AUTOMATIC BEHAVIOUR-BASED ANALYSIS AND CLASSIFICATION SYSTEM FOR MALWARE DETECTION

Jaime Devesa, Igor Santos, Xabier Cantero, Yoseba K. Penya and Pablo G. Bringas
S^3Lab, Deusto Technological Foundation, Bilbao, Spain

Keywords: Security, Malware detection, Machine learning, Data-mining.

Abstract: Malware is any kind of program explicitly designed to harm, such as viruses, trojan horses or worms. Since the amount of malware is growing exponentially, it already poses a serious security threat. Therefore, every incoming code must be analysed in order to classify it as malware or benign software. These tests commonly combine static and dynamic analysis techniques in order to extract the major amount of information from distrustful files. Moreover, the increment of the number of attacks hinders manually testing the thousands of suspicious archives that every day reach antivirus laboratories. Against this background, we address here an automatised system for malware behaviour analysis based on emulation and simulation techniques. Hence, creating a secure and reliable sandbox environment allows us to test the suspicious code retrieved without risk. In this way, we can also generate evidences and classify the samples with several machine-learning algorithms. We have developed the proposed solution, testing it with real malware. Finally, we have evaluated it in terms of reliability and time performance, two of the main aspects for such a system to work.

1 INTRODUCTION

Malware is any kind of code explicitly designed with harmful intentions, such as viruses, trojan horses or worms. Malware represents a high-priority issue to security researchers and poses a major threat to the privacy of computer users and their information.

Still, the traditional approach to analyse malware requires that a human analyst manually performs the tests and extracts the information in order to classify the sample (Moser et al., 2007). Unfortunately, due to the tremendous growth of malicious code, antivirus companies receive everyday thousands of suspicious files that have to be analysed and classified as malware or benign software. Hence, a reliable and fast automation of the analysis and classification is a crucial point to be able to cope with this threat.

With this scenario in mind, there are two malware analysis approaches: *static analysis* (Carrera and Erdélyi, 2004) which is performed without actually executing the file, only observing the binary looking for suspicious patterns, and *dynamic analysis* (Christodorescu et al., 2007; Rieck et al., 2008) which implies running the sample in an isolated and controlled environment monitoring its behaviour.

Against this background, we propose here an automatic system for malware behaviour analysis based on emulation and simulation techniques. We advance the state of the art in three main ways. First, we propose a sandbox to monitor program executions within a contained environment. Second, we present and describe a method for feature extraction from behaviour reports, and we train different machine-learning classifiers in order to provide detection of unknown malware instances. Last but not least, we evaluate the system in terms of reliability and time performance.

2 SYSTEM ARCHITECTURE

2.1 System Manager

The System Manager is the main process of the entire system, and it is responsible for orchestrating the other components.

When the System Manager starts, it launches *n* Qemu virtual machines and creates a TCP server in order to establish communications and synchronise other processes. Moreover, the suspicious files and the generated behaviour trace logs are exchanged with the sandboxes via these communication channels. Once any of the sandboxes finishes the analysis of a file, the behaviour trace is copied into a net share

Devesa J., Santos I., Cantero X., K. Penya Y. and G. Bringas P. (2010).
AUTOMATIC BEHAVIOUR-BASED ANALYSIS AND CLASSIFICATION SYSTEM FOR MALWARE DETECTION.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Artificial Intelligence and Decision Support Systems*, pages 395-399
DOI: 10.5220/0002895203950399

folder. Then, the System Manager is informed and negotiates the extraction of features with the regular expression parser. Finally, the information gathered is stored in a database with the aim of further classification.

2.2 Behaviour Monitoring

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.

Besides, as 87.9 % of users work with MS-based platforms, most of the malware is aimed at these operating systems. Therefore, we have developed a sandbox with the purpose of dynamically analyse Windows Portable Executable (PE) files (Pietrek, 1994). To this extent, the suspicious files are executed inside an emulated environment, and relevant Windows API calls are logged, showing the program's behaviour.

Therefore, we propose 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. In this way, we describe now the two main platforms of our sandbox solution.

First, Wine is an open-source and complete re-implementation (simulation) of the Win-32 Application Programming Interface (API). Hence, it allows Windows PE files to run as-if-natively under Unix-based operating systems. Moreover, the main reason of our choice, is that as being open-source we can modify and improve the code to adjust it to our needs, and therefore we can log any call to the Win-32 API done by the malware.

Second, 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. Plus, they can be fixed easily (Ferrie, 2006).

To this extent, we have made several improvements in Wine. First, we have modified Wine's source-code to write every call done by a process (identified by its *PID*) to the Windows API (divided in families, i.e. *registry*, *memory* or *files*) into a log, specifying parameters' state before and after the functions body. Thereby, we can obtain a complete and homogenous trace with all processes behaviour, without any interference with the system.Second, we have made several modifications to the basic Wine installation by adding various Windows XP system dlls and creating additional registry keys and folder structures.

In this way, we have obtained the most transparent and similar system to a Windows O.S. as possible in order to be able to execute the largest possible variety of programs.

2.3 Feature Extraction

Whenever the input data to machine learning and classification methods is too large and complex, it is necessary to transform this data into a reduced representation set of carefully chosen features (feature vector), a process known as *feature extraction*.

Furthermore, for each malware sample analysed in the sandbox, we may obtain a complete *in-raw* trace with its detailed behaviour. These reports are not suitable for machine learning applications as these usually work with vectorial data. Hence, with the aim of automatically extracting relevant information in vector format from the traces, we developed regular expression rules and a parser to identify them. This approach is a powerful and fast tool for identifying patterns of characters within a text (Friedl, 2006), in our case, the detailed trace log.

To this end, we have defined different specific actions taken by the program as regular expression rules, with the help of an expert, in order to conform the knowledge base. In this way, the creation of new rules becomes very simple and intuitive and the system is easily improvable.

Moreover, most of the actions defined are characteristic of malware but there are both benign and malicious behaviour rule definitions.

Therefore, we have converted all the information achieved in a vector, parsing the behaviour reports with the defined regular expressions.

To this extent, we defined a program behaviour as a vector made up of the aforementioned characteristics. More specifically, we represented an executable as a vector $\vec{v}$ that is composed by binary characteristics c, where c can be either 1 (true) or 0 (false), $\vec{v} = (c_1, c_2, c_3, ..., c_{n-1}, c_n)$ and n is the number of 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 (Lee and Mody, 2006). Likewise, both *in-raw* trace log and feature sequence for each analysed executable are stored in a database for further treatment.

3 EXPERIMENTS AND RESULTS

3.1 Datasets

For the following experiments, we have used two different datasets for testing the system: a malware dataset and a benign software dataset. First, we have downloaded a big malware collection from *VxHeavens* website (VX-Heavens, 2009) conformed by different malicious code such as trojan horses, virus or worms. Even though they were already named with their family and variant names, we have scanned them using AVG Anti-virus to guarantee the correctness of the labelling.

Hereafter, we have analysed the whole malware dataset in our system. Table 1 details the number of files per family, the number of files that have been correctly analysed by our system and the correctly analysed percentage. We have considered a correct analysis when at least one of the features was activated, this is, when at least one action covered in the rules was performed. In summary, the average success rate was 85,16%, which is a good result taking into account that our system relies in a single execution and that some of the malware executables can detect that they are being executed in an emulated environment (Ferrie, 2006). Further, we have removed from the dataset the 379 malware files that were not correctly analysed, since no features were activated.

Table 1: Results for the malware analysed in the system.

Families	Files	Correctly analysed	Rate (%)
Agobot	340	250	73.53
Bagle	127	124	97.64
MyDoom	47	43	91.49
Rbot	1161	950	81.83
Sdbot	879	808	91.92
TOTAL	**2554**	**2175**	**85.16**

In addition, we have collected diverse executable files from a recent installation of a Windows XP system without internet connection, in order to conform the benign software dataset. In this way, we have ensured that there were not infected files. For a further assurance, we have also scanned them using AVG Anti-virus. The whole benign software dataset has been correctly labelled as legitimate executables.

Finally, we have created a 1500 balanced dataset conformed by 750 random malware files chosen within the 2175 correctly analysed malware samples, and 750 non-malware files. It is proven that this balancing renders the evaluation of the machine learning methods more reliable (Batista et al., 2004).

3.2 Time Performance

In order to evaluate the system's time performance, we have analysed a corpus of 500 files from the malware dataset using three different configurations: with one to four sandboxes being executed simultaneously in the same machine. The default analysis timeout for each file was initially estimated at 20 seconds, but most of them finished their execution before reaching this limit.

In this way, Table 2 shows the obtained results using the four configurations. Note that the improvement percentage has been calculated based on the single sandbox configuration.

As we increment the amount of computers devoted to the analysis, the performance grows as well, in a geometrical progression.

Table 2: Analysis of 500 files time performance results.

VM	Total time (sec)	Time per file (sec/file)	Improvement percentage
1	9555	19.11	-
2	4640	9.28	51.44
3	3390	6.78	64.53
4	2815	5.63	70.54

In conclusion, we had an excellent time performance running 4 sandbox instances for each computer dedicated to execute these components.

3.3 Malware Detection Experiments

We have conducted an experiment focused on classifying generic malware without taking into account which family of malware belonged to. Therefore, we can provide detection of unknown malicious instances.

In particular, we have trained and tested Näive Bayes, Decision Trees and Support Vector Machine classifiers in order to evaluate them. Choosing these machine-learning classifiers was not a trivial decision since they have already been widely applied to malware detection (Ye et al., 2008). To this extent, we have performed the following steps:

- **Cross Validation.** Despite the small dataset, we had to use as much of the available information in order to obtain a proper representation of the data. To this end, K-fold cross validation is usually used in machine-learning experiments (Bishop, 2006).

Thereby, for each classifier we tested, we performed a k-fold cross validation (Kohavi, 1995) with k = 10. In this way, our dataset was 10 times split into 10 different sets of learning (66% of the total dataset) and testing (34% of the total data).

- **Learning and Testing the Model.** We made a learning phase where each classifier acquires the required knowledge using the learning dataset. Hereafter, the algorithm classified the rest of the dataset. We used different configurations for each machine-learning classifier:

  1. *Näive Bayes:* We employed EM parametrical learning for training the Bayesian network.
  2. *Random Forest:* We run the training algorithm with a number of random trees of 100.
  3. *J48:* We trained this classifier with a confidence factor of 0.25 and a minimal number of objects of 2.
  4. *SMO:* We used the Weka implementation of SVM trained with a polynomial kernel and a complexity factor of 1.

- **Evaluation of the Model.** In order to evaluate each classifier's capability we measured the True Positive Ratio (TPR) that is the number of malware instances correctly detected, divided by the total amount of malware files.

  Moreover, we measured the False Positive Ratio (FPR), that is the number of benign executables that were misclassified as malware divided by the total number of benign files Note that keeping this measure low is highly important in commercial antivirus for user satisfaction.

  Furthermore, we measured Total Accuracy, that is the total number of the classifier's hits divided by the number of instances in the whole dataset.

Table 3 shows the obtained results. In this way, every classifier that we tested, achieved overall good results in accuracy, malware detection (TPR) and low false alarms (FPR).

Specifically, both of the decision tree algorithms that we tested outperformed the rest of the classifiers in terms of TPR and total accuracy. These two algorithms, *J48* and *random forest*, achieved a very high accuracy (more than 96%) and high detection ratio (more than 0.94).

Still, if we focus on false positive ratio, random forest presented the best results. Therefore, it seems to be the best option for detecting generic malware.

Furthermore, the good results obtained of the proposed system renders it a very good approach for detecting unknown malware. In this way, we think that this system can help to identify faster the malware variants for further analysis and creation of their signatures.

Table 3: Results per classifier.

Classifiers	T. Accuracy (%)	TPR	FPR
Naive Bayes	91.6	0.852	0.02
Random Forest	96.2	0.937	0.013
J48	96	0.943	0.023
SMO	95.4	0.931	0.023
Average	**94.8**	**0.915**	**0.019**

4 RELATED WORK

Due to the great growth of malware in the last few years, an extensive work to cope with this malicious code plague has been published.

To deal with obfuscation techniques, various dynamic analysis solutions based in sandbox environments have been developed, using different techniques for monitoring the malware's behaviour: TT-Analyze (Bayer et al., 2006) like our sandbox, uses the Qemu PC emulator, but it runs a Window operating system combined with the API hooking technique. On the other hand, CWSandbox (Willems et al., 2007) relies in DLL injection to trace and monitor all relevant system calls, and to generate a machine-readable report.

Further, two different approaches for malicious executables classification have been proposed (Rieck et al., 2008): discrimination between families of malware, and discrimination between specific malware instances and benign software.

First, recent work has focused in discrimination between different malware families using clustering of behaviour reports (Lee and Mody, 2006), transforming reports into sequences for later grouping them into clusters, which finally represent different families of malicious programs. Nevertheless there are some limitations taking into account that clustering methods have unsupervised nature, with its inherent problems. Another recent approach (Rieck et al., 2008) monitors the suspicious file execution and builds a vector space model with the frequencies of the contained strings as features to classify malicious behaviour into malware families with Support Vector Machines (SVM). Second, in (Christodorescu et al., 2007) it is presented an approach to discriminate between malware and benign software based on mining differences between malicious programs and benign executables behaviour reports.

5 CONCLUSIONS AND FUTURE WORK

In this paper we have presented a distributed and automatic system for malware detection based on the dynamic analysis of suspicious files and the later classification into malware and benign software groups. For the behaviour analysis we have developed a new sandbox based in Qemu and Wine.

Focusing on the obtained results, we consider that the features extracted from the behaviour logs are suitable for training classifiers in order to detect malware. Hence, we have achieved a high detection rate (94.8 %) and a low percentage of false positive ratio (0.019). On the other hand, we have shown that the system presents a great time performance. Still, our system also presents some limitations, as follows.

First, in order to take advantage over antivirus researchers, malware writers have included diverse evasion techniques (Ferrie, 2006) 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 in order to solve the bugs and not to be vulnerable to the above-mentioned techniques.

Second, it is possible that some malicious actions are only triggered under specific circumstances depending on the environment, so the rely on a single program execution will not manifest all its behaviour. This is solved with a technique called *multiple execution path* (Moser et al., 2007), making the system able to obtain different behaviours displayed by the suspicious executable.

Finally, as Wine is not a completed project, it does not perfectly simulate a Windows operating system. Moreover, it grows fast, improving its own reimplementation of the Win-32 API in each published version. In conclusion, we might install the last stable version available in order to maintain the system up-to-date and to have a greater capacity of emulation.

As future lines of work, we plan to expand the features identified by the system until now. This is, defining more regular expression rules to perceive both malicious and legitimate behaviour since, in this way, the classification will give better results. Moreover, combining our approach with static analysis techniques may improve the obtained results.

REFERENCES

Batista, G., Prati, R., and Monard, M. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29.

Bayer, U., Kruegel, C., and Kirda, E. (2006). TTAnalyze: A tool for analyzing malware. In *Proceedings of the 15th Annual Conference of EICAR*.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer New York.

Carrera, E. and Erdélyi, G. (2004). Digital genome mapping–advanced binary malware analysis. In *Proceedings of the 14th Virus Bulletin Conference*, pages 187–197.

Christodorescu, M., Jha, S., and Kruegel, C. (2007). Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the ESEC and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 5–14.

Ferrie, P. (2006). Attacks on virtual machine emulators. In *Proc. of AVAR Conference*, pages 128–143.

Friedl, J. (2006). *Mastering regular expressions*. O'Reilly Media, Inc.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145.

Lee, T. and Mody, J. (2006). Behavioral classification. In *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR) Conference*.

Moser, A., Kruegel, C., and Kirda, E. (2007). Exploring multiple execution paths for malware analysis. In *Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 231–245.

Pietrek, M. (1994). Peering Inside the PE: A Tour of the Win32 (R) Portable Executable File Format. *Microsoft Systems Journal*, 3.

Rieck, K., Holz, T., Willems, C., Dussel, P., and Laskov, P. (2008). Learning and Classification of Malware Behavior. *Lecture Notes in Computer Science*, 5137:108–125.

VX-Heavens (2009). VX heavens. Online: http://vx.netlux.org/.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5(2):32–39.

Ye, Y., Wang, D., Li, T., Ye, D., and Jiang, Q. (2008). An intelligent PE-malware detection system based on association mining. *Journal in Computer Virology*, 4(4):323–334.