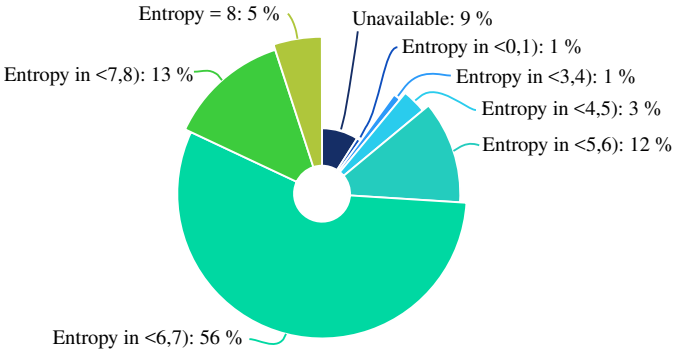

Fig. 10. Pie chart with entropy of section *.text* in *malware samples*.

D. Entropy of section .text

Section *.text* (or *.code*) contains executable instructions, therefore its entropy should not be normally very high.

Pie charts in Fig. 9 and 10 show measured entropies. Height of a slice represents range of values of entropy and width shows the amount of samples with given entropy value. We can see that both malicious and harmless group have majority of samples with entropy in range $< 6, 7)$. In *harmless samples*, other ranges of entropy occur seldom – only in 6 samples – and the rest comprises samples in which the section could not be precisely identified. *Malware* shows wider variety of entropy, both on the lower and higher spectrum of value ranges.

Boxplot (Fig. 11) shows that inter-quartile range is wider in *malware* and extremes are much more apart. Medians, however, are close to 6.5 in both malicious and harmless samples. This suggests that the difference in values may not be significant regarding *.text* section entropy.

For alternative hypothesis H1 that values of *.text* section entropy are higher in malware than in goodware (Table II, row *.text entropy*, alternative *Higher*), the U-test resulted with p-value 0.6556 which by far exceeds the significance level. We fail to reject the null hypothesis for this case.

For alternative hypothesis H2 that values of *.text* section entropy are lower in malware (Table II, row *.text entropy*, alternative *Lower*), the U-test resulted with p-value 0.3453 which also exceeds the significance level. We fail to reject the null hypothesis and conclude that differences in values of entropy of section *.text* are not statistically significant.

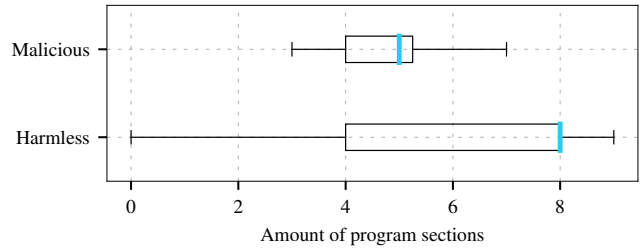

Fig. 8. Boxplots for amount of program's sections in *harmless* and *malicious* samples. Outliers are not shown in the figure.

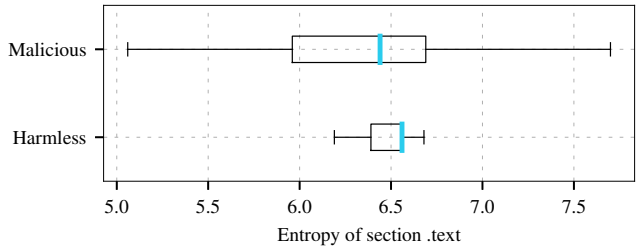

Fig. 11. Boxplots for entropy of section *.text* in *harmless* and *malicious* samples. Outliers are not shown in the figure.

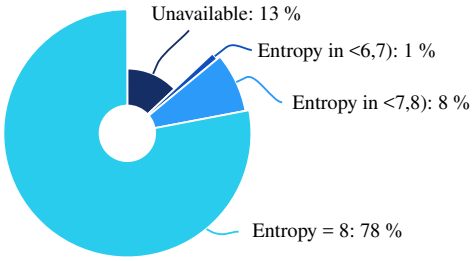

Fig. 12. Pie chart with entropy of section *.rest* in *harmless samples*.

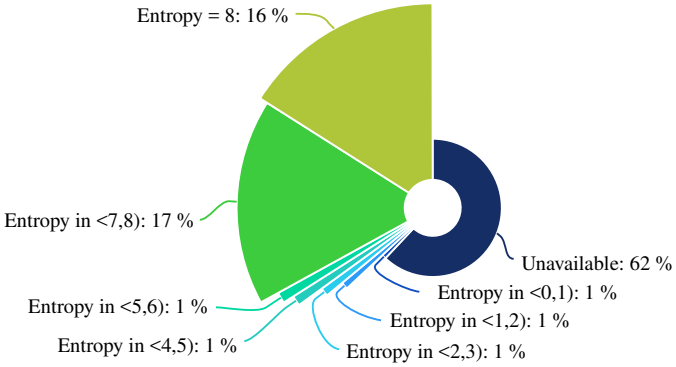

Fig. 13. Pie chart with entropy of section *.rest* in *malware samples*.

E. Entropy of section .rest

Since *.rest* is not a program section *per se*, it may not be detected in many programs. On the other hand, its presence suggests that special measures have been made to conceal (at least from plain sight) some incriminating portions of code. It comes naturally to assume presence of *.rest* in majority of malware samples, but looking at our previous experiments, assumptions about malware can be quite misleading.

Pie charts in Fig. 12 and 13 show measured entropy of section *.rest*. Height of a slice represents range of values of entropy and width shows the amount of samples with given entropy value. In case of *harmless samples*, 13 programs had no detectable code out of common sections but the rest of them had, with measured entropy from 6 to 8 – which is a maximum. Contrary to that, *malware samples* showed no section *.rest* in majority of cases (62 samples), and in case of its presence, entropy reached value from 6 to 8 only in 33 samples.

Lower quartile, upper quartile and median for *harmless samples* meet at value 8 (Fig. 14) and contrast with values of malware – it has median at zero, mostly due to absence of the section.

For alternative hypothesis H1 that values of *.rest* section entropy are higher in malware than in harmless software (Table II, row *.rest entropy*, alternative *Higher*), the U-test resulted with p-value > 0.99 which by far exceeds the significance level, so we fail to reject the null hypothesis.

For alternative hypothesis H2 that values of *.rest* section entropy are lower in malware (Table II, row *.rest entropy*, alternative *Lower*), the U-test resulted with p-value 2.7074×10^{-18} which is far below the significance level. We reject the null hypothesis and accept the alternative hypothesis H2 that *.rest* section entropy is lower in malware than in goodware.

F. Percentage of code in section .rest

Percentages that were found among analysed *harmless samples* are shown in Fig. 15 in ascending order.

Surprisingly, large programs' size present in section *.rest* prevails in *harmless software*. We suppose that it may be inflicted by some commonly used packers and application building tools, such as INNO. However, further research needs to be made to confirm this opinion.

Examination of *malware* showed that *.rest* manifested in much fewer samples than in the harmless set. This result corresponds with findings from analysis of entropy of section *.rest* (Sec. IV-E). The percentage of *.rest* section in file's size (Fig. 16) was calculated with data obtained from VirusTotal analytic reports.

62 *malware* samples contained no data outside sections listed in PE file header and so section *.rest* was confirmed to be absent in them. The case of 90% or more of program's bytes was present only in 11 samples. Again, this strongly contrasts with results from harmless samples.

Figure 17 shows that for *harmless samples* the first and the third quartile have high values and samples with less than 80 % percent of code in *.rest* are basically outliers. Quartiles of malware data contrast to that as low – the first quartile and median match at value 0. The boxplot suggests notable differences in values of malicious and harmless samples.

For alternative hypothesis H1 that percentages of code in *.rest* section are higher in malware than in goodware (Table II, row *.rest percentage*, alternative *Higher*), the U-test resulted with p-value > 0.99 which by far exceeds the significance level. As a result, we fail to reject the null hypothesis.

For alternative hypothesis H2 that percentages of code in *.rest* section are lower in malware (Table II, row *.rest percentage*, alternative *Lower*), the U-test resulted with p-value 3.2413×10^{-17} which is far below the significance level. As a result, we reject the null hypothesis and accept the alternative hypothesis H2 about percentage of code in section *.rest* being lower in malware than in harmless software.

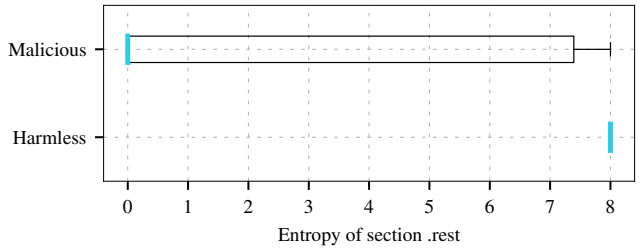

Fig. 14. Boxplots for entropy of section *.rest* in *harmless* and *malicious* samples. Outliers are not shown in the figure.

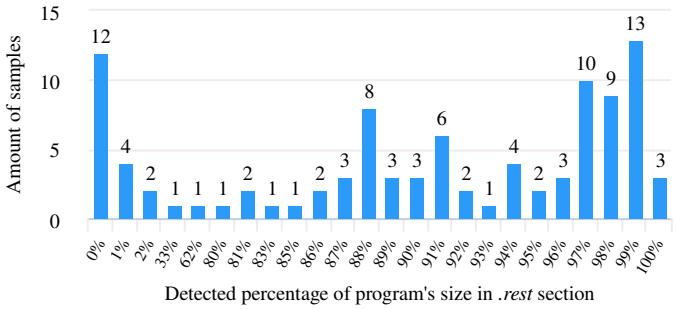

Fig. 15. Measured percentages of *.rest* section from total size of code in *harmless samples*, and number of samples with that percentage of code in *.rest* section.

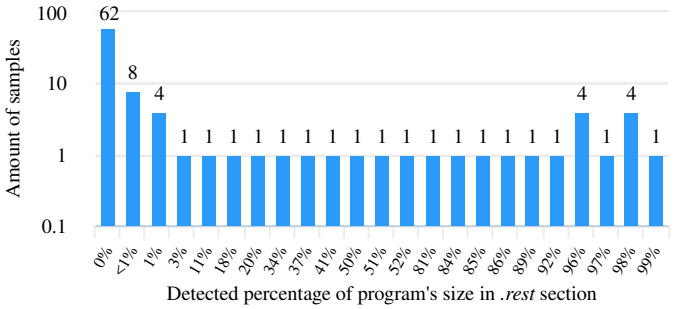

Fig. 16. Measured percentages of *.rest* section from total size of code in *malicious samples*, and number of samples with that percentage of code in *.rest* section. The y-axis with amounts of samples is scaled logarithmically due to great differences among values.

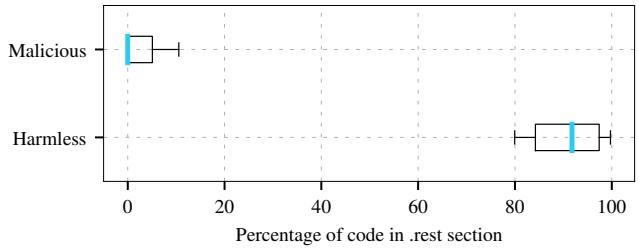

Fig. 17. Boxplots for percentage of program's code in section *.rest* in *harmless samples* and *malicious samples*. Outliers are not shown in the figure.

Data sets			
Malicious	**Harmless**	**Alternative**	**Result**
Packers amount		Lower	7.3124×10^{-12}
Packers amount		Higher	≈ 1
Sections amount		Lower	1.9375×10^{-6}
Sections amount		Higher	≈ 1
.text entropy		Lower	0.3453
.text entropy		Higher	0.6556
.rest entropy		Lower	2.7074×10^{-18}
.rest entropy		Higher	≈ 1
.rest percentage		Lower	3.2413×10^{-17}
.rest percentage		Higher	≈ 1

TABLE II
SUMMARY OF RESULTS FROM STATISTICAL ANALYSIS MADE WITH *Wilcoxon rank sum test (U-test)* WITH CONFIDENCE LEVEL 95% ($\alpha = 0.05$). THE ALTERNATIVE DESCRIBES RELATION BETWEEN VALUES OF MALICIOUS AND HARMLESS SET OF SAMPLES.

G. Relationships Between Analysed Features

We explored also relationships between analysed features by calculating *Pearson's correlation coefficients* for each pair of features that were discussed in statistical analysis in previous section. The measure of correlation is described by values from interval $\langle -1, 1 \rangle$, with the following interpretation:

- value -1 represents perfect negative correlation, i.e. when values of one feature are high, second feature's values are low, and vice versa,
- value 0 represents no measurable linear correlation,
- value 1 represents perfect positive correlation, i.e. when value of one feature is high, so is value from the second feature, and vice versa.

Visual representation of correlation matrix as a "heatmap" (Fig. 18, 19) can aid in understanding notable linear relationships between features. We can see that in heatmaps of both malicious and harmless samples only positive correlations were present. For *harmless samples* the most notable correlation is between amount of detected sections and entropy of section *.rest*. It seems that when packers are used for hiding program's code the amount of sections is decreased and large portions of code are then in section *.rest*. We explored the issue further and discovered that it occurred mainly when a sample was packed by packer *UPX*. If it was the only packer used, it placed all the code into section named *.upx*.

Another notable correlations in *harmless samples* were between amounts of sections and detected packers, and between percentage of code in section *.rest* and entropy of this section, but values for these correlations are not that high.

Regarding *malicious samples*, only one correlation is notable and it was measured between values of percentage of code in section *.rest* and entropy of this section. This relates to finding that numerous malware samples do not have this section so presence of this section with some entropy will cause this correlation to be high.

H. High-Level Behaviours

Beside features related to packing we recorded and measured also quantity of executed system calls which belong to behavioural categories listed in Table I. After observing the values we realised that much more possibilities of analysis are opening ahead of us and thus will require further work. However, to supplement findings from previous sections with at least several interesting insights about behaviour, we analysed correlations between pairs of behaviours and created heatmaps for malicious (Fig. 20) and harmless samples (Fig. 21).

First notable thing is that *harmless samples* had no occurrence in behavioural categories *CI - code injection* and *FC - file created*, therefore lines for these features are blank in Fig. 21.

We can see that heatmap of harmless samples contains more high correlations than heatmap of malware samples. Some of

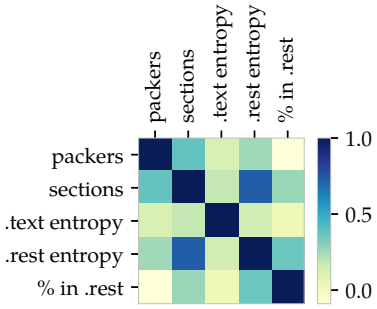

Fig. 18. Correlation matrix heatmap for properties related to packing in *harmless samples*.

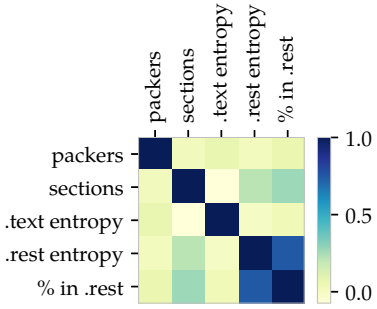

Fig. 19. Correlation matrix heatmap for properties related to packing in *malicious samples*.

them match but for *harmless samples*, these high correlations between pairs of features are unique:

- *FM - file moved* and *FD - file deleted*,
- *FO - file opened* and *FM - file moved*,
- *FO - file opened* and *FD - file deleted*,
- *FW - file written* and *FM - file moved*,
- *FW - file written* and *FD - file deleted*,
- *HG - HTTP GET request* and *DNS - DNS request*,
- *MO - mutex opened* and *MC - mutex created*,
- *SW - searched window* and *MC - mutex created*,
- *TCP - TCP data flow* and *DNS - DNS request*.

It seems that for *harmless samples* operations related to files are often performed and quantitatively relate to each other.

For *malicious samples*, interesting correlations are between following features:

- *MC - mutex created* and *FD - file deleted*,
- *MO - mutex opened* and *DLL - runtime DLL*,
- *REG - registry entry* and *FO - file opened*,
- *REG - registry entry* and *FR - file read*,
- *SS - service started* and *FD - file deleted*,
- *UDP - UDP data flow* and *HG - HTTP GET request*.

Malware samples seem to be focused more on mutex, registry and service operations and they are performed similarly often as various operations with files, mainly opening, reading and deleting.

I. Summary

Entropy of section *.text* was notably high in many malicious and harmless samples, so it seems that concealment is targeted

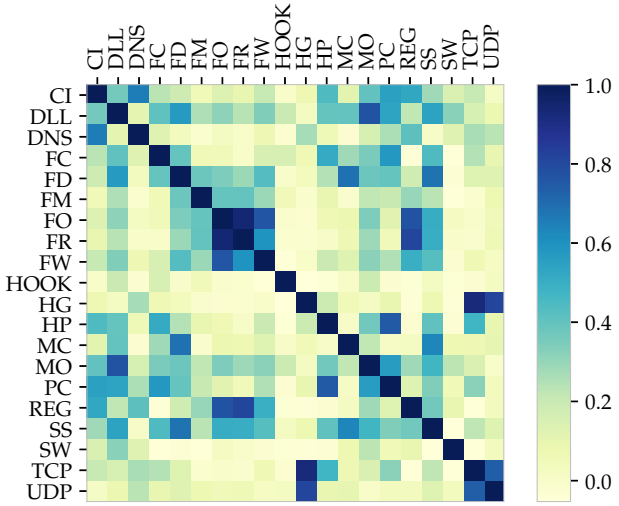

Fig. 20. Correlation matrix heatmap for high-level behaviours of *malicious samples*.

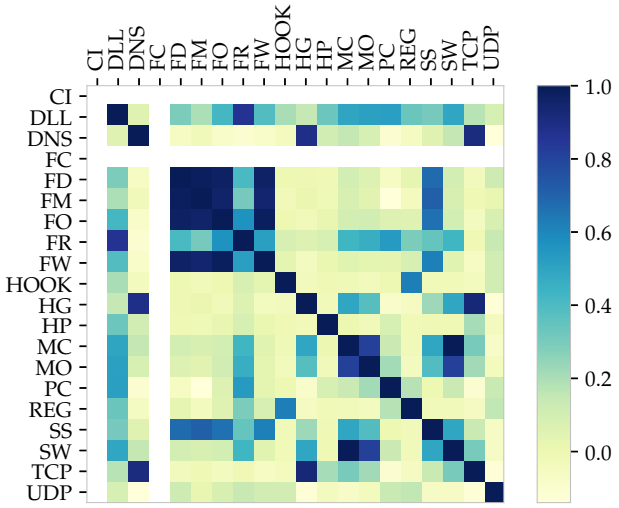

Fig. 21. Correlation matrix heatmap for high-level behaviours of *harmless samples*.

at program's section with executable instructions. Difference between sets resulted as insignificant, but lower amount of detected packers in malware suggests that custom packers are used, which are undetectable by common analytic tools. Section *.rest* was not detected in majority of malware samples. This suggests that malware writers use custom code concealing techniques that do not create this quasi-section.

Regarding the assumptions about malware being packed, they are probably true, however, common tools seem now insufficient for malware packers detection. Information pointing at programs being packed need to be collected from deeper levels of analysis: Amounts and names of packers provided by analytic tools may be incomplete, but dissecting a program into sections and observing their entropy shows the missing

pieces of information.

Correlations between high-level behaviours hint some interesting relationships between types of operations performed during execution of programs and it would be interesting to analyse specific cases of behaviours that are executed together. Also, it would be beneficial to look at behaviours of malware and harmless software from another software domain and examine if there are some repeating patterns present in values.

Insights obtained from experiments that we performed are significant regarding research and implementation of novel detection systems based on machine learning and neural networks, since they utilise quantitatively measurable features of software samples. Our results present options for feature engineering to obtain high-quality training data: Neglecting features that are insignificant or combining closely related features to one for reducing dimensionality of data. Mindful selection of training data will allow to direct more attention and time to selection and experimentation with detection algorithms.

Based on our work new research questions arise. Behavioural and non-behavioural properties of programs vary greatly and as we "zoom in" to specific software domain of malicious and harmless samples, we see that usual assumptions cease to be applicable. For example, results of signature-based detection of packers do not agree with expectations towards malware and goodware (Sec. IV-B) and are therefore unsuitable for machine learning purposes. Clearly, malware writers made it harder to detect packers, so deeper analysis of samples is required to obtain desired data. This opens up a question whether now-so-popular utilisation of massive data is not in fact a limitation in improving malware detection, and we ought to pursue more specific and narrow features for differentiating malware from goodware instead.

V. Discussion

Beside reports from analysis obtained from VirusTotal we used tools of static analysis in this experiment for the following reasons:

1) Although dynamic analysis compared with static analysis indicates smaller issues with malware obfuscation, static analysis allows to detect malicious features which seem to occur randomly during a program execution or in a specific execution environment.
2) We could automate the analytic process for large amount of experimental samples in our custom created utilities.
3) Automated process of static analysis is less time consuming compared with dynamic analysis, which requires execution of each analysed sample.

Concerning point 1, every program can comprise numerous execution paths, also called *execution traces*. The disadvantage of dynamic analysis is that only one execution trace can be observed at a time. Concerning several traces, also static analysis with reverse engineering is problematic. However, Beaucamps *et al.* addressed this problem in their work [13] and proposed a method for static analysis of execution traces acquired from control-flow graphs. Macedo and Touili also discuss the issue in their work [14].

UPX packer is commonly known as packing tool often misused for covering malicious code. Marak states, however, that obfuscating effects of the packer are not among its original features and result from altering its original code [15]. A look at licence of UPX packer reveals that modifications and usage of the packer in such way is violating the rules of tool's legal usage [7]. In fact, UPX packer should not be suspicious by itself, like many blogs and papers state, but the illegal modifications made to it are what causes trouble. This fact should be given more attention and researchers should avoid improper simplifications of the matter.

In some cases, PE sections names may reveal name of a packer used for concealing program's code, however, this information is not fully reliable since section names can be modified by various tools, e.g. by PE Explorer [8]. In our observations we also encountered section names being some gibberish or blank—obviously someone removed the original section name on purpose.

A. Related Work

Malware signatures have still very important role in malware detection, although their effectiveness on malicious samples concealed by techniques that alter syntactic form of a program is questionable. What is more, with rapidly growing number of new malware samples the extraction of signatures requires a lot of precious time. Griffin *et al.* addressed this problem and presented a system for automated generation of malware signatures [16]. An interesting part of their work describes features which they analysed in malicious programs. Concerning syntactic form of a program authors mention patterns emerging in operational code which may represent precursors of non-standard or suspicious behaviour of the program:

- Constant values like IP addresses, email addresses,
- access to memory with unusual offset,
- local function calls, non-library function calls, context of a function call and used parameters,
- suspicious mathematical operations which may indicate obfuscation.

These patterns are used for refining potential malware signatures through, as they call it, *code interestingness check*. In our research they served as an inspiration for comparing features of malicious and harmless programs and looking for patterns which could be employed as indicators of malicious intentions.

B. Influences on the Study and its Outcomes

Results of our research showed that for special-purpose software packing may be detected more often in harmless samples than in malicious samples, which is in contrast with common assumptions about malware. Nevertheless, several factors could have affected the outcomes even when we made en effort to mitigate them as best we could.

[7]UPX licence: https://upx.github.io/upx-license.html
[8]PEexplorer: http://www.heaventools.com/PE_Explorer_section_editor.htm

- *Selection of samples.* Commercial paid software was not included in the study but its properties may have been different from what we found in harmless samples. However, obtaining numerous samples of paid maintenance software was not feasible in our research project. Concerning malware samples, it is hard to trace their origin since we worked just with reports from their analysis, not with samples directly. This may have also considerable effect.
- *Collection of samples.* Samples were found on the internet by search engine with specific keywords. Different keywords may have led to different outputs, even when we tried to explore as many various results as we could.
- *The usage domain.* Samples that we experimented with belong to system utility software. Samples from different domains may have different properties regarding packing.
- *Analytic tools.* The tools that we used in our study to gather data of interest do not guarantee 100% correctness of data. There is a chance that detecting fewer packers among malicious samples is caused by inability of tools to unveil usage of hidden, sophisticated custom packer developed by malware creators. This problem, however, is not in our power to mitigate.
- *Other errors.* Several samples had no program sections detected. This may have been caused by an unknown error during analysis performed by tools we used or by difference of actual file format from the format declared by the sample.

VI. Conclusion

We presented a different, novel approach to malware research that is based on narrow selection of experimental samples from specific software domain and statistical evaluation of differences between malicious and harmless software. Several ideas inspired us to perform this experiment:

1) Packing is often applied to malicious software with intent to obstruct reverse-engineering, hinder static analysis, and hide incriminating code from malware detectors.
2) Although packing is typical for malware, it may be used also on harmless software for protection against intellectual property theft.
3) In research circles, a discussion about distinguishing malicious packing from harmless one regards syntactic features of program's operational code, e.g. bytes distribution, entropy, data in so-called *.rest* section.
4) Harmless programs have not been given appropriate attention, especially from the context of features relevant for distinguishing between malicious and harmless case of packing, and their reliability.

Although packers are massively used by malware creators, they are also applied for protection of intellectual property in harmless software, making it complicated to separate bad and good intentions behind packer's usage.

In the paper we showed that differences in values between malicious and harmless programs are significant regarding amount of detected packers, amount of program sections, percentage of code in section *.rest* and its entropy. Entropy of section *.text* together with amounts of packers detected suggest that malware writers create custom packers that are nearly undetectable by common analytic tools.

It is necessary to keep in mind that results presented here concern samples from the domain of maintenance and utility software and samples from other domains may yield different results. In that case, however, it would be interesting to research the influence of software domain selection on values of analysed features, since it may be significant.

References

[1] J. Šťastná and M. Tomášek, "Exploring malware behaviour for improvement of malware signatures," in *IEEE 13th International Scientific Conference on Informatics, 2015*, Nov 2015. doi: 10.1109/Informatics.2015.7377846 pp. 275–280.

[2] J. Šťastná and M. Tomášek, "The problem of malware packing and its occurrence in harmless software," *Acta Electrotechnica et Informatica*, vol. 16, no. 3, pp. 41–47, 2016. doi: 0.15546/aeei-2016-0022

[3] T.-Y. Wang and C.-H. Wu, "Detection of packed executables using support vector machines," in *International Conference on Machine Learning and Cybernetics (ICMLC), 2011*, vol. 2, 2011. doi: 10.1109/ICMLC.2011.6016774. ISSN 2160-133X pp. 717–722.

[4] S. Josse, "Secure and advanced unpacking using computer emulation," *Journal in Computer Virology*, vol. 3, no. 3, pp. 221–236, 2007. doi: 10.1007/s11416-007-0046-0

[5] M. Šipoš and S. Šimoňák, "Rasp abstract machine emulator – extending the emustudio platform," *Acta Electrotechnica et Informatica*, vol. 17, no. 3, pp. 33–41, 2017. doi: 0.15546/aeei-2017-0024

[6] G. Jacob, P. Comparetti, M. Neugschwandtner, C. Kruegel, and G. Vigna, "A static, packer-agnostic filter to detect similar malware samples," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. LNCS, vol. 7591. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-37300-8_6. ISBN 978-3-642-37299-5 pp. 102–122.

[7] F. Guo, P. Ferrie, and T.-c. Chiueh, "A study of the packer problem and its solutions," in *Recent Advances in Intrusion Detection*, ser. LNCS, vol. 5230. Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-87403-4_6. ISBN 978-3-540-87402-7 pp. 98–115.

[8] A. Singh and A. Lakhotia, "Game-theoretic design of an information exchange model for detecting packed malware," in *6th International Conference on Malicious and Unwanted Software (MALWARE), 2011*, 2011. doi: 10.1109/MALWARE.2011.6112319 pp. 1–7.

[9] P. Arntz. Analyzing malware by api calls. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2017/10/analyzing-malware-by-api-calls/

[10] J. Parsons and D. Oja, *New Perspectives on Computer Concepts 2013: Comprehensive*, ser. New Perspectives. Cengage Learning, 2012. ISBN 9781133190561

[11] M. Davis, S. Bodmer, and A. LeMasters, *Hacking exposed malware and rootkits*. New York: Mc-Graw Hill, 2010. ISBN 978-0-07-159119-5

[12] N. Biasini, E. Brumaghin, W. Mercer, and J. Reynolds. Ransom where? malicious cryptocurrency miners takeover, generating millions. [Online]. Available: http://blog.talosintelligence.com/2018/01/malicious-xmr-mining.html

[13] P. Beaucamps, I. Gnaedig, and J.-Y. Marion, "Abstraction-based malware analysis using rewriting and model checking," in *Computer Security - ESORICS 2012*, ser. LNCS, vol. 7459. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-33167-1_46. ISBN 978-3-642-33166-4 pp. 806–823.

[14] H. Macedo and T. Touili, "Mining malware specifications through static reachability analysis," in *Computer Security - ESORICS 2013*, ser. LNCS, vol. 8134. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-40203-6_29. ISBN 978-3-642-40202-9 pp. 517–535.

[15] V. Marak, *Windows Malware Analysis Essentials*. Packt Publishing, 2015. ISBN 9781785287633

[16] K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh, "Automatic generation of string signatures for malware detection," in *Recent Advances in Intrusion Detection*, ser. LNCS, vol. 5758. Springer Berlin Heidelberg, 2009. doi: 10.1007/978-3-642-04342-0_6. ISBN 978-3-642-04341-3 pp. 101–120.