OPEM: A Static-Dynamic Approach for Machine-learning-based Malware Detection

Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo G. Bringas

S^3Lab, DeustoTech - Computing, Deusto Institute of Technology
University of Deusto,
Avenida de las Universidades 24, 48007
Bilbao, Spain
{isantos,jaime.devesa,felix.brezo,pablo.garcia.bringas}@deusto.es

Abstract. Malware is any computer software potentially harmful to both computers and networks. The amount of malware is growing every year and poses a serious global security threat. Signature-based detection is the most extended method in commercial antivirus software, however, it consistently fails to detect new malware. Supervised machine learning has been adopted to solve this issue. There are two types of features that supervised malware detectors use: (i) static features and (ii) dynamic features. Static features are extracted without executing the sample whereas dynamic ones requires an execution. Both approaches have their advantages and disadvantages. In this paper, we propose for the first time, OPEM, an hybrid unknown malware detector which combines the frequency of occurrence of operational codes (statically obtained) with the information of the execution trace of an executable (dynamically obtained). We show that this hybrid approach enhances the performance of both approaches when run separately.

Keywords: malware, hybrid, static, dynamic, machine learning, computer security

1 Introduction

Machine-learning-based malware detectors (e.g., [1–4]) commonly rely on datasets that include several characteristic features for both malicious samples and benign software to build classification tools that detect malware in the wild (i.e., undocumented malware). Two kind of features can be used to face obfuscated and previously unseen malware: statically or dynamically extracted characteristics. Static analysis extract several useful features from the executable in inspection without actually executing it, whereas dynamic analysis executes the inspected specimen in a controlled environment called 'sandbox' [5]. The main advantages of static techniques are that they are safer because they do not execute malware, they are able to analyse all the execution paths of the binary, and the analysis and detection is usually fast [5]. However, they are not resilient to packed malware (executables that have been either compressed or cyphered) [6] or complex obfuscation techniques [7]. On the contrary, dynamic techniques can guarantee that the executed code shows the actual behaviour of the executable and, therefore, they are the preferred choice when a whole understanding of the binary is required [8]. However, they have also several shortcomings: they can only analyse a single execution path, they introduce a significant performance overhead, and malware can identify the controlled environments [9].

Given this background, we present here OPEM, the first machine-learning-based malware detector that employs a set of features composed of both static and dynamic features. The static features are based on a novel representation of executables: opcode sequences [10]. This technique models an executable as sequences of operational codes (i.e., the action to perform in machine code language) of a fixed length and computes their frequencies to generate a vector of frequencies of opcode sequences. On the other hand, the dynamic features are extracted by monitoring system calls, operations, and raised exceptions on an execution within an emulated environment to finally generate a vector of binary characteristics representing whether a specific comportment is present within an executable or not [11]. In summary, our main contributions to the state of the art are the following ones: (i) we present a new hybrid representation of executables composed of both statically and dynamically extracted features, (ii) based upon this representation, we propose a new malware detection method which employs supervised learning to detect previously unseen and undocumented malware and (iii) we perform an empirical study to determine which benefits brings this hybrid approach to the standalone static and dynamic representations.

2 Overview of OPEM

2.1 Statically Extracted Features

To represent executables using opcodes, we extract the *opcode-sequences* and their frequency of appearance. Specifically, we define a program ρ as a set of ordered opcodes o, $\rho = (o_1, o_2, o_3, o_4, ..., o_{\ell-1}, o_\ell)$, where ℓ is the number of instructions I of the program ρ. An opcode sequence os is defined as a subset of opcodes within the executable file where $os \subseteq \rho$; it is made up of opcodes o, $os = (o1, o2, o3, ..., o_{m1}, o_m)$ where m is the length of the sequence of opcodes os. Consider an example code formed by the opcodes `mov`, `add`, `push` and `add`; the following sequences of length 2 can be generated: $s_1 =$ (`mov,add`), $s_2 =$ (`add`, `push`) and $s_3 =$ (`push`, `add`).

Afterwards, we compute the frequency of occurrence of each opcode sequence within the file by using *term frequency* (tf) [12] that is a weight widely used in information retrieval: $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$ where $n_{i,j}$ is the number of times the sequence $s_{i,j}$ (in our case opcode sequence) appears in an executable e, and $\sum_k n_{k,j}$ is the total number of terms in the executable e (in our case the total number of possible opcode sequences)

We define the *Weighted Term Frequency* (WTF) as the result of weighting the relevance of each opcode when calculating the term frequency. To calculate the relevance of each individual opcode, we collected malware from the VxHeavens website[1] to assemble a malware dataset of 13,189 malware executables and we collected 13,000 executables from our computers. Using this dataset, we disassemble each executable and compute the mutual information gain for each opcode and the class: $I(X;Y) = \sum_{y \epsilon Y} \sum_{x \epsilon X} p(x,y) \log \left(\frac{p(x,y)}{p(x) \cdot p(y)} \right)$ where X is the opcode frequency and Y is the class of the file (i.e., malware or benign software), $p(x,y)$ is the joint probability distribution function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y. In our particular case, we defined the two variables as the single opcode and whether or not the instance was malware. Note that this weight only measures the relevance of a single opcode and not the relevance of an opcode sequence.

Using these weights, we computed the WTF as the product of sequence frequencies and the previously calculated weight of every opcode in the sequence: $wtf_{i,j} = tf_{i,j} \cdot \prod_{o_z \epsilon S} \frac{weight(o_z)}{100}$ where $weight(o_z)$ is the calculated weight, by means of mutual information gain, for the opcode o_z and $tf_{i,j}$ is the *sequence frequency measure* for the given opcode sequence. We obtain a vector $\boldsymbol{v}$ composed of weighted opcode-sequence frequencies, $\boldsymbol{v} = ((os_1, wtf_1), ..., (os_n, wtf_n))$, where os_i is the opcode sequence and wtf_i is the weighted term frequency for that particular opcode sequence.

2.2 Dynamically Extracted Features

Behaviour monitoring is a dynamic analysis technique in which the suspicious file is executed inside a contained and secure environment, called sandbox, in order to get a complete and detailed trace of the actions performed in the system. There are two different approaches for dynamic analysis [13]: (i) taking a snapshot of the complete system before running the suspicious program and comparing it with another snapshot of the system after the execution in order to find out differences and (ii) monitoring the behaviour of the executable during execution with specialised tools.

For our research we have chosen a sandbox [11] that monitors the behaviour of the executable during execution. The suspicious Windows Portable Executable (PE) files are executed inside the sandbox environment, and relevant Windows API calls are logged, showing their behaviour. This work is a new approach of sandbox using both emulation (Qemu) and simulation (Wine) techniques, with the aim of achieving the greatest transparency possible without interfering with the system.

We describe now the two main platforms of our sandbox solution:

- Wine is an open-source and complete re-implementation (simulation) of the Win-32 Application Programming Interface (API). It allows Windows PE files to run as-if-natively under Unix-based operating systems. However. there are still some limitations in the implementation, which hinders some programs from working properly.

[1] http://vx.netlux.org/

- Qemu is an open-source pure software virtual machine emulator that works by performing equivalent operations in software for any given CPU instruction. Unfortunately, there are several malicious executables aware of being executed in a contained environment exploiting different bugs within this virtual machine. However, they can be fixed easily [14]. As Peter Ferrie stated [14], only pure software virtual machine emulators can approach complete transparency, and it should be possible, at least in theory, to reach the point where detection of the virtual machine is unreliable.

Every call done by a process (identified by its *PID*) to the Windows API (divided in families, e.g., *registry*, *memory* or *files*) is stored into a log, specifying the state of the parameters before (*IN*) and after (*OUT*) in the body of the functions. Thereby, we can obtain a complete and homogeneous trace with all the behaviour of the processes, without any interference with the system.

For each executable analysed in the sandbox, we obtain a complete *in-raw* trace with its detailed behaviour. To automatically extract the relevant information in a vector format from the traces, we developed several regular expression rules, which define various specific actions performed by the binary, and a parser to identify them. Most of the actions defined are characteristic of malicious behaviour but there are both benign and malicious behaviour rule definitions. We have classified them into seven different groups:

- **Files**: Every action involving manipulation of files, like creation, opening or searching.
- **Protection**: Most of malware avoid execution if they are being debugged or executed in a virtual environment.
- **Persistence**: Once installed in the System, the malware wants to survive reboots, e.g., by adding registry keys or creating toolbars.
- **Network**: Actions regarding to network connectivity, e.g., creation of a RPC pipe or accessing an URL.
- **Processes**: Manipulation of processes and threads, like creation of multiple threads.
- **System Information**: Retrieving information about the System, e.g., getting the web browsing history.
- **Errors**: Errors raised by Wine, like error loading a DLL, or an unhandled page fault.

The behaviour of an executable is a vector made up of the aforementioned features. We represent an executable as a vector $\boldsymbol{v}$ composed by binary characteristics c, where c can be either 1 (true) or 0 (false), $\boldsymbol{v} = (c_1, c_2, c_3, ..., c_{n-1}, c_n)$ and n is the number of total monitored actions.

In this way, we have characterised the vector information as binary digits, called features, each one representing the corresponding characteristic of the behaviour. When parsing a report, if one of the defined actions is detected by a rule, the corresponding feature is activated. The resulting vector for each program's trace is a finite sequence of bits, a proper information for classifiers to effectively recognize patterns and correlate similarities across a huge amount of instances [15]. Likewise, both *in-raw* trace log and feature sequence for each analysed executable are stored in a database for further treatment.

3 Experimental Validation

To validate our proposed method, we used two different datasets to test the system: a malware dataset and a benign software dataset. We downloaded several malware samples from the VxHeavens website to assemble a malware dataset of 1,000 malicious programs. For the benign dataset, we gathered 1,000 legitimate executables from our computers.

We extracted the opcode-sequence representation for every file in that dataset for a opcode-sequence length $n = 2$. The number of features obtained with an opcode-length of two was very high: 144,598 features. To deal with this, we applied a feature selection step using Information Gain [16] and we selected the top 1,000 features. We extracted the dynamic characteristics for the malware and benign by monitoring it in the emulated environment. The number of features was 63. We combined this two different datasets into one, creating thus a hybrid static-dynamic dataset. To compare our method, we have also kept the datasets with only the static features and only the dynamic features. To validate our approach, we performed the following the steps:

- **Cross validation:** To evaluate the performance of machine-learning classifiers, k-fold cross validation is usually used in machine-learning experiments [17].
  Thereby, for each classifier we tested, we performed a k-fold cross validation [18] with $k = 10$. In this way, our dataset was split 10 times into 10 different sets of learning (90% of the total dataset) and testing (10% of the total data).
- **Learning the model:** For each validation step, we conducted the learning phase of the algorithms with the training datasets, applying different parameters or learning algorithms depending on the concrete classifier. Specifically, we used the following four models:
  - *Decision Trees:* We used Random Forest [19] and J48 (Weka's *C4.5* [20] implementation).
  - *K-Nearest Neighbour:* We performed experiments over the range $k = 1$ to $k = 10$ to train KNN.
  - *Bayesian networks:* We used several structural learning algorithms; K2 [21], Hill Climber [22] and Tree Augmented Naïve (TAN) [23]. We also performed experiments with a Naïve Bayes classifier [24].
  - *Support Vector Machines:* We used a Sequential Minimal Optimization (SMO) algorithm [25], and performed experiments with a polynomial kernel [26], a normalised polynomial kernel [26], Pearson VII function-based universal kernel [27], and a Radial Basis Runction (RBF) based kernel [26].

Table 1. Accuracy results (%).

Classifier	Static	Dynamic	Hybrid
	Approach	Approach	Approach
KNN K=1	94.83	77.19	96.22
KNN K=2	93.15	76.72	95.36
KNN K=3	94.16	76.68	94.63
KNN K=4	93.89	76.58	94.46
KNN K=5	93.50	76.35	93.68
KNN K=6	93.38	76.34	93.52
KNN K=7	92.87	76.33	93.51
KNN K=8	92.89	76.31	93.30
KNN K=9	92.10	76.29	92.94
KNN K=10	92.24	76.24	92.68
DT: J48	92.61	76.72	93.59
DT: Random Forest N=10	95.26	77.12	95.19
SVM: RBF Kernel	91.93	76.75	93.25
SVM: Polynomial Kernel	95.50	76.87	95.99
SVM: Normalised Polynomial Kernel	95.90	77.26	96.60
SVM: Pearson VII Kernel	94.35	77.23	95.56
Naïve Bayes	90.02	74.36	90.11
Bayesian Network: K2	86.73	75.73	87.20
Bayesian Network: Hill Climber	86.73	75.73	87.22
Bayesian Network: TAN	93.40	75.47	93.53

- **Testing the model:** To evaluate each classifier's capability, we measured the True Positive Ratio (TPR), i.e., the number of malware instances correctly detected, divided by the total number of malware files:

$$TPR = \frac{TP}{TP + FN} \quad (1)$$

Table 2. TPR results.

Classifier	Static	Dynamic	Hybrid
	Approach	Approach	Approach
KNN K=1	0.95	0.88	0.95
KNN K=2	0.96	0.88	0.97
KNN K=3	0.94	0.88	0.94
KNN K=4	0.95	0.89	0.96
KNN K=5	0.92	0.89	0.90
KNN K=6	0.93	0.89	0.94
KNN K=7	0.90	0.89	0.92
KNN K=8	0.91	0.89	0.93
KNN K=9	0.88	0.89	0.91
KNN K=10	0.90	0.89	0.91
DT: J48	0.93	0.95	0.94
DT: Random Forest	0.96	0.85	0.96
SVM: RBF Kernel	0.89	0.95	0.90
SVM: Polynomial Kernel	0.96	0.93	0.97
SVM: Normalised Polynomial Kernel	0.94	0.94	0.96
SVM: Pearson VII Kernel	0.95	0.89	0.93
Naïve Bayes	0.90	0.57	0.90
Bayesian Network: K2	0.83	0.63	0.83
Bayesian Network: Hill Climber	0.83	0.63	0.83
Bayesian Network: TAN	0.91	0.85	0.91

Table 3. FPR results.

Classifier	Static	Dynamic	Hybrid
	Approach	Approach	Approach
KNN K=1	0.05	0.34	0.03
KNN K=2	0.10	0.35	0.06
KNN K=3	0.05	0.35	0.05
KNN K=4	0.07	0.36	0.07
KNN K=5	0.05	0.36	0.05
KNN K=6	0.06	0.36	0.07
KNN K=7	0.04	0.36	0.07
KNN K=8	0.05	0.36	0.07
KNN K=9	0.04	0.36	0.07
KNN K=10	0.05	0.36	0.06
DT: J48	0.08	0.34	0.01
DT: Random Forest N=10	0.06	0.31	0.06
SVM: RBF Kernel	0.05	0.42	0.03
SVM: Polynomial Kernel	0.05	0.39	0.05
SVM: Normalised Polynomial Kernel	0.02	0.40	0.03
SVM: Pearson VII Kernel	0.06	0.34	0.01
Naïve Bayes	0.10	0.09	0.10
Bayesian Network: K2	0.09	0.12	0.09
Bayesian Network: Hill Climber	0.09	0.12	0.09
Bayesian Network: TAN	0.04	0.34	0.04

where TP is the number of malware cases correctly classified (true positives) and FN is the number of malware cases misclassified as legitimate software (false negatives).

We also measured the False Positive Ratio (FPR), i.e., the number of benign executables misclassified as malware divided by the total number of benign files:

Table 4. AUC results.

Classifier	Static	Dynamic	Hybrid
	Approach	Approach	Approach
KNN K=1	0.95	0.89	0.96
KNN K=2	0.96	0.88	0.97
KNN K=3	0.97	0.88	0.98
KNN K=4	0.97	0.88	0.98
KNN K=5	0.97	0.88	0.98
KNN K=6	0.98	0.88	0.98
KNN K=7	0.98	0.88	0.98
KNN K=8	0.98	0.88	0.98
KNN K=9	0.98	0.88	0.98
KNN K=10	0.97	0.88	0.98
DT: J48	0.93	0.78	0.93
DT: Random Forest N=10	0.99	0.89	0.99
SVM: RBF Kernel	0.92	0.77	0.93
SVM: Polynomial Kernel	0.95	0.77	0.96
SVM: Normalised Polynomial Kernel	0.96	0.77	0.97
SVM: Pearson VII Kernel	0.94	0.77	0.96
Naïve Bayes	0.93	0.85	0.93
Bayesian Network: K2	0.94	0.86	0.94
Bayesian Network: Hill Climber	0.94	0.86	0.94
Bayesian Network: TAN	0.98	0.87	0.98

$$FPR = \frac{FP}{FP + TN} \qquad (2)$$

where FP is the number of benign software cases incorrectly detected as malware and TN is the number of legitimate executables correctly classified. Furthermore, we measured the accuracy, i.e., the total number of the classifier's hits divided by the number of instances in the whole dataset:

$$Accuracy(\%) = \frac{TP + TN}{TP + FP + TP + TN} \cdot 100 \qquad (3)$$

Besides, we measured the Area Under the ROC Curve (AUC) that establishes the relation between false negatives and false positives [28]. The ROC curve is obtained by plotting the TPR against the FPR.

Tables 1, 2, 3 and 4 show the obtained results in terms of accuracy, TPR, FPR and AUC, respectively. For every classifier, the results were improved when using the combination of both static and dynamic features. In particular, the best overall results were obtained by SVM trained with Polynomial Kernel and Normalised Polynomial Kernel.

The obtained results validate our initial hypothesis that building an unknown malware detector based on opcode-sequence is feasible. The machine-learning classifiers achieved high performance in classifying unknown malware. Nevertheless, there are several considerations regarding the viability of this method.

First, regarding the static approach, it cannot counter packed malware. Packed malware is the result of cyphering the payload of the executable and deciphering it when the executable is finally loaded into memory. A way to solve this obvious limitation of our malware detection method is the use of a generic dynamic unpacking schema such as PolyUnpack [6], Renovo [29], OmniUnpack [30] and Eureka [31].

Second, with regards to the dynamic approach, in order to take advantage over antivirus researchers, malware writers have included diverse evasion techniques [14, 32] based on bugs on the virtual machines implementation to fight back. Nevertheless, with the aim of reducing the impact of these countermeasures, we can improve the Qemu's source code [14] in order to solve the bugs and not to be vulnerable to the above-mentioned techniques. It is also possible that some malicious actions are only triggered under specific circumstances depending on the environment, so relying on a single program execution will not manifest all its behaviour. This is solved with a technique called *multiple execution path* [33], making the system able to obtain different behaviours displayed by the suspicious executable.

4 Concluding remarks

While machine-learning methods are a suitable approach for unknown malware, they use either static or dynamic features to train the algorithms. A combination of both approaches can be useful in order to improve the results of static and dynamic approaches. In this paper, we have presented OPEM which is the first combination of both static and dynamic approaches to detect unknown malware.

The future development of this malware detection system will be concentrated in three main research areas. First, we will focus on facing packed executables using a dynamic unpacker. Second, we plan to extend both the dynamic analysis and the static dynamic in order to improve the results of this hybrid malware detector. Finally, we will study the problem of scalability of malware databases using a combination of feature and instance selection methods.

References

1. Schultz, M., Eskin, E., Zadok, F., Stolfo, S.: Data mining methods for detection of new malicious executables. In: Proceedings of the 22^nd IEEE Symposium on Security and Privacy. (2001) 38–49
2. Kolter, J., Maloof, M.: Learning to detect malicious executables in the wild. In: Proceedings of the 10^th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA (2004) 470–478
3. Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown malcode detection via text categorization and the imbalance problem. In: Proceedings of the 6^th IEEE International Conference on Intelligence and Security Informatics (ISI). (2008) 156–161
4. Santos, I., Penya, Y., Devesa, J., Bringas, P.: N-Grams-based file signatures for malware detection. In: Proceedings of the 11^{th} International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS. (2009) 317–320
5. Christodorescu, M.: Behavior-based malware detection. PhD thesis (2007)
6. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Proceedings of the 22^{nd} Annual Computer Security Applications Conference (ACSAC). (2006) 289–300
7. Moser, A., Kruegel, C., Kirda, E.: Limits of static analysis for malware detection. In: Proceedings of the 23^{rd} Annual Computer Security Applications Conference (ACSAC). (2007) 421–430
8. Kolbitsch, C., Holz, T., Kruegel, C., Kirda, E.: Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In: Proceedings of the 30^{th} IEEE Symposium on Security & Privacy. (2010)
9. Cavallaro, L., Saxena, P., Sekar, R.: On the limits of information flow techniques for malware analysis and containment. Lecture Notes in Computer Science **5137** (2008) 143–163
10. Santos, I., Brezo, F., Nieves, J., Penya, Y.K., Sanz, B., Laorden, C., Bringas, P.G.: Opcode-sequence-based malware detection. Lecture notes in computer science **5965** (2010) 35–43
11. Devesa, J., Santos, I., Cantero, X., Penya, Y.K., Bringas, P.G.: Automatic Behaviour-based Analysis and Classification System for Malware Detection. In: Proceedings of the 12^{th} International Conference on Enterprise Information Systems (ICEIS). (2010)
12. McGill, M., Salton, G.: Introduction to modern information retrieval. McGraw-Hill (1983)

13. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using cwsandbox. IEEE Security & Privacy **5**(2) (2007) 32–39
14. Ferrie, P.: Attacks on virtual machine emulators. In: Proc. of AVAR Conference. (2006) 128–143
15. Lee, T., Mody, J.: Behavioral classification. In: Proceedings of the 15^{th} European Institute for Computer Antivirus Research (EICAR) Conference. (2006)
16. Kent, J.: Information gain and a general measure of correlation. Biometrika **70**(1) (1983) 163
17. Bishop, C.: Pattern recognition and machine learning. Springer New York. (2006)
18. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International Joint Conference on Artificial Intelligence. Volume 14. (1995) 1137–1145
19. Breiman, L.: Random forests. Machine learning **45**(1) (2001) 5–32
20. Quinlan, J.: C4. 5 programs for machine learning. Morgan Kaufmann Publishers (1993)
21. Cooper, G.F., Herskovits, E.: A bayesian method for constructing bayesian belief networks from databases. In: Proceedings of the 7^{th} conference on Uncertainty in artificial intelligence. (1991)
22. Russell, S.J., Norvig: Artificial Intelligence: A Modern Approach (Second Edition). Prentice Hall (2003)
23. Geiger, D., Goldszmidt, M., Provan, G., Langley, P., Smyth, P.: Bayesian network classifiers. In: Machine Learning. (1997) 131–163
24. Lewis, D.: Naive (Bayes) at forty: The independence assumption in information retrieval. Lecture Notes in Computer Science **1398** (1998) 4–18
25. Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machines. Advances in Kernel Methods-Support Vector Learning **208** (1999)
26. Amari, S., Wu, S.: Improving support vector machine classifiers by modifying kernel functions. Neural Networks **12**(6) (1999) 783–789
27. Üstün, B., Melssen, W., Buydens, L.: Facilitating the application of Support Vector Regression by using a universal Pearson VII function based kernel. Chemometrics and Intelligent Laboratory Systems **81**(1) (2006) 29–40
28. Singh, Y., Kaur, A., Malhotra, R.: Comparative analysis of regression and machine learning methods for predicting fault proneness models. International Journal of Computer Applications in Technology **35**(2) (2009) 183–193
29. Kang, M., Poosankam, P., Yin, H.: Renovo: A hidden code extractor for packed executables. In: Proceedings of the 2007 ACM workshop on Recurring malcode. (2007) 46–53
30. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware. In: Proceedings of the 23^{rd} Annual Computer Security Applications Conference (ACSAC). (2007) 431–441
31. Sharif, M., Yegneswaran, V., Saidi, H., Porras, P., Lee, W.: Eureka: A Framework for Enabling Static Malware Analysis. In: Proceedings of the European Symposium on Research in Computer Security (ESORICS). (2008) 481–500
32. Ferrie, P.: Anti-Unpacker Tricks. In: Proc. of the 2^{nd} International CARO Workshop. (2008)
33. Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proceedings of the 28^{th} IEEE Symposium on Security and Privacy. (2007) 231–245