

A Machine-Learning-Based Framework for Supporting Malware Detection and Analysis

Alfredo Cuzzocrea[1,2(✉)], Francesco Mercaldo[3], and Fabio Martinelli[3]

[1] University of Calabria, Rende, Italy
alfredo.cuzzocrea@unical.it
[2] LORIA, Nancy, France
[3] IIT-CNR, Pisa, Italy
{francesco.mercaldo,fabio.martinelli}@iit.cnr.it

Abstract. Malware is one of the most significant threats in today's computing world since the number of websites distributing malware is increasing at a rapid rate. The relevance of features of unpacked malicious and benign executables like mnemonics, instruction opcodes, API to identify a feature that classifies the executables is investigated in this paper. By applying Analysis of Variance and Minimum Redundancy Maximum Relevance to a sizeable feature space, prominent features are extracted. By creating feature vectors using individual and combined features (mnemonic), we conducted the experiments. By means of experiments we observe that Multimodal framework achieves better accuracy than the Unimodal one.

Keywords: Malware · Machine learning · Opcode · Classification · Static analysis · Dynamic analysis · Hybrid analysis · Security

1 Introduction

Malware or malicious code is harmful code injected into legitimate programs to perpetrate illicit intentions. The primary source of infection is represented by the Internet. It also enables malcode to enter the system without users' knowledge. The primary sources used by the malicious users are free compilers, games, web browsers, etc. Most transactions performed using Internet attracted malware writers to gain unauthorized access to the machines to extract valuable information concerning the user and system. The stealthy malware exploits computer systems' vulnerabilities in order to open the computer systems' to remote monitoring. In particular, modern malware are classified in metamorphic or polymorphic [3]. Hence, it is required to develop a non-signature-based anti-malware detector for the effective detection of modern-day malware.

A. Cuzzocrea—This research has been made in the context of the Excellence Chair in Computer Engineering – Big Data Management and Analytics at LORIA, Nancy, France.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12951, pp. 353–365, 2021.
https://doi.org/10.1007/978-3-030-86970-0_25

Several classes distinguish the malwares by considering the breach policy and its introduction in the system. For example, metamorphic malwares have the ability to change as they propagate [26]. This is in contrast to cloned software, where all instances of a piece of software are identical. Metamorphism is usually considered by malware writers.

From a theoretical point of view, metamorphism enable malware to become undetectable To make viruses more resistant to emulation, virus writers have developed numerous advanced metamorphic techniques. For example, both decryptor and virus body are changed by metamorphic virus.

Furthermore, they change generation after generation. That is, a metamorphic virus changes its "shape" but not its behaviour. Instruction permutation, register re-assignment, code permutation using conditional instructions, no-operation insertion and many other are techniques implemented by malware writers.

The study of the malware is called malware analysis [17]. Malware analysis is an essential part of extracting run-time behaviour such as the API calls, import and export functions, etc. of the program. Hiding the malware execution from the malware analyzer is the primary battle. The counterpart that analyses the malware are projects tools such that Ether [13].

The motivation behind Ether is that it must not induce any side-effects to the host environment that are detectable by malware. The application of hardware virtualization extension such as Intel VT [16] are the building blocks of Ether. The target OS environment is the location where the system resides. In order to avoid this analysis modern malware implements anti-debugging, anti-instrumentation, and anti-VM techniques to thwart the run time analysis. Hence, there are no guest software modules answerable to detection, and there is no deprivation that arises from incomplete system emulation. Apart from providing anti-debugging facilities, de-armoring dynamically is a possible for using Ether.

In this paper, we present an innovative technique for malware detection. The experiments conducted on malware and benign executables are in Portable Executable (PE) format. By seeing the submissions received over the Virus Total [29], which performs offline scanning of malicious samples is the motivation for using Win32 PE executable. In particular, we define a set of features (as mnemonics, instruction opcodes, and API calls) obtained from benign and malicious executable.

2 Related Work

API calls have been used in the past research for modeling program behaviour [27,38] and for detecting malware [2,39]. The behaviour of the malicious programs in a specific malware class differs considerably from programs in other malware classes and benign programs is the main idea exploited in this paper.

In [25], the authors extracted prominent API calls statically using IDA-Pro. Thus, all the late-bounded API calls that are made using `GetProcAddress`, `LoadLibraryEx`, etc. are not taken into account. For packed malware the proposed approach is not working. Instead, the proposed approach is shown to

be effective against the aforementioned obfuscation techniques, encryption or decryption, and compression. It is also effective against new metamorphic versions of the same program as the dynamic behaviour remains the same, and only a single signature prepared for an entire malware family.

In [4], a mining algorithm for performing dynamic analysis and for detecting obfuscated malicious code is used by the authors. Despite the good performance, the computational time is of several seconds for each single program.

In [23] authors uses static analysis to detect system call locations and runtime monitoring to check all system calls made from a location identified during static analysis. In [27], dynamic monitoring for detecting worms and other exploits is used. The main limit of the proposed work is its applicability only for detecting worms and exploits that use hard-coded addresses of API calls. MetaAware [39] identifies patterns of system or library functions called from a malware sample statically to detect its metamorphic version.

In [12,36], the statistical properties of a set of metamorphic virus variants are represented by using a Hidden Markov Models (HMMs). The authors generate a database by using: Mass Code Generator (MPCGEN), Virus Creation Lab for Win32 (VCL32), Next Generation Virus Construction Kit (NGVCK), and Second Generation virus generator (G2). The HMM was trained on a family of metamorphic viruses and used to determine whether a given program is similar to the viruses the HMM represents.

In [19] a method to find the metamorphism in malware constructors like NGVCK, G2, IL_SMG, MPCGEN is proposed. The method executes each malware samples in a controlled environment like QEMU and it monitors API calls using STraceNTX.

Furthermore, they used the proposed method for estimating the similarity between malware constructors used in the study by computing the proximity indices amongst malware constructors. They found that the NGVCK generated malware were least similar to other constructors. The fact that the NGVCK generated malware invariants are metamorphic compared to all other malware constructors is highlighted by the authors.

In the presented literature, the main lack is the consideration of metamorphic viruses. The unimodal framework used earlier had many drawbacks as classification done based on a single modality. Only a single attribute type is used by the framework. Instead, in the present work, an innovative method for detecting obfuscated variants or mutants of malware dynamically is presented. Here we use a multimodal framework which involves learning based on multiple modalities. In particular, we extract prominent features using Minimum Redundancy and Maximum Relevance (mRMR) and Analysis of Variance (ANOVA). We also concatenate these prominent features, we developed the ensemble classifier from the various feature set which classifies into malware or benign.

3 Malware Detection and Analysis: A Machine Learning Methodology

3.1 Software Armoring

Software armoring or Executable Packing is the process of compressing/encrypting an executable file and prepending a stub that is responsible for decompressing/decrypting the executable and initiating its execution. When execution starts, the original executable code is unpacked and transferred by the stub for controlling it. Nowadays, the vast majority of the malware authors use packed executables to hide from detection. It is also due to software armoring that more and more false detection of malware comes up.

Whether the malware is armored or not is the first necessary knowledge needed for starting with its analysis. In the proposed methodology, Ether is considered as a tool for de-armoring. This is due to its characteristic to be not signature-based and transparent to malware. Signature-based unpacking tools like VMPacker [31], GUnPacker [14] requires that the executable is packed with standard packers. In order to extract the signature of a program we use PEiD [22]. In general, the authors of the malware write their stub and prepend it to the executable [5], moreover stymie's the signature-based detection. The two methods used commonly to unpack the actual malicious code are: (1) to reverse engineering the unpack algorithm, then to use a scripting language to re-implement it and finally to run the script on the packed file as in OllyDbg [21] or IDAPro [15]; (2) at run time, use a breakpoint just previously the malicious code which requires manual intervention [28].

Ether detects all the writes to memory, a program does and dumps the program back in the binary executable form. A hash-table for all the memory maps is created. Whenever a write to a slot in the hash table is available then a Original Entry Point is reported. This is the starting point of execution of a packed executable.

3.2 Feature Extraction

In our approach, we used API calls to measure the dynamic malware analysis and mnemonic or opcode, instruction opcode, 4-gram mnemonic as a measure of static malware analysis. The features are extracted by using the following methods: (1) Usage of VeraTrace, the API Call tracer [30]. Veratrace is an Intel PIN-based API call tracing program for Windows. API calls can be traced by this program of de-armored programs obtained from Ether. A Frequency Vector Table (FVT) is obtained by parsing the output of Veratrace for final API traces. (2) Mnemonics and instruction opcode features are extracted using an open source tool namedscriptsizeObjDump [20]. The FVT of mnemonics and instruction opcode are obtained by parsing both these features. (3) The instruction opcode and mnemonics features are extracted by running scriptsizeObjDump [20] (a freely available open-source tool). A parser for both these features has been developed for obtaining the FVT of mnemonics and instruction opcode.

In particular, only a subset of the features mentioned above is used for our analysis. The selection is made by using feature selection methods. Thus, the selected features are the ones that most represent variation in the data.

3.3 Normalization

In statistics, a standard score indicates the number of standard deviations an observation or datum is above or below the observed mean. By subtracting the feature mean from an individual feature raw score and dividing this difference by the feature standard deviation the score is obtained. This procedure is usually called standardizing or normalizing. The frequency vector table is a table populating feature as the column and their frequencies in a particular sample as a row. The majority of the classifier and feature selection methods need normalization and discretization into states of the frequency vector table. In the proposed approach, the Z-Score is chosen for normalization.

Z–Score. Standard scores are also called z-values, z-scores, normal scores, and standardized variables, the use of "Z" is because the normal distribution is also known as the "Z distribution". The comparison of a sample to a standard normal deviate (standard normal distribution, with $\mu = 0$) is their main usage. Nevertheless, as shown in Eq. 1, they can be defined without the assumption of normality. In statistics, both the mean and the standard deviation are used for standardizing a general feature X.

$$Z_j = \sum_{i=1}^{n} \frac{X_i^j - \mu_j}{\sigma_j}, \tag{1}$$

where μ_j is the mean of all the frequencies obtained from samples of the jth feature, $\sigma_j = \sqrt{\text{Var}(X)}$ is the standard deviation of the jth feature, Z_j is the Z–Score of the jth feature, X_i^j is the frequency of jth feature in i^{th} sample and n is the total number of samples. The following equation defines the variance:

$$Var(X_j) = \sum_{i=1}^{n} \frac{(X_i^j - \mu_j)^2}{n}. \tag{2}$$

3.4 Feature Selection

Given the financially-motivated nature of these threats, methods of recovery mandate more than just remediation. In general, understanding the runtime behaviour of modern malware is the main need. It includes basic artifacts or features of any Portable Executable like the mnemonics, API Call, instruction opcode, etc. As Fig. 1 shows, the proposed methodology consider mnemonics, instruction opcode, API Calls [19] and 4-gram mnemonic [24] for Feature Fusion [18] to analyze a malware.

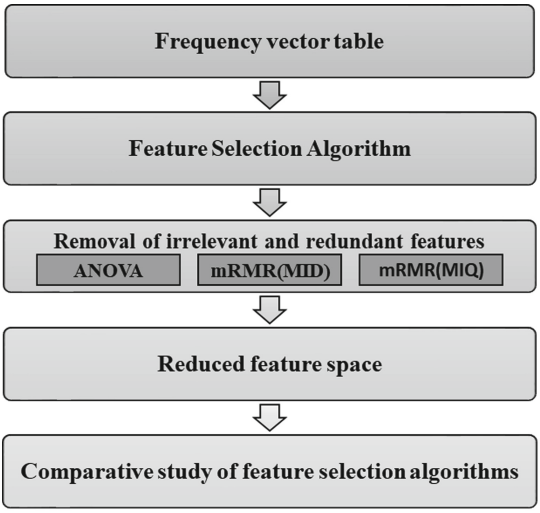

Fig. 1. Feature selection

Today's malware is increasing at a tremendous rate, resulting in the proliferation of high dimensional data getting accumulated, thus challenging state-of-the-art data mining techniques. The importance of feature selection in data mining is becoming widely recognise [35]. This is due to the high dimension of the data extracted. For example, the mnemonics features includes x86 instructions (such as push, xor, pop, test etc.) instruction opcodes, API Call (`GetProcAddress`, `LoadLibrary` etc.). Feature selection effectively reduces the data dimensionality by removing redundant and irrelevant features.

Many feature selection algorithms have been developed for different purposes. Moreover, each algorithm is characterized by the usage of its criterion for feature selection that will be explained later. These techniques are of paramount importance if several irrelevant features are considered otherwise learning models cannot generalize in identifying a new sample. Besides the selection of features, we have also presented a comparative study of the algorithms used for feature selection.

3.5 Classification

Databases are rich with hidden information that can be used for intelligent decision making. In order to extract to extract models describing classes, classification is used. It has been successfully applied in several settings (e.g., [9]). This kind of data analysis enable us to better understand the pattern in the data. Classification predicts categorical (discrete, unordered) labels. The Support Vector Machine Classifier [35] (SVM) is used to classify unknown samples. SVM belongs to the class of supervised learning methods. In the proposed appli-

cation it has been trained by using 2000 different software (1000 malware and 1000 benign samples).

4 Malware Detection and Analysis: Framework Architecture

To classify unknown samples, we used malwares such as viruses, trojans, backdoor, and worms obtained from the VX Heavens repository, freely available for research purpose [32]. The benign programs are considered from the Windows library. The computation of the of features like Mnemonic, instruction opcode, API calls [1], and 4-gram mnemonic needs the samples to be unpacked.

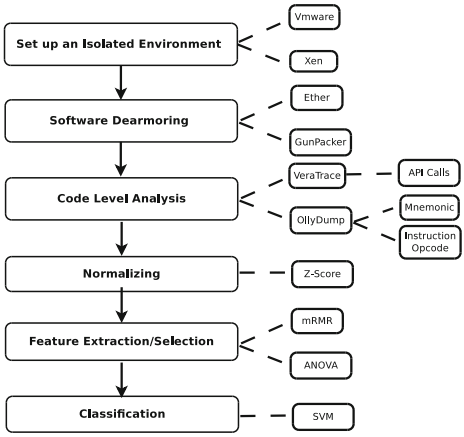

Fig. 2. Architecture

So, the executable files were made to run on a hardware virtualized machine such as Xen [37]. The actual host machine is not affected by the viruses during the dynamic API tracing step is the main advantage of using an emulator.

In Fig. 2, we show the complete architecture of the proposed methodology. A total of 1000 malware samples and 1000 benign samples are considered. Then, they are executed on the hardware virtualized machine using Veratrace to capture the API calls. The emulator moves all the text files with the information of the API calls to the actual host OS. The frequency of each API call is found. The following sections discuss the concepts used here.

5 Feature Extraction Methods

Once the samples unpacked, we carry out the code-level analysis of the samples. Features are extracted using parsers for each feature, as described in Fig. 3.

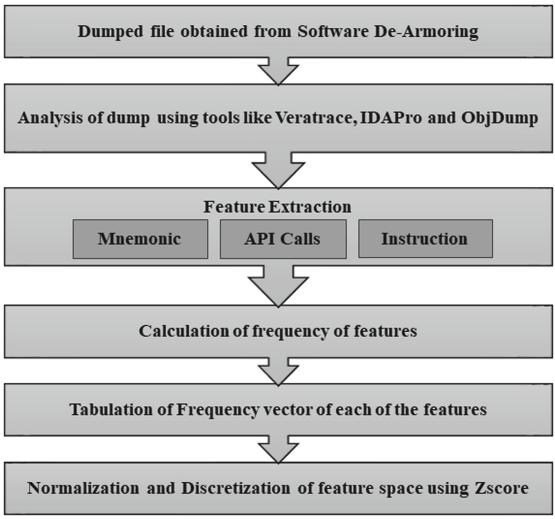

Fig. 3. Feature extraction

5.1 API Calls Tracing

The Windows API, informally WinAPI, is Microsoft's core set of application programming interfaces (APIs) available in the Microsoft Windows operating systems. In Windows, a set of API calls are needed in order for an executable program to perform its assigned work. For instance, some of the API calls for file management are `OpenFile` that creates, opens, reopens, or deletes a given file; `DeleteFile` that deletes an existing file; `FindClose` that closes a file search handle that is opened by `FindFirstFile`, `FindFirstFileEx`, or `FindFirstStreamW` function; `FindFirstFile` that searches a file inside a directory or look for a subdirectory name that matches a specified name;`GetFileSize` that retrieves, in bytes, the size of a specified file. Thus, no executable program can run without the API calls. Hence, a good measure to record the behaviour of an executable is by using API calls made by an executable.

`Veratrace` is a Windows API Call Tracer. It can trace all the calls made by a process to the imported functions from a DLL. Veratrace traces the API calls by mandating the unpacking of the samples. Whenever packed, the import table would be populated with `GetProcAddress()` and `LoadLibrary()`-like API calls. Trace files contain information about all the API Calls made by these executables. For the purposes of parsing all traces, a parser was designed. API Traces are passed to the parser as input yielding the following output: (1) All API traces of malware and benign programs are kept in malware_api and benign_api directory, respectively. (2) API Trace parser takes these input directories and creates a file named api.csv. FVT contains a grid representing the frequencies of all the features.

5.2 Mnemonic, Instruction Opcode and 4-Gram Mnemonic Trace

Following mnemonics, instruction opcode, 4-gram mnemonic form the structural aspect of the program. In a matter of tracing them, a static analysis of the sample is required. Alongside performing Static Analysis using the open-source ObjDump tool, we developed a customized parser to filter mnemonics and create a feature vector table with the occurrence of mnemonics as elements of the vectors. Likewise, we carry out the vector representation for opcodes.

6 Classification Procedure

Data classification is a two step process.

1. In the first step, a classifier describing a predetermined set of data classes or concepts is built. The first step being the learning step (or training phase), a classification algorithm, in that step, builds the classifier by analyzing or learning from a training set made up of database tuples and their associated class labels. A tuple or feature, X, is represented by an n-dimensional attribute vector, $X = (A_1\&A_2\&...\&A_n)$, where, n depicts measurements made on the tuple from n database attributes, respectively, A_1, A_2, ..., A_n. Each tuple, X, is assumed to belong to a predefined class determined by another database attribute called the class label. The class label attribute is represented by discrete unordered values. The individual tuples making up the training set namely the training tuples are selected from the database under analysis. The class label of each training tuple is known as supervised learning as the classifier already told, to which class a particular tuple belongs.
2. In the second step, the classification model is used. Test data were created through test tuples and their class. This data was not used in the classifier's training. The percentage of test set tuples that are correctly classified by the classifier is identified as the accuracy of a classifier on a given test set. The associated class label of every test tuple compared with the learned classifiers class prediction for the concerned tuple.

The SVM model is exploited in our approach. Linear as well as nonlinear data are classified by the SVM classification method. It employs a nonlinear mapping to turn the original training data into a higher dimensional data. Within this new dimension, it searches for the linear optimal separating hyperplane (a decision boundary separating the tuples of one class from another). With an appropriate nonlinear mapping to a sufficiently high dimension, a hyperplan can always separate data from two classes. The SVM defines this hyperplane through support vectors (essential training tuples) and margins (defined by the support vectors).

Although the training time of even the fastest SVMs prolonge, they are highly accurate due to their ability to model complex nonlinear decision boundaries. A short description of the learned model is provided by the support vectors.

Our approach is a two-class problem as we have to classify the samples as either benign or malware samples.

We obtain a nonlinear SVM by extending the approach for linear SVMs as follows. In the first step, original input data is transformed into a higher dimensional space using a nonlinear mapping. Once the data has transformed into the new space, a linear separating hyperplane in the new space is searched by the next step. We end up with a quadratic optimization problem that is solved using the linear SVM formulation [33]. This way, a nonlinear, separating surface in the original space is obtained, which is represented by the maximal marginal hyperplane found in the new space. In addition, a nonlinear mapping function is used to transform the training tuples. It turns out that it is mathematically equivalent to applying a kernel function to the original input data. In this way, in the original input space, potentially in a much lower dimensionality the performance of the calculation is done. *Gaussian Radial Basis Function kernel* (RBF) was used.

Other than the SVM Classifier, our proposed method classifies unseen samples using Random Forest, which is an ensemble classifier and also AdaBoost, which uses out–of–bag(oob) data set. We present a comparative study of SVM, Random Forest and AdaBoost. We apply classification using WEKA [34].

7 Experimental Assessment and Analysis

In this section, we discuss the experiment's setup, results obtained and the analysis of the result. We collected a total of 2000 Portable Executables which consists of 1000 malware samples gathered from sources VxHeaven (650) [32], User Agency (250), Offensive Computing (100) [30], besides that we collected benign samples from Windows XP System32 Folder (450), Windows7 System32 Folder (100), MikTex/Matlab Library (400) and Games (50).

We extracted mnemonics or opcodes from 2000 samples. In Fig. 4, we show the experimental results obtained from feature reduction using mRMR (MID and MIQ) and ANOVA. We derived these results after sample classification using SVM, AdaBoost, Random Forest, J48. Five mnemonic based models were constructed at a variable length, starting from 40 to 120 at an interval of 20. Among these five models, the best result with a strong positive likelihood ratio of 16.38 for the feature-length 120 using Random Forest was provided by ANOVA. Low error rate and speed are the main advantages of this model. Nevertheless, mnemonic based features can be effortlessly modified using code obfuscation techniques. Generally, metamorphic and polymorphic malware employ these obfuscation techniques to bypass the detectors provided by current anti-malware free and commercial software.

8 Conclusion and Future Work

In this paper, we proposed a new hybrid approach for malware detection, integrating the prominent attributes realized from a set of unimodal classifiers, which

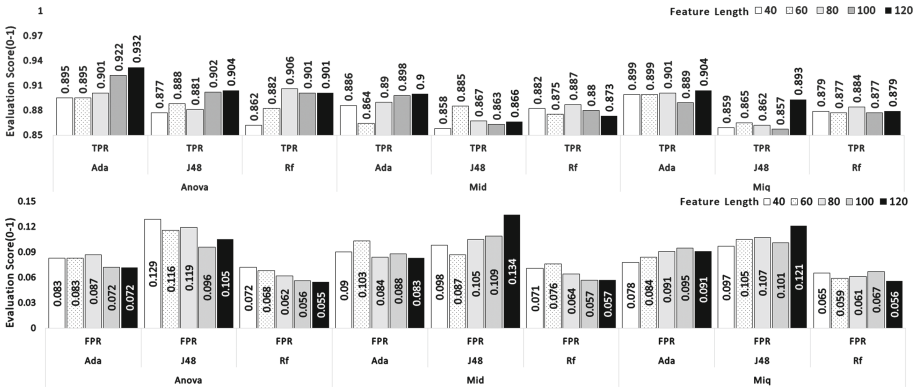

Fig. 4. Performance of mnemonic features

employ both static and dynamic model for malware analysis. Here, a substantial performance gain over the set of unimodal classifiers, was derived from the multimodal feature fusion technique we followed and from which it derived its features. Our proposed solution consistently yielded superior outputs in comparison with the unimodal classifier set we employed, with an accuracy rate exceeding 95% on average.

In future, our method can be extended by analyzing the malware and benign samples using ensemble features and classifying the unseen samples using ensemble classifiers by using different voting schemes. The unseen samples are voted by each classifier from the pool as malware or benign, and accordingly, the sample with the maximum number of votes would be classified. Currently our data set consists of bi-class, malware or benign, a plausible extension to the classification scheme in the future is toward a multi-class classification. Another line of future research predicates the integration of our proposed framework with novel features dictated by the emerging big data trend (e.g., [6–8,10,11]).

Acknowledgements. This research has been partially supported by the French PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE.

References

1. Alkhateeb, E.M., Stamp, M.: A dynamic heuristic method for detecting packed malware using naive bayes. In: International Conference on Electrical and Computing Technologies and Applications (ICECTA), pp. 1–6. IEEE (2019)
2. Bergeron, J., Debbabi, M., Erhioui, M.M., Ktari, B.: Static analysis of binary code to isolate malicious behaviors. In: WETICE 1999: Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises, Washington, DC, USA, pp. 184–189. IEEE Computer Society (1999)
3. Bulazel, A., Yener, B.: A survey on automated dynamic malware analysis evasion and counter-evasion: pc, mobile, and web. In: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, pp. 1–21 (2017)

4. Christodorescu, M., Jha, S., Kruegel, C.: Mining specifications of malicious behavior. In: ESEC-FSE 2007: Proceedings of the the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 5–14, New York, NY, USA. ACM (2007)
5. Chuan, L.L., Yee, C.L., Ismail, M., Jumari, K.: Automating uncompressing and static analysis of conficker worm. In: 2009 IEEE 9th Malaysia International Conference on Communications (MICC), pp. 193–198. IEEE (2009)
6. Cuzzocrea, A.: Improving range-sum query evaluation on data cubes via polynomial approximation. Data Knowl. Eng. **56**(2), 85–121 (2006)
7. Cuzzocrea, A., Matrangolo, U.: Analytical synopses for approximate query answering in OLAP environments. In: Galindo, F., Takizawa, M., Traunmüller, R. (eds.) DEXA 2004. LNCS, vol. 3180, pp. 359–370. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30075-5_35
8. Cuzzocrea, A., Moussa, R., Xu, G.: OLAP*: effectively and efficiently supporting parallel OLAP over big data. In: Cuzzocrea, A., Maabout, S. (eds.) MEDI 2013. LNCS, vol. 8216, pp. 38–49. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41366-7_4
9. Cuzzocrea, A., Mumolo, E., Fadda, E., Tessarotto, M.: A novel big data analytics approach for supporting cyber attack detection via non-linear analytic prediction of IP addresses. In: Gervasi, O., et al. (eds.) ICCSA 2020. LNCS, vol. 12249, pp. 978–991. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58799-4_70
10. Cuzzocrea, A., Saccà, D., Serafino, P.: A hierarchy-driven compression technique for advanced OLAP visualization of multidimensional data cubes. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2006. LNCS, vol. 4081, pp. 106–119. Springer, Heidelberg (2006). https://doi.org/10.1007/11823728_11
11. Cuzzocrea, A., Serafino, P.: LCS-Hist: taming massive high-dimensional data cube compression. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp. 768–779 (2009)
12. Damodaran, A., Di Troia, F., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. J. Comput. Virol. Hacking Tech. **13**(1), 1–12 (2017)
13. Ether: http://ether.gtisc.gatech.edu/
14. Gunpacker. http://www.woodmann.com/collabarative/tools/
15. Ida Pro: http://www.hex-rays.com/idapro/
16. Intel: http://www.intel.com/
17. Mandiant: http://www.mandiant.com/
18. Masud, M.M., Khan, L., Thuraisingham, B.: A hybrid model to detect malicious executables. In: Proceedings of IEEE International Conference on Communications, ICC 2007, pp. 1443–1448. IEEE (2007)
19. Nair, V.P., Jain, H., Golecha, Y.K., Gaur, M.S., Laxmi, V.: Medusa: metamorphic malware dynamic analysis using signature from api. In: Proceedings of the 3rd International Conference on Security of Information and Networks, SIN 2010, pp. 263–269, New York, NY, USA. ACM (2010)
20. Objdump. https://ubuntu.pkgs.org/16.04/ubuntu-universe-amd64/dissy_9-3.1_all.deb.html
21. Ollydbg. http://www.ollydbg.de
22. Peid: http://www.peid.info
23. Rabek, J.C., Khazan, R.I., Lewandowski, S.M., Cunningham, R.K.: Detection of injected, dynamically generated, and obfuscated malicious code. In: WORM 2003: Proceedings of the 2003 ACM workshop on Rapid malcode, pp. 76–82, New York, NY, USA. ACM (2003)

24. Santos, I., Penya, Y.K., Devesa, J., Bringas, P.G.: N-grams-based file signatures for malware detection. ICEIS (2), **9**, 317–320 (2009)
25. Sathyanarayan, V.S., Kohli, P., Bruhadeshwar, B.: Signature generation and detection of malware families. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 336–349. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70500-0_25
26. Sharma, A., Sahay, S.K.: Evolution and detection of polymorphic and metamorphic malwares: a survey. arXiv preprint arXiv:1406.7061 (2014)
27. Sun, H.-M., Lin, Y.-H., Wu, M.-F.: API monitoring system for defeating worms and exploits in MS-windows system. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 159–170. Springer, Heidelberg (2006). https://doi.org/10.1007/11780656_14
28. Vilkeliskis, T.: Automated unpacking of executables using dynamic binary instrumentation (2009)
29. Virus Total. http://www.virustotal.com/stats.html
30. Veratrace: http://www.offensivecomputing.net/
31. Vmpacker. http://www.leechermods.com/
32. VX heavens. http://vxheaven.0l.wtf/
33. Wadkar, M., Di Troia, F., Stamp, M.: Detecting malware evolution using support vector machines. Expert Syst. Appl. **143**, 113022 (2020)
34. Open source Machine Learning Software Weka. http://www.cs.waikato.ac.nz/ml/weka/
35. Witten, I.H.: Frank, and E. Morgan Kaufmann, Practical Machine Learning Tools and Techniques with Java Implementation (1999)
36. Wong, W., Stamp, M.: Hunting for metamorphic engines. J. Comput. Virol. **2**(3), 211–229 (2006)
37. Xen: http://www.xen.org
38. Zhang, B., Yin, J., Hao, J.: Using fuzzy pattern recognition to detect unknown malicious executables code. In: Wang, L., Jin, Y. (eds.) FSKD 2005. LNCS (LNAI), vol. 3613, pp. 629–634. Springer, Heidelberg (2005). https://doi.org/10.1007/11539506_78
39. Zhang, Q., Reeves, D.S.: Metaaware: identifying metamorphic malware. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), pp. 411–420. IEEE (2007)