IJoFCS (2011) 1, 44-58

e-forensic PRESS

The International Journal of
FORENSIC COMPUTER SCIENCE

www.IJoFCS.org

DOI: 10.5769/J201101003 or http://dx.doi.org/10.5769/J201101003

BinStat
Tool for Recognition of Packed Executables

Kil Jin Brandini Park(1), Rodrigo Ruiz(2), Antônio Montes(3)

Divisão de Segurança de Sistemas da Informação (DSSI)
Centro de Tecnologia da Informação Renato Archer (CTI)
Campinas – SP, Brasil.
(1) kil.park@cti.gov.br
(2) rodrigo.ruiz@dssi.cti.gov.br
(3) antonio.montes@cti.gov.br

Abstract - The quantity of malicious artifacts (malware) generated by the combination of unique attack goals, unique targets and various tools available for the developers, demands the automation of prospecting and analysis of said artifacts.
Considering the fact that one problem handled by experts in analysis of executable code is packing, this paper presents a method of packing detection through the appliance of statistical and information theory metrics.
The tool developed in this study, called BinStat, generated a high recognition rate of executable packing status within the test samples, proving its effectiveness.

Keywords - Packing, Packed Executables, Malware Analysis

I. Introduction

With the advent of the Internet and the resulting offering of sensitive online services, the action of malicious artifacts was raised to a new level. The breach of data confidentiality became more widely persecuted for its financial potential gain. The complexity of those artifacts follows the trend of growth of available online services, through the application of more refined techniques of attack and obfuscation.

A sector greatly affected by criminal activity is that of online banking. Data presented by [1] indicates that in 2009, the Brazilian banks losses from online fraud reached the figure of nine hundred million dollars. Moreover, [2] says that Brazil is a major source of malicious artifacts of trojan type aiming at internet banking activity. Corroborating this idea of brazilian leadership in malware production, [3] presents data that points Brazil as the source of four percent of all new malicious artifacts captured in the world.

Migrating from public to private sector, one can observe an intense movement of the major world powers in order to build safeguards against

cyber attacks aimed at critical national infrastructure. The raise of the worm Stuxnet which, according to [4], [5] and [6] was developed with the specific intention of maiming Iran's nuclear program as its target was the embedded control components of nuclear centrifuges, was considered by many information security experts as cyber war´s first movement.

The quantity of malware generated by the combination of unique attack goals, unique targets and various tools available for the developers, demands the automation of prospecting and analysis of said artifacts. Thus, the development of tools that extract information from all stages of executable analysis be it static or dynamic becomes essential.

One type of tool that can be used by developers is the Packer. These may, in addition to obfuscate the source code from an executable, reduce its size. Therefore, developers can use them with the legitimate purposes of reducing the space occupied by a program as well as safeguarding their intellectual property.

However, regarding the development of malware, packers are used in order to circumvent the mechanisms of recognition on signatures based antivirus and hinder or prevent access to malware source code, employing various methods such as multi-layered packing and anti-unpacking techniques: anti-dumping, anti-emulating and anti-debugging [7].

So one of the issues to be addressed by researchers who wish to carry out the analysis of the source code of a particular executable is exactly to check whether it is packed or not.

Thus, this paper discusses the development of a tool named BinStat, which aims to, through analysis of statistics and information theory formulas, sort executables as packed or unpacked.

This paper presents the following structure:

In Packing Recognition Methods, we present various packing recognition methods, including the one used in the development of the BinStat tool.

In Application Architecture, we discuss the architecture of BinStat and the implementation characteristics of each of its modules.

In Preliminary Results, we present a comparison between the statistics and information theory formulas calculated for a given executable and a packed version of the same executable.

In Results and Discussion, we analyze the results generated in the development and testing of BinStat.

Finally, follows the conclusions and references used.

II. Packing Recognition Methods

Some tools available for checking the packing status of binaries, such as PEiD [8], apply the methodology of verification of packing through signatures.

If on the one hand the use of signatures not only permits the assessment of the packing status of the executables but also the tool used for that, on the other hand, the technique does not recognize the tools that are not yet available in its database of signatures as [9] and [10] show. In addition to that, recognition through signatures is subject to attempts of fraud such as the case of packing tools that mask their signature, hiding it or making it similar to other tools.

Another technique presented by [10] suggests the adoption of tracking of operations performed in memory using an emulated environment. The main idea is to control write operations performed on a given region of memory used by a process running on an emulated environment. A region of memory that was written at runtime is marked as "dirty."

To be executed, the malware original obfuscated code must undergo some kind of transformation and be written in a given memory address, that will store the original unpacked instructions, and those instructions will eventually be send to execution. Therefore, “dirty” memory regions which stores data sent for execution can

contain only multi-level packing instructions or obfuscated original instructions.

The disadvantage of this methodology lies in the need to run the executable to be analyzed inside virtual machines.

Another possibility is the use of statistics and information theory formulas in order to extract information about the binary files of executables, seeking to construct a decision tree used to classify them as packed or not. This methodology was applied to this work and was also used by [11] in an attempt to identify executables with malicious behavior, showing promising results.

III. Application Architecture

The application is divided into four modules, as Figure 1:

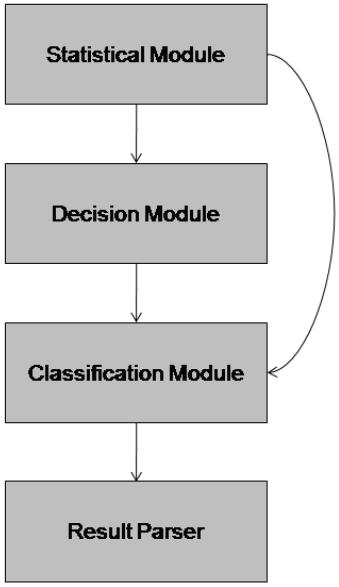

Figure 1. Application Modules

The application works in two ways. In the training phase, every single executable will be packed and both versions, the original unpacked and packed, will serve as input for the statistical module which in turn feeds the decision module, where the decision tree will be trained.

Once the tree is built, it is passed to the classification module, which in turn will then be able to receive requests for classification of executables with unknown packing status.

In the production phase, the decision module does not work. Thus, calculations generated by the statistical module about the executable to be analyzed are passed to the classification module, the result of which feeds the parser module, responsible for formatting the output in a predetermined manner convenient for further processing.

A. Statistical Module

The statistical module segments the input binary artifact in blocks of 1024 bytes. For each block, the number of occurrences and frequency histogram for all possible values of of unigram, bigram, trigram or four-gram will be calculated, i.e. the occurrences and frequency histogram of the values found for the combinations of one byte (0-255), two bytes (0 - 65,535), three bytes (0-16777215) and four bytes (0 - 4,294,967,295) will be calculated.

This frequency histogram is used as input for thirteen statistical and information theory calculations [12]:

1) Simpson´s Index

$$S_{b_i} = \frac{\sum_{f=0}^{n} k_f \left(k_f - 1\right)}{N\left(N-1\right)}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, k_f is the number of occurrences of the n-gram of value f inside the block and N is the total number of bytes of the block. It is important to note that the last block of an executable to be analyzed may not have all the 1024 bytes.

2) Canberra´s Distance

$$CA_{b_i} = \sum_{f=0}^{n} \frac{\left|X_f - X_{f+1}\right|}{\left|X_f\right| + \left|X_{f+1}\right|}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the

maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

3) Minkowski´s Order of Distance

$$M_{b_i} = \sqrt[\lambda]{\sum_{f=0}^{n} \left| X_f - X_{f+1} \right|^{\lambda}}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1. The adopted value of λ is 3 as defined in [11].

4) Manhattan´s Distance

$$MH_{b_i} = \sum_{f=0}^{n} \left| X_f - X_{f+1} \right|$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

5) Chebyshev´s Distance

$$CH_{b_i} = max_f \left| X_f - X_{f+1} \right|$$

Where b_i refers to the ith block of the executable to be analyzed, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

6) Bray Curtis´s Distance

$$BC_{b_i} = \frac{\sum_{f=0}^{n} \left| X_f - X_{f+1} \right|}{\sum_{f=0}^{n} \left(X_f + X_{f+1} \right)}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

7) Angular Separation

$$BC_{b_i} = \frac{\sum_{f=0}^{n} \left| X_f - X_{f+1} \right|}{\sum_{f=0}^{n} \left(X_f + X_{f+1} \right)}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

8) Correlation Coefficient

$$CC_{b_i} = \frac{\sum_{f=0}^{n} \left(X_f - \overline{X}_{b_i} \right) \cdot \left(X_{f+1} - \overline{X}_{b_i} \right)}{\left(\sum_{f=0}^{n} \left(X_f - \overline{X}_{b_i} \right)^2 \cdot \sum_{f=0}^{n} \left(X_{f+1} - \overline{X}_{b_i} \right)^2 \right)^{1/2}}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_{bi} refers to the mean frequency of n-grams in block b_i, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

9) Entropy

$$E(R) = -\sum_{v \in \Delta_n} t(r_v) \log_2 t(r_v)$$

Where Δ_n represents the images, i.e., all valid values for a given n-gram and $t(r_v)$ is the frequency of the v-value n-gram.

10) Kullback – Leibler´s Divergence

$$KL_{b_i}\left(X_f \parallel X_{f+1} \right) = \sum_{f=0}^{n} X_f \log \frac{X_f}{X_{f+1}}$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram

of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

11) Jensen-Shannon´s Divergence

$$JSD_{b_i}\left(X_f \parallel X_{f+1}\right) = \frac{1}{2} D\left(X_f \parallel M\right) + \frac{1}{2} D\left(X_{f+1} \parallel M\right)$$

Where

$$M = \frac{1}{2}\left(X_f + X_{f+1}\right)$$

Where b_i refers to the ith block of the executable to be analyzed, D refers to Kullback – Leibler Divergence, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

12) Itakura – Saito´s Divergence

$$BF_{b_i}\left(X_f, X_{f+1}\right) = \sum_{f=0}^{n}\left(\frac{X_f}{X_{f+1}} - \log\left(\frac{X_f}{X_{f+1}}\right) - 1\right)$$

Where b_i refers to the ith block of the executable to be analyzed, n reaches the maximum value of the n-gram to be analyzed less one, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

13) Total Variation

$$\delta_{b_i}\left(X_f, X_{f+1}\right) = \frac{1}{2}\sum_f \left|X_f - X_{f+1}\right|$$

Where b_i refers to the ith block of the executable to be analyzed, X_f refers to the frequency of the n-gram of value f, X_{f+1} refers to the frequency of the n-gram of value f + 1.

In the training phase, for each of the n-grams (unigram up to four-gram) used this module calculates the thirteen measures presented for each block that composes the executables pertaining to the training base. This information is then passed to the decision module for the construction of a decision tree for each of the n-grams.

In the production phase, the result generated by the statistical module will be passed on to the classification module previously powered by the decision trees generated during the training phase.

B. Decision Module

To implement the decision module, applying the techniques of decision tree building, we used the public source code versions of the C5.0/See5 tools, whose operations are described in [13].

C. Classification Module

The classification module takes as input the calculations of each of the blocks that compose the executable to be tested and the decision trees generated by the decision module. Note that one must decide which n-gram will be tested, as there is a decision tree for each of the n-grams used.

Based on these data, the module classifies each block as packed ("yes") or unpacked ("no").

D. Result Parser

The parser module receives the result generated by the decision module and formats it in order to provide a format better suited for further use.

It is important to note that this module can be adapted, enabling application integration with other mechanisms.

IV. Preliminary results

As a pre assessment of the calculations that compose the statistical module, a specific executable (identified by the md5 e4a18adf075d 1861bd6240348a67cce2) was selected and packed with UPX packer (identified by the md5 745528339c38f3eb1790182db8febee1).The original application and its packed version were used as input for this module. The distributions of normalized results (for values in the range 0 to 1) were then compared graphically:

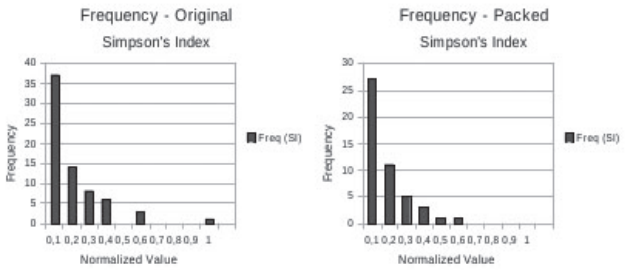

Figure 2. Comparison of the Frequency of the Normalized Values for Simpson´s Index Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

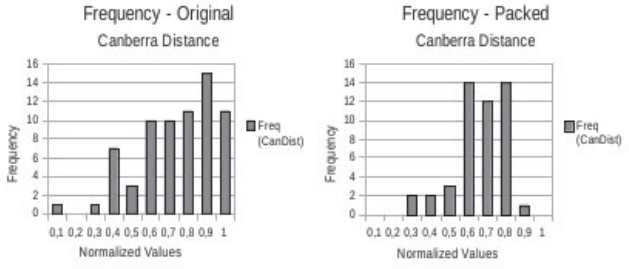

Figure 3. Comparison of the Frequency of the Normalized Values for Canberra´s Distance Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

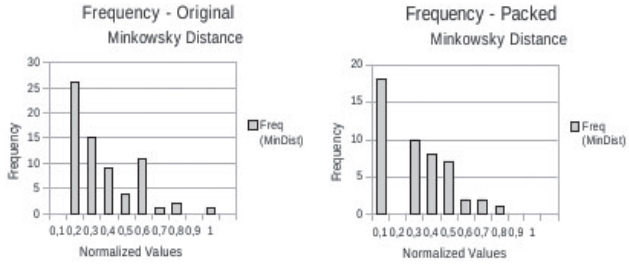

Figure 4. Comparison of the Frequency of the Normalized Values for Minkowsky´s Distance Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

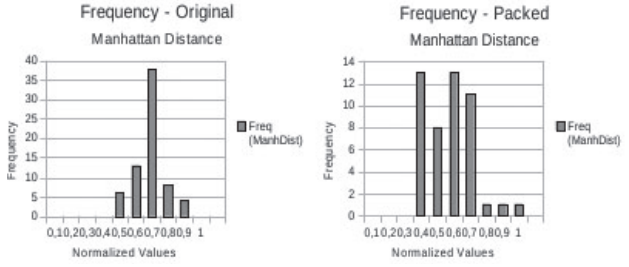

Figure 5. Comparison of the Frequency of the Normalized Values for Manhattan´s Distance Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

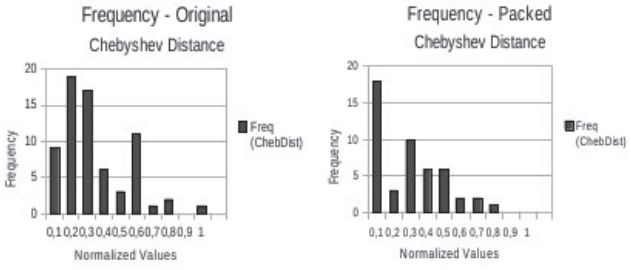

Figure 6. Comparison of the Frequency of the Normalized Values for Chebyshev´s Distance Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

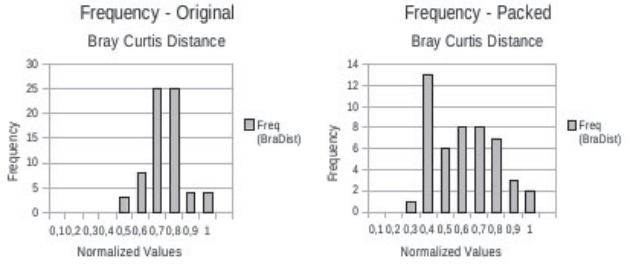

Figure 7. Comparison of the Frequency of the Normalized Values for Bray - Curtis´s Distance Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

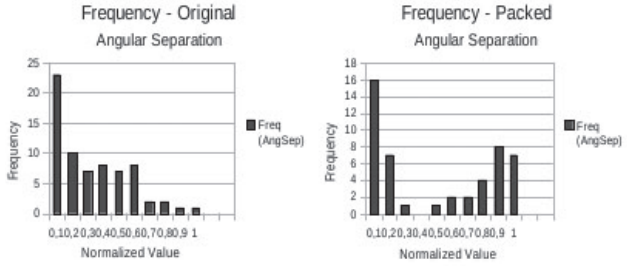

Figure 8. Comparison of the Frequency of the Normalized Values for Angular Separation Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

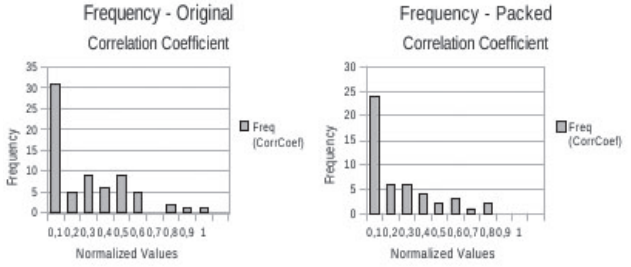

Figure 9. Comparison of the Frequency of the Normalized Values for Correlation Coefficient Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

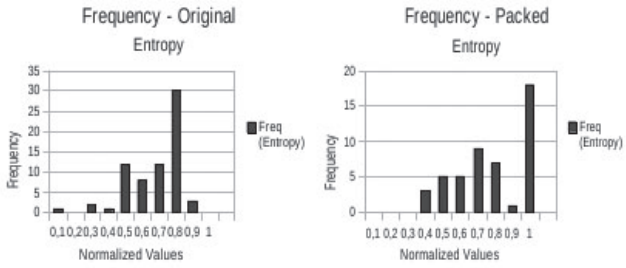

Figure 10. Comparison of the Frequency of the Normalized Values for Entropy Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

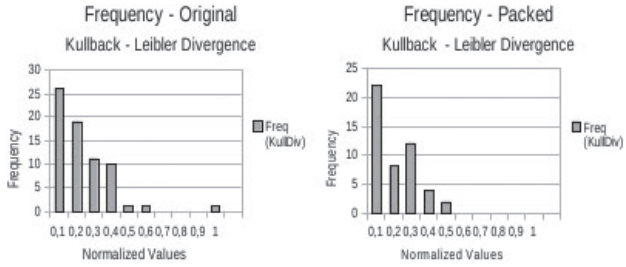

Figure 11. Comparison of the Frequency of the Normalized Values for Kullback - Leibler´s Divergence Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

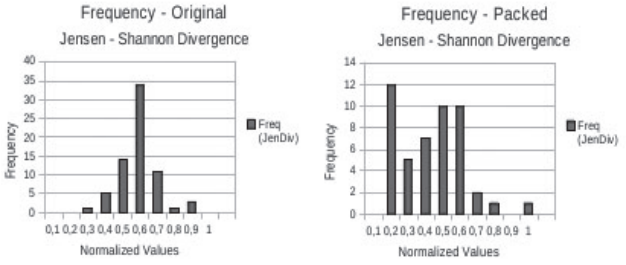

Figure 12. Comparison of the Frequency of the Normalized Values for Jensen – Shannon´s Divergence Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

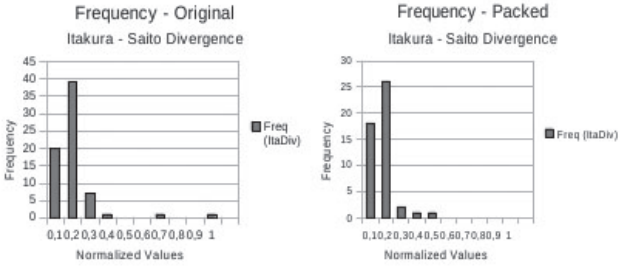

Figure 13. Comparison of the Frequency of the Normalized Values for Itakura – Saito´s Divergence Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

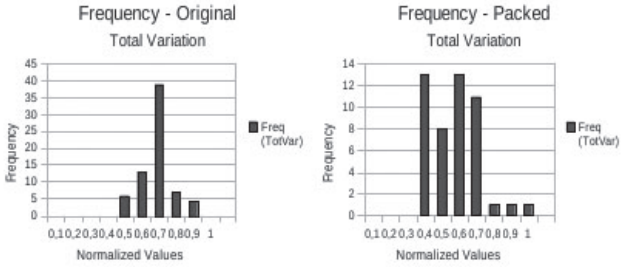

Figure 14. Comparison of the Frequency of the Normalized Values for Total Variation Based on Unigram Distribution for the Original Executable (Left) and Packed Executable (Right).

The degree of change observed in the calculations between the original and packed executables justifies their adoption as inputs for the decision tree building process.

V. Results and Discussion

A. Training Phase

For the training phase, we selected four hundred and fifty six (456) unpacked executables:

Table 1:
Data on the Training Base Selected Executables

	Minimum Size (bytes)	Medium Size (bytes)	Maximum Size (bytes)
Executables	817	124981.96	3558912

Table 2:
Data on Adopted Packers

Packers	MD5
mew11	cbfbb5517bf4d279cf82d9c4cb4fe259
upx	745528339c38f3eb1790182db8febee1
cexe	fa0e3f80b8e188d90e800cb6a92de28e
fsg	00bd8f44c6176394caf6c018c23ea71b
pecompact	21180116c1bc30cda03befa7be798611
mpress	18cabd06078dc2d7b728dbf888fe9561
xcomp97	e28f888ec49ff180f24c477ca3446315

After this step, the original and packed executables are received by the statistical module, which generates the information needed for the training of the decision tree. The information is stored in a text file following the pattern of data entry adopted by the C5.0 program, as given by [13]:

0.005388,95.159503,0.029534,0.646484,0.019531,0.326108,0.746169,0.344504,7.592288,0.130506,0.041571,101.373204,0.323242,yes,BLOCK29,3afb6adccb65b1c4284833080e878db3

Where each line presents – comma separated - the thirteen statistical and information theory calculations over a given block, the block status ("yes" when packed, "no" otherwise), the block id within the executable and the MD5 value of the original executable.

For the preliminary test, each of the training sets for the n-grams adopted are provided as input to C5.0, and two training options are selected. The first is the default option, and the second determines the construction of the decision tree using boost consisting of 10 steps, where the error cases of previous steps are reviewed and used as new inputs to those that follows, generating subsequent changes in the current decision tree in an attempt to improve the final decision tree´s efficiency.

Table 3:
Data for Decision Tree Generated With Default Options.

N-gram	Size of the Decision Tree	Error Rate
Unigram	680	10.40%
Bigram	618	10.90%
Trigram	554	11.20%
Four-gram	575	11.40%

Table 4:
Data for Decision Tree Generated With 10 Step Boost Option.

N-gram	Error Rate
Unigram	9.30%
Bigram	9.70%
Trigram	10.20%
Four-gram	10.50%

Based on these data, the statistics generated over the distribution of unigrams was adopted for the new training attempts with boost option with more steps:

Table 5:
Data for Decision Trees Generated Over Unigram Distribution and Various Steps Boost Option.

Number of Boost Steps	Error Rate
10	9.30%
20	8.80%
30	8.60%
40	8.60%

Therefore, the decision tree built over the distribution of unigrams and boost of 30 steps was adopted as input for the classification module.

Part of the adopted decision tree can be viewed in the following figure:

```
entropy > 6.623186:
|--- corrcoef > 0.684858:
|    |--- manhdis <= 0.519531: no (128)
|    |    manhdis > 0.519531:
|    |    |--- entropy > 7.102338: yes (13)
|    |         entropy <= 7.102338:
|    |         |--- itadiv <= 67.9743: no (14)
|    |              itadiv > 67.9743:
|    |              |--- jendiv <= 0.036151: no (4)
|    |                   jendiv > 0.036151: yes (64/28)
|    corrcoef <= 0.684858:
|    |--- candis > 90.49059: yes (33631/425)
|         candis <= 90.49059:
|         |--- angsep <= 0.763649:
|              |--- mindis <= 0.111632:
|              |    |--- itadiv > 68.00925: yes (23209/2354)
|              |    |    itadiv <= 68.00925:
|              |    |    |--- entropy <= 7.195227: no (19)
|              |    |         entropy > 7.195227: yes (89/13)
|              |    mindis > 0.111632:
|              |    |--- corrcoef <= 0.113134: yes (92/4)
|              |         corrcoef > 0.113134:
|              |         |--- chedis <= 0.10397: no (38)
|              |              chedis > 0.10397: yes (22/4)
|              angsep > 0.763649:
|              |--- candis <= 60.67399:
|                   |--- itadiv <= 55.78081: no (14)
|                   |    itadiv > 55.78081: yes (12/1)
|                   candis > 60.67399:
|                   |--- corrcoef <= 0.488466: yes (77242/1263)
|                        corrcoef > 0.488466:
|                        |--- candis <= 74.81892: no (13)
|                             candis > 74.81892:
|                             |--- jendiv > 0.030838: yes (41/2)
|                                  jendiv <= 0.030838:
|                                  |--- candis <= 82.95027: yes (3)
|                                       candis > 82.95027:
|                                       |--- chedis <= 0.008789: yes (2)
|                                            chedis > 0.008789: no (7)
entropy <= 6.623186:
|--- chedis <= 0.167969:
     |--- entropy > 6.161778:
          |--- manhdis <= 1.005081:
               |--- jendiv <= 0.035845: no (49)
               |    jendiv > 0.035845:
               |    |--- bradis > 0.44367: yes (450/80)
               |         bradis <= 0.44367:
               |         |--- itadiv <= 60.18798:
               |              |--- entropy <= 6.532473: no (28/1)
               |              |    entropy > 6.532473: yes (3)
               |              itadiv > 60.18798:
               |              |--- angsep <= 0.162795: yes (27/1)
               |                   angsep > 0.162795:
               |                   |--- candis <= 97.74641: no (29/3)
               |                        candis > 97.74641: yes (380/149)
```

Figure 15. Segment of the Adopted Decision Tree

B. Production Phase

In order to test this phase, we used a set of 22 unpacked executables that do not belong to the training base of the decision tree.

In addition to the seven previously used packers, packer Themida (identified by md5 6e8ef3480f36ce538d386658b9ba011a), NakedPack (identified by md5 2012b87a57e-1b9e4c05126a1fdc6ed99) and Morphine (identified by md5 fc0c8387125ab4eaad-a551b71d274f8b) were used in the construction of sixty-six (66) packed executables in order to test the robustness of the proposed method in detecting packers which were not part of the original training set.

Table 6:
Test Results for Unpacked Executables.

Executable MD5	Blocks Recognized As Packed (false positives)	Blocks Recognized As Unpacked
09c7859269563c240ab2aaab574483dd	14.085%	85.915%
1a9b51a0d07be16bc44a4f8ff6f538fd	26.667%	73.333%
1f06d05ef9814e4cb5202c197710d2f5	27.778%	72.222%
1f171553f1138dc0062a71a7d275055a	7.285%	92.715%
2cffa74f01e50f2fc07d45dbe56561bb	11.111%	88.889%
378da78d3d3c981b38fb4d10b049d493	37.037%	62.963%
41fb70824080b8f9774f688532a89e01	12.903%	87.097%
5723ccbd541e553b6ca337a296da979f	6.515%	93.485%
6d12a84c55f20a45c78eb1b5c720619b	31.034%	68.966%
8cace33911b71d63fca920cabda3a63a	40.000%	60.000%
8e93cdf0ea8edba63f07e2898a9b2147	15.556%	84.444%
97297c74d02e522b6a69d24d4539a359	16.667%	83.333%
9872199bec05c48b903ca87197dc1908	23.077%	76.923%
9a6a653adf28d9d69670b48f535e6b90	41.463%	58.537%
9d1f6b512a3ca51993d60f6858df000d	10.256%	89.744%
b2099fbd58a8f43282d2f7e14d81f97e	11.765%	88.235%
b65a1a4b606ec35603e98d7ca10d09d7	29.167%	70.833%
c07f1963e4ff877160ca12bcf0d40c2d	29.167%	70.833%
de7cf7de23de43272e708062d0a049b8	16.667%	83.333%
e8b0a9ecb76aaa0c3519e16f34a49858	21.466%	78.534%
ecef404f62863755951e09c802c94ad5	44.737%	55.263%
fbd6b3bb2a40478df5434a073d571cae	21.739%	78.261%

Table 7:
Test Results for Executables Packed with Mew11.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	92.500%	7.500%
1a9b51a0d07be16bc44a4f8ff6f538fd	62.500%	37.500%
1f06d05ef9814e4cb5202c197710d2f5	70.000%	30.000%
1f171553f1138dc0062a71a7d275055a	94.737%	5.263%
2cffa74f01e50f2fc07d45dbe56561bb	82.353%	17.647%
378da78d3d3c981b38fb4d10b049d493	78.571%	21.429%
41fb70824080b8f9774f688532a89e01	78.571%	21.429%
5723ccbd541e553b6ca337a296da979f	95.588%	4.412%
6d12a84c55f20a45c78eb1b5c720619b	78.571%	21.429%
8cace33911b71d63fca920cabda3a63a	75.000%	25.000%
8e93cdf0ea8edba63f07e2898a9b2147	85.714%	14.286%
97297c74d02e522b6a69d24d4539a359	85.714%	14.286%
9872199bec05c48b903ca87197dc1908	76.923%	23.077%
9a6a653adf28d9d69670b48f535e6b90	88.889%	11.111%
9d1f6b512a3ca51993d60f6858df000d	83.333%	16.667%
b2099fbd58a8f43282d2f7e14d81f97e	81.250%	18.750%
b65a1a4b606ec35603e98d7ca10d09d7	75.000%	25.000%
c07f1963e4ff877160ca12bcf0d40c2d	76.923%	23.077%
de7cf7de23de43272e708062d0a049b8	86.957%	13.043%
e8b0a9ecb76aaa0c3519e16f34a49858	96.471%	3.529%
ecef404f62863755951e09c802c94ad5	79.167%	20.833%
fbd6b3bb2a40478df5434a073d571cae	75.000%	25.000%

Table 8:
Test Results for Executables Packed with UPX.

Executable MD5	Blocks Recog-nized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	93.478%	6.522%
1a9b51a0d07be16bc44a4f8ff6f538fd	70.000%	30.000%
1f06d05ef9814e4cb5202c197710d2f5	76.471%	23.529%
1f171553f1138dc0062a71a7d275055a	93.333%	6.667%
2cffa74f01e50f2fc07d45dbe56561bb	80.000%	20.000%
378da78d3d3c981b38fb4d10b049d493	77.778%	22.222%
41fb70824080b8f9774f688532a89e01	81.250%	18.750%
5723ccbd541e553b6ca337a296da979f	95.062%	4.938%
6d12a84c55f20a45c78eb1b5c720619b	76.471%	23.529%
8cace33911b71d63fca920cabda3a63a	73.684%	26.316%
8e93cdf0ea8edba63f07e2898a9b2147	84.000%	16.000%
97297c74d02e522b6a69d24d4539a359	83.333%	16.667%
9872199bec05c48b903ca87197dc1908	69.231%	30.769%
9a6a653adf28d9d69670b48f535e6b90	90.323%	9.677%

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
9d1f6b512a3ca51993d60f6858df000d	86.364%	13.636%
b2099fbd58a8f43282d2f7e14d81f97e	78.947%	21.053%
b65a1a4b606ec35603e98d7ca10d09d7	81.250%	18.750%
c07f1963e4ff877160ca12bcf0d40c2d	75.000%	25.000%
de7cf7de23de43272e708062d0a049b8	84.615%	15.385%
e8b0a9ecb76aaa0c3519e16f34a49858	93.478%	6.522%
ecef404f62863755951e09c802c94ad5	82.759%	17.241%
fbd6b3bb2a40478df5434a073d571cae	73.333%	26.667%

Table 9:
Test Results for Executables Packed with CEXE.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	95.161%	4.839%
1a9b51a0d07be16bc44a4f8ff6f538fd	71.429%	28.571%
1f06d05ef9814e4cb5202c197710d2f5	27.778%	72.222%
1f171553f1138dc0062a71a7d275055a	96.341%	3.659%
2cffa74f01e50f2fc07d45dbe56561bb	88.462%	11.538%
378da78d3d3c981b38fb4d10b049d493	86.364%	13.636%
41fb70824080b8f9774f688532a89e01	86.364%	13.636%
5723ccbd541e553b6ca337a296da979f	97.170%	2.830%
6d12a84c55f20a45c78eb1b5c720619b	81.818%	18.182%
8cace33911b71d63fca920cabda3a63a	81.818%	18.182%
8e93cdf0ea8edba63f07e2898a9b2147	91.176%	8.824%
97297c74d02e522b6a69d24d4539a359	88.235%	11.765%
9872199bec05c48b903ca87197dc1908	86.364%	13.636%
9a6a653adf28d9d69670b48f535e6b90	89.474%	10.526%
9d1f6b512a3ca51993d60f6858df000d	86.667%	13.333%
b2099fbd58a8f43282d2f7e14d81f97e	84.615%	15.385%
b65a1a4b606ec35603e98d7ca10d09d7	81.818%	18.182%
c07f1963e4ff877160ca12bcf0d40c2d	86.364%	13.636%
de7cf7de23de43272e708062d0a049b8	89.474%	10.526%
e8b0a9ecb76aaa0c3519e16f34a49858	97.273%	2.727%
ecef404f62863755951e09c802c94ad5	86.667%	13.333%
fbd6b3bb2a40478df5434a073d571cae	86.364%	13.636%

Table 10:
Test Results for Executables Packed with FSG.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	91.111%	8.889%
1a9b51a0d07be16bc44a4f8ff6f538fd	66.667%	33.333%
1f06d05ef9814e4cb5202c197710d2f5	72.727%	27.273%
1f171553f1138dc0062a71a7d275055a	95.313%	4.688%
2cffa74f01e50f2fc07d45dbe56561bb	84.211%	15.789%
378da78d3d3c981b38fb4d10b049d493	81.250%	18.750%
41fb70824080b8f9774f688532a89e01	80.000%	20.000%
5723ccbd541e553b6ca337a296da979f	96.250%	3.750%
6d12a84c55f20a45c78eb1b5c720619b	81.250%	18.750%
8cace33911b71d63fca920cabda3a63a	72.222%	27.778%
8e93cdf0ea8edba63f07e2898a9b2147	87.500%	12.500%
97297c74d02e522b6a69d24d4539a359	86.957%	13.043%
9872199bec05c48b903ca87197dc1908	80.000%	20.000%
9a6a653adf28d9d69670b48f535e6b90	90.000%	10.000%
9d1f6b512a3ca51993d60f6858df000d	85.000%	15.000%
b2099fbd58a8f43282d2f7e14d81f97e	83.333%	16.667%
b65a1a4b606ec35603e98d7ca10d09d7	78.571%	21.429%
c07f1963e4ff877160ca12bcf0d40c2d	80.000%	20.000%
de7cf7de23de43272e708062d0a049b8	88.000%	12.000%
e8b0a9ecb76aaa0c3519e16f34a49858	94.624%	5.376%
ecef404f62863755951e09c802c94ad5	85.185%	14.815%
fbd6b3bb2a40478df5434a073d571cae	71.429%	28.571%

Table 11:
Test Results for Executables Packed with PECompact.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	91.304%	8.696%
1a9b51a0d07be16bc44a4f8ff6f538fd	75.000%	25.000%
1f06d05ef9814e4cb5202c197710d2f5	73.333%	26.667%
1f171553f1138dc0062a71a7d275055a	93.220%	6.780%
2cffa74f01e50f2fc07d45dbe56561bb	86.364%	13.636%
378da78d3d3c981b38fb4d10b049d493	80.000%	20.000%
41fb70824080b8f9774f688532a89e01	78.947%	21.053%
5723ccbd541e553b6ca337a296da979f	96.000%	4.000%
6d12a84c55f20a45c78eb1b5c720619b	75.000%	25.000%
8cace33911b71d63fca920cabda3a63a	63.636%	36.364%
8e93cdf0ea8edba63f07e2898a9b2147	85.714%	14.286%
97297c74d02e522b6a69d24d4539a359	81.481%	18.519%
9872199bec05c48b903ca87197dc1908	78.947%	21.053%
9a6a653adf28d9d69670b48f535e6b90	88.235%	11.765%

9d1f6b512a3ca51993d60f6858df000d	87.500%	12.500%
b2099fbd58a8f43282d2f7e14d81f97e	81.818%	18.182%
b65a1a4b606ec35603e98d7ca10d09d7	72.222%	27.778%
c07f1963e4ff877160ca12bcf0d40c2d	77.778%	22.222%
de7cf7de23de43272e708062d0a049b8	89.655%	10.345%
e8b0a9ecb76aaa0c3519e16f34a49858	93.407%	6.593%
ecef404f62863755951e09c802c94ad5	90.323%	9.677%
fbd6b3bb2a40478df5434a073d571cae	77.778%	22.222%

Table 12:
Test Results for Executables Packed with MPress.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	90.698%	9.302%
1a9b51a0d07be16bc44a4f8ff6f538fd	72.727%	27.273%
1f06d05ef9814e4cb5202c197710d2f5	76.923%	23.077%
1f171553f1138dc0062a71a7d275055a	94.643%	5.357%
2cffa74f01e50f2fc07d45dbe56561bb	80.952%	19.048%
378da78d3d3c981b38fb4d10b049d493	78.947%	21.053%
41fb70824080b8f9774f688532a89e01	83.333%	16.667%
5723ccbd541e553b6ca337a296da979f	95.714%	4.286%
6d12a84c55f20a45c78eb1b5c720619b	83.333%	16.667%
8cace33911b71d63fca920cabda3a63a	75.000%	25.000%
8e93cdf0ea8edba63f07e2898a9b2147	84.000%	16.000%
97297c74d02e522b6a69d24d4539a359	88.000%	12.000%
9872199bec05c48b903ca87197dc1908	83.333%	16.667%
9a6a653adf28d9d69670b48f535e6b90	90.323%	9.677%
9d1f6b512a3ca51993d60f6858df000d	86.364%	13.636%
b2099fbd58a8f43282d2f7e14d81f97e	76.190%	23.810%
b65a1a4b606ec35603e98d7ca10d09d7	81.250%	18.750%
c07f1963e4ff877160ca12bcf0d40c2d	82.353%	17.647%
de7cf7de23de43272e708062d0a049b8	85.185%	14.815%
e8b0a9ecb76aaa0c3519e16f34a49858	95.349%	4.651%
ecef404f62863755951e09c802c94ad5	86.667%	13.333%
fbd6b3bb2a40478df5434a073d571cae	75.000%	25.000%

Table 13:
Test Results for Executables Packed with Xcomp97.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	91.111%	8.889%
1a9b51a0d07be16bc44a4f8ff6f538fd	70.000%	30.000%
1f06d05ef9814e4cb5202c197710d2f5	66.667%	33.333%
1f171553f1138dc0062a71a7d275055a	93.103%	6.897%
2cffa74f01e50f2fc07d45dbe56561bb	80.000%	20.000%
378da78d3d3c981b38fb4d10b049d493	82.353%	17.647%
41fb70824080b8f9774f688532a89e01	81.250%	18.750%
5723ccbd541e553b6ca337a296da979f	93.671%	6.329%
6d12a84c55f20a45c78eb1b5c720619b	82.353%	17.647%
8cace33911b71d63fca920cabda3a63a	63.158%	36.842%
8e93cdf0ea8edba63f07e2898a9b2147	83.333%	16.667%
97297c74d02e522b6a69d24d4539a359	82.609%	17.391%
9872199bec05c48b903ca87197dc1908	81.250%	18.750%
9a6a653adf28d9d69670b48f535e6b90	87.097%	12.903%
9d1f6b512a3ca51993d60f6858df000d	85.000%	15.000%
b2099fbd58a8f43282d2f7e14d81f97e	78.947%	21.053%
b65a1a4b606ec35603e98d7ca10d09d7	80.000%	20.000%
c07f1963e4ff877160ca12bcf0d40c2d	75.000%	25.000%
de7cf7de23de43272e708062d0a049b8	84.615%	15.385%
e8b0a9ecb76aaa0c3519e16f34a49858	95.506%	4.494%
ecef404f62863755951e09c802c94ad5	82.759%	17.241%
fbd6b3bb2a40478df5434a073d571cae	73.333%	26.667%

Table 14:
Test Results for Executables Packed with NakedPack.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	87.719%	12.281%
1a9b51a0d07be16bc44a4f8ff6f538fd	53.333%	46.667%
1f06d05ef9814e4cb5202c197710d2f5	64.706%	35.294%
1f171553f1138dc0062a71a7d275055a	92.000%	8.000%
2cffa74f01e50f2fc07d45dbe56561bb	76.923%	23.077%
378da78d3d3c981b38fb4d10b049d493	72.727%	27.273%
41fb70824080b8f9774f688532a89e01	72.727%	27.273%
5723ccbd541e553b6ca337a296da979f	93.069%	6.931%
6d12a84c55f20a45c78eb1b5c720619b	72.727%	27.273%
8cace33911b71d63fca920cabda3a63a	65.385%	34.615%
8e93cdf0ea8edba63f07e2898a9b2147	80.645%	19.355%
97297c74d02e522b6a69d24d4539a359	80.645%	19.355%
9872199bec05c48b903ca87197dc1908	71.429%	28.571%
9a6a653adf28d9d69670b48f535e6b90	83.784%	16.216%
9d1f6b512a3ca51993d60f6858df000d	77.778%	22.222%
b2099fbd58a8f43282d2f7e14d81f97e	76.000%	24.000%
b65a1a4b606ec35603e98d7ca10d09d7	70.000%	30.000%
c07f1963e4ff877160ca12bcf0d40c2d	61.905%	38.095%
de7cf7de23de43272e708062d0a049b8	78.788%	21.212%
e8b0a9ecb76aaa0c3519e16f34a49858	93.162%	6.838%

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
ecef404f62863755951e09c802c94ad5	82.500%	17.500%
fbd6b3bb2a40478df5434a073d571cae	70.000%	30.000%

Table 15:
Test Results for Executables Packed with Morphine.

Executable MD5	Blocks Recognized As Packed	Blocks Recognized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	97.959%	2.041%
1a9b51a0d07be16bc44a4f8ff6f538fd	78.947%	21.053%
1f06d05ef9814e4cb5202c197710d2f5	86.364%	13.636%
1f171553f1138dc0062a71a7d275055a	97.436%	2.564%
2cffa74f01e50f2fc07d45dbe56561bb	92.500%	7.500%
378da78d3d3c981b38fb4d10b049d493	90.625%	9.375%
41fb70824080b8f9774f688532a89e01	88.571%	11.429%
5723ccbd541e553b6ca337a296da979f	99.035%	0.965%
6d12a84c55f20a45c78eb1b5c720619b	90.909%	9.091%
8cace33911b71d63fca920cabda3a63a	90.000%	10.000%
8e93cdf0ea8edba63f07e2898a9b2147	94.000%	6.000%
97297c74d02e522b6a69d24d4539a359	93.617%	6.383%
9872199bec05c48b903ca87197dc1908	86.667%	13.333%
9a6a653adf28d9d69670b48f535e6b90	91.111%	8.889%
9d1f6b512a3ca51993d60f6858df000d	90.909%	9.091%
b2099fbd58a8f43282d2f7e14d81f97e	92.105%	7.895%
b65a1a4b606ec35603e98d7ca10d09d7	89.286%	10.714%
c07f1963e4ff877160ca12bcf0d40c2d	89.286%	10.714%
de7cf7de23de43272e708062d0a049b8	94.231%	5.769%
e8b0a9ecb76aaa0c3519e16f34a49858	98.462%	1.538%
ecef404f62863755951e09c802c94ad5	92.857%	7.143%
fbd6b3bb2a40478df5434a073d571cae	89.286%	10.714%

Table 16:
Test Results for Executables Packed With Themida.

Executable MD5	Blocks Recog-nized As Packed	Blocks Recog-nized As Unpacked (false negatives)
09c7859269563c240ab2aaab574483dd	99.254%	0.746%
1a9b51a0d07be16bc44a4f8ff6f538fd	99.298%	0.702%
1f06d05ef9814e4cb5202c197710d2f5	99.318%	0.682%
1f171553f1138dc0062a71a7d275055a	99.496%	0.504%
2cffa74f01e50f2fc07d45dbe56561bb	99.126%	0.874%
378da78d3d3c981b38fb4d10b049d493	99.212%	0.788%
41fb70824080b8f9774f688532a89e01	99.388%	0.612%
5723ccbd541e553b6ca337a296da979f	99.241%	0.759%
6d12a84c55f20a45c78eb1b5c720619b	99.326%	0.674%
8cace33911b71d63fca920cabda3a63a	99.235%	0.765%
8e93cdf0ea8edba63f07e2898a9b2147	99.178%	0.822%
97297c74d02e522b6a69d24d4539a359	99.241%	0.759%
9872199bec05c48b903ca87197dc1908	99.220%	0.780%
9a6a653adf28d9d69670b48f535e6b90	99.222%	0.778%
9d1f6b512a3ca51993d60f6858df000d	99.382%	0.618%
b2099fbd58a8f43282d2f7e14d81f97e	99.410%	0.590%
b65a1a4b606ec35603e98d7ca10d09d7	99.294%	0.706%
c07f1963e4ff877160ca12bcf0d40c2d	99.337%	0.663%
de7cf7de23de43272e708062d0a049b8	99.385%	0.615%
e8b0a9ecb76aaa0c3519e16f34a49858	98.139%	1.861%
ecef404f62863755951e09c802c94ad5	98.501%	1.499%
fbd6b3bb2a40478df5434a073d571cae	99.128%	0.872%

We present below the histogram of false positives for the original unpacked executables and the histograms of false negatives for the packed executables:

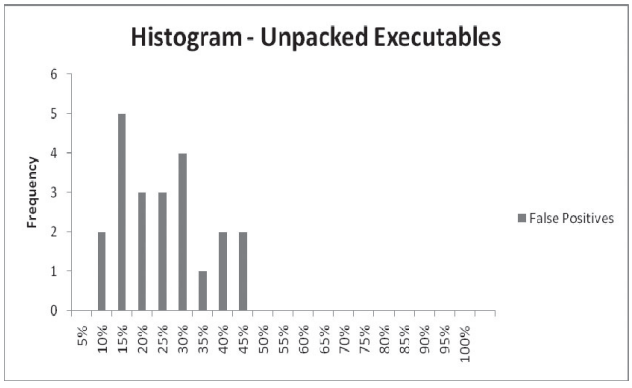

Figure 16. Histogram of False Positives for the Original Unpacked Executables

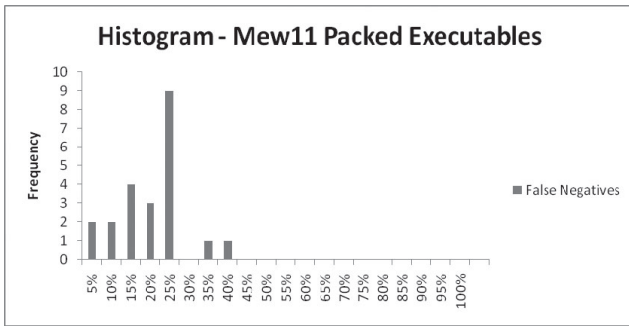

Figure 17. Histogram of False Negatives for the Mew11 Packed Executables

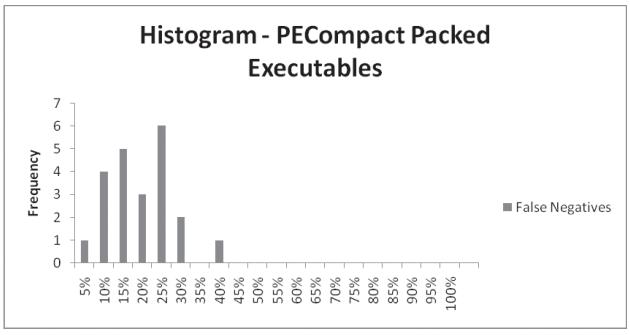

Figure 21. Histogram of False Negatives for the PECompact Packed Executables

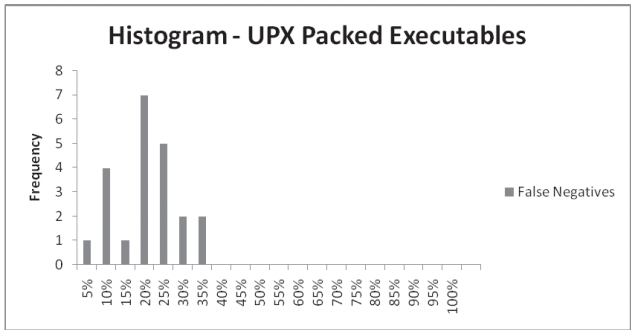

Figure 18. Histogram of False Negatives for the UPX Packed Executables

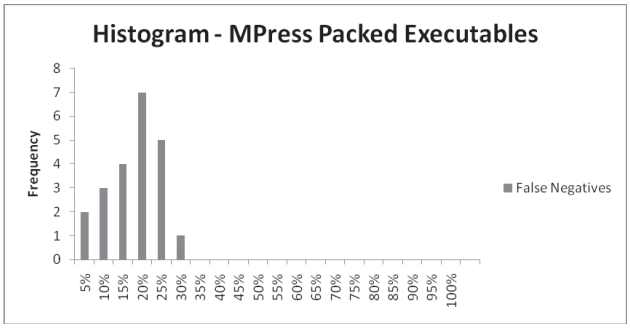

Figure 22. Histogram of False Negatives for the MPress Packed Executables

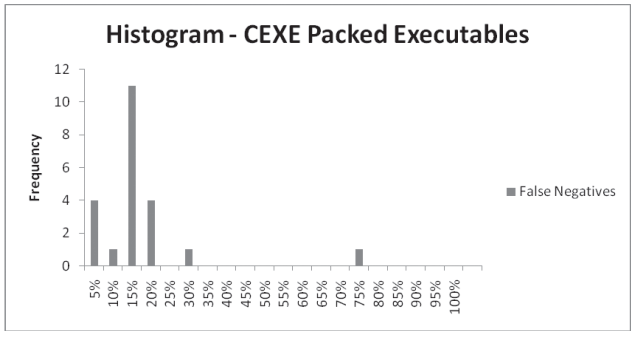

Figure 19. Histogram of False Negatives for the CEXE Packed Executables

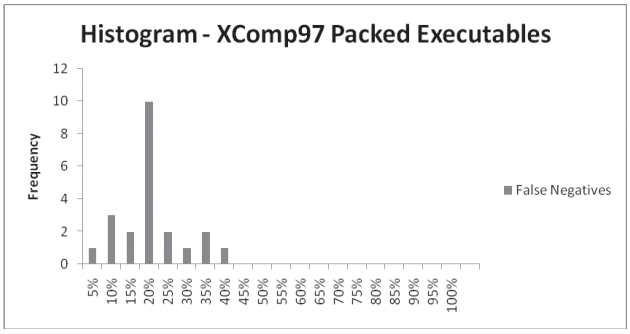

Figure 23. Histogram of False Negatives for the XComp97 Packed Executables

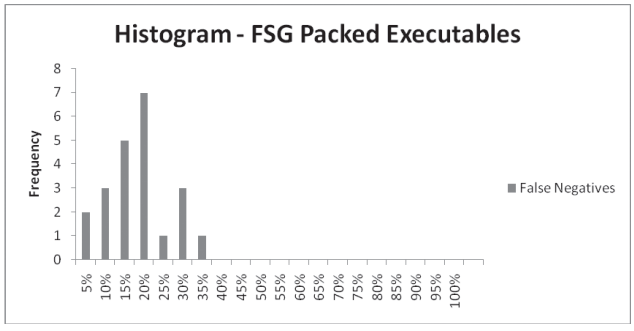

Figure 20. Histogram of False Negatives for the FSG Packed Executables

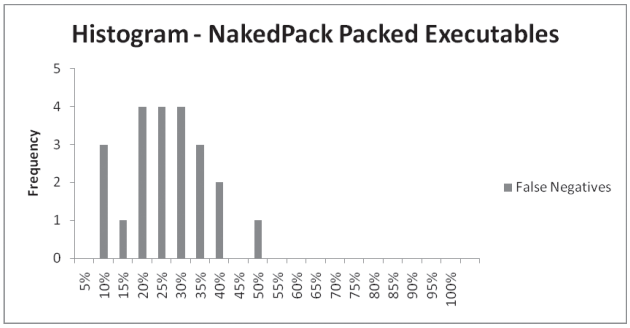

Figure 24. Histogram of False Negatives for the NakedPack Packed Executables

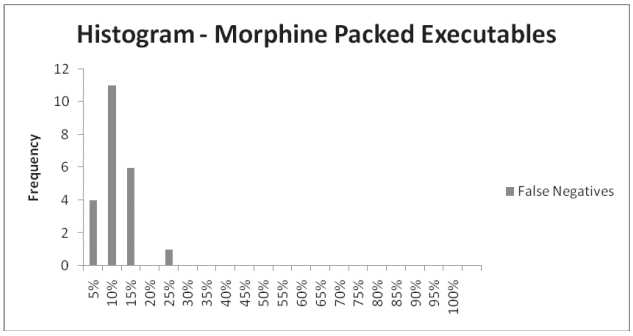

Figure 25. Histogram of False Negatives for the Morphine Packed Executables

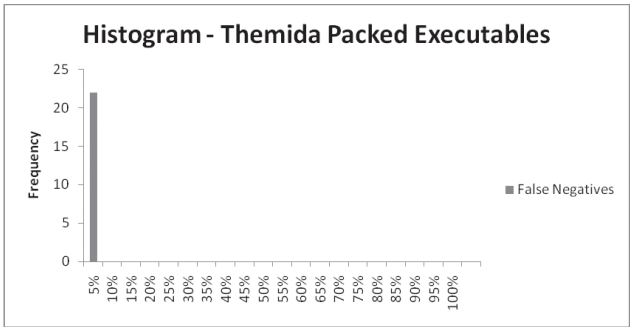

Figure 26. Histogram of False Negatives for the Themida Packed Executables

In the analysis of the original binaries, we see three cases of false positives in about 40 to 45% of the blocks analyzed. Still, considering the classification criteria adopted for packing - 50% of blocks recognized as such - no binaries suffer misclassification.

In the analysis of binaries packed with Mew11, FSG, PECompact, Xcomp97 and Mpress, the false negative rate is lower, reaching a maximum around 37%. Again, no binaries suffered misclassification.

For binaries packed with CEXE, there is one case of misclassification out of the twenty-two analyzed, where the false negative rate reached 72.222%.

For the data generated with the usage of packers that where not in the set of packers used for the training of the BinStat application, we can see that no binary packed with NakedPack and Morphine suffered misclassification.

Finally, one can see that the binaries generated with Themida packer were easily identified as packed, with positive rates in the range of ninety-nine (99) percent.

VI. Conclusion

The paper presented a methodology for determining the packing status of executables by analyzing their binary content.

The data presented demonstrates that among the four n-grams adopted, the one that presented the best result for the construction of the decision tree was the unigram, paired with the option of algorithm boosting with 30 steps.

With these options and considering the classification criteria adopted for packaging - 50% of blocks recognized as such – only one binary packed with CEXE suffered misclassification.

Additionally, all executables packed with NakedPack, Morphine and Themida where correctly classified even though those packers were not part of the training base of the BinStat application.

In the case of Themida, all binaries had more than ninety-nine (99) percent of their blocks recognized as packed, showing that this packer mechanism is easily recognized by BinStat tool.

These results demonstrate the robustness of the methodology presented in this paper.

It is noteworthy that the presented method does not use the signature technique presented on tools such as PEiD. Because of that, it is able to detect executables packed with tools that circumvents such technique.

For future work, we intend to broaden the base of training and testing with more binaries and the use of other packing tools in addition to the seven used.

In addition, we intend to investigate the impact on the methodology described of the introduction of techniques for binary processing considering some peculiarities of the PE (Portable Executable) format, before subjecting them to presented statistical module.

VII. References

[1] RODRIGUES, R. Febraban: fraudes online somaram prejuízo de R$ 900 mi em 2009. Available in: http://computerworld.uol.com.br/seguranca/2010/08/31/febraban-fraudes-online-somaram-prejuizo-de-r-900-mi-em-2009/. Accessed in: 11 jan 2011.

[2] BESTUZHEV, D. Brazil: a country rich in banking Trojans. Available in: http://www.securelist.com/en/analysis?pubid=204792084. Accessed in: 11 jan 2011.

[3] MCAFEE. Mcafee Threat Intelligence. Available in: http://www.mcafee.com/us/mcafee-labs/threat-intelligence.aspx. Accessed in: 11 jan 2011.

[4] MCMILLAN, R. Was Stuxnet built to attack Iran's nuclear program? Available in: http://www.networkworld.com/news/2010/092110-was-stuxnet-built-to-attack.html. Accessed in: 12 jan 2011.

[5] RIEGER, F. Stuxnet: targeting the iranian enrichment centrifuges in Natanz? Available in: http://frank.geekheim.de/?p=1189. Accessed in: 12 jan 2011.

[6] SCHNEIER, B. The Stuxnet Worm. Available in: http://www.schneier.com/blog/archives/2010/09/the_stuxnet_wor.html. Accessed in: 12 jan 2011

[7] GUO, F., FERRIE, P. CHIUEH, T. A Study of the Packer Problem and Its Solutions. 11th International Symposium On Recent Advances In Intrusion Detection (RAID), 2008, Boston, USA. Anals... Available in: https://wiki.smu.edu.sg/w/flyer/images/f/fe/RAID08_Packer.pdf. Accessed in: 22 jul 2011.

[8] JIBZ, QUERTON, SNAKER, XINEOHP., / BOB.PEiD. PEiD. Available in: http://www.peid.info/. Accessed in: 21 jan 2011.

[9] KIM, H. C., INOUE, D. ETO, M. TAKAGI, Y. NAKAO, K. Toward Generic Unpacking Techniques for Malware Analysis with Quantification of Code Revelation. Joint Workshop on Information Security (JWIS), 2009, Kaohsiung, Taiwan. Anals... Available in: http://jwis2009.nsysu.edu.tw/location/paper/Toward%20Generic%20Unpacking%20Techniques%20for%20Malware%20Analysis%20with%20Quantification%20of%20Code%20Revelation.pdf. Accessed in: 18 jan 2011.

[10] KANG, M. G., POOSANKAM, P., YIN, H. Renovo: A Hidden Code Extractor for Packed Executables. 2007. 5th ACM Workshop on Recurring Malcode (WORM) 2007, Virginia, USA. Anals... Available in:http://bitblaze.cs.berkeley.edu/papers/renovo.pdf. Accessed in: 22 jul 2011.

[11] TABISH, S. M. , SHAFIQ, M. Z., FAROOQ, M. Malware Detection using Statistical Analysis of Byte-Level File Content. In: ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Workshop on CyberSecurity and Intelligence Informatics (CSI), 2009, Paris. Anals... Paris: ACM Press, 2009. Available in: http://www.cse.msu.edu/~tabishsy/papers/momina_CSIKDD.pdf. Accessed in: 22 jul 2011.

[12] COVER, T. M., THOMAS, J. A. Elements of Information Theory. Hoboken: Wiley, 2006.

[13] RULEQUEST. Data Mining Tools See5 and C5.0. Available in: http://www.rulequest.com/see5-info.html. Accessed in: 10 mar 2011.