Poster: Static Malware Detection

Khanh Huu The Dam
LIPN, CNRS and University Sorbonne Paris Nord

Tayssir Touili
IRIF, CNRS and University Paris Cité

***Abstract*—Malware detection is nowadays a big challenge. The existing techniques for malware detection require a huge effort of engineering to manually extract the malicious behaviors. To avoid this tedious task of manually discovering malicious behaviors, we propose in this poster two approaches: (1) Apply Information Retrieval techniques to *automatically discover* malicious behaviors, and (2) Apply machine learning to *automatically learn* malwares. We use API call graphs to represent programs. API call graphs are graphs whose nodes are API functions and whose edges represent the order of execution of the different calls to the API functions (i.e., functions supported by the operating system). To *automatically* learn malwares, we apply well-known learning techniques based on Random Walk Graph Kernels (combined with Support Vector Machines). We achieve a high detection rate with only few false alarms (98.93% of detection rate with 0.73% of false alarms). As for the *automatic extraction* of malicious behaviors, we reduce this problem to the problem of retrieving from the benign and malicious API call graphs the set of subgraphs that are relevant for malicious behaviors. We solve this issue by applying and adapting well-known efficient Information Retrieval techniques based on the TFIDF scheme. We use our automatically extracted malicious behavior specification for malware detection using a kind of product between graphs. We obtained interesting experimental results, as we get 99.04% of detection rate. Using our two approaches, we were able to detect several malwares that well-known and widely used antiviruses such as Panda, Avira, Kaspersky, Avast, Qihoo360, McAfee, BitDefender, ESET-NOD32, F-Secure and Symantec could not detect.**

I. Introduction

The number of new malwares increased by 36 percent in one year from 2014 to 2015 according to the annual security threat report of Symantec. It is estimated that there are more than one million of new pieces of malwares released everyday. Thus, malware detection is a big challenge.

Most of the applications for malware detection are based on the signature matching technique. It consists on searching for patterns in the form of binary sequences (called signatures) in the program. Databases of malware signatures can be manually constructed by experts. A new observed program is declared as a virus if it contains a signature in the database. However, it is very easy for virus writers to get around this signature matching approach by obfuscating the binaries while keeping the same behaviors.

Another technique for malware detection is dynamic analysis, where the traces of a program are analysed while we run it in an emulated environment to look for a malicious behavior. However, it is hard to trigger the malicious behaviors as the running time is limited, since these may be hidden behind user interaction or require delays.

To sidestep the limitations of the above approaches, static analysis techniques were applied [1]–[4]. Indeed, this technique allows to analyse the behavior (not the syntax) of the program without executing it. However, in these works, discovring the malicious behaviors is done manually after a careful reading and analysis of the malwares' binary code. This task is tedious and necessitates an enourmeous amount of time and of engineering effort. Thus, one of the main challenges in malware detection is the automatic discovery of malicious behaviors. We tackle this problem in this poster. Following [5]–[7], we use API function calls to specify malicious behaviors. Indeed, it has been widely observed that malicious tasks are usually performed by calling sequences of API functions, since API functions allow to access the system and/or modify it.

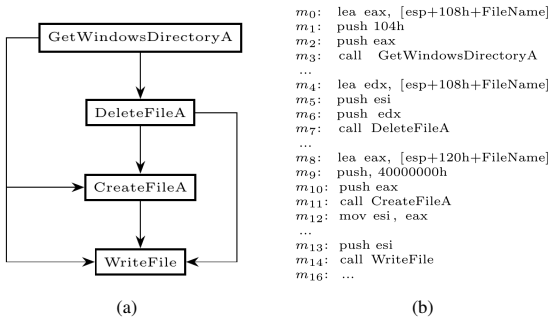


Fig. 1: Malicious API graph (a) and a fragment of the assembly code (b) of the malware Trojan-Dropper.Win32.Small.akd.

Let us consider a typical malicious behavior. Figure 1(b) is a fragment of the assembly code of the malware Trojan-Dropper.Win32.Small.akd. This malware drops a file and deletes a system file in Windows folder. First, the function `GetWindowsDirectoryA` is called. This allows the program to get the location of the Windows' directory. Then, the function `DeleteFileA` is called to delete a system file and the two functions `CreateFileA` and `WriteFile` are called to drop a file to this directory. This is a common malicious behavior that exists in several other malwares.

In order to represent this behavior we use a malicious API graph, which is a graph whose vertices are API functions, and whose edges (f, f') express that the API function f is called before the API function f'. Figure 1(a) represents the malicious API graph of this malicious behavior. E.g., the edges (`GetWindowsDirectoryA`, `DeleteFileA`), (`GetWindowsDirectoryA`, `CreateFileA`) and (`GetWindowsDirectoryA`, `WriteFile`) express that a call to the function `GetWindowsDirectoryA` is followed by the calls to the functions `DeleteFileA`, `CreateFileA` and `WriteFile`.

II. Automatic Extraction of Malicious Behaviors

Our goal is then to *automatically* extract such a malicious API graph. For that, we represent programs using API call graphs,

which are graphs whos vertices are pairs (m, f) consisting of an API function f and a control point m, and whose edges ((m, f),(m', f')) express that there is a call to the API function f at the control point m, followed by a call to the API function f' at the control point m'. Using such representations, checking whether a program is malicious or not can be done by performing a kind of product between the API call graph of the program and the malicious API graph. The program is declared as a malware if the product contains a feasible trace. It is declared as safe otherwise.

Then, given a set of API call graphs that correspond to malwares and a set of API call graphs corresponding to benign programs, we want to extract in a *completely automatic way* a malicious API graph that corresponds to the malicious behaviors of the malwares. This malicious API graph should represent the parts of the API call graphs of the malwares that correspond to the malicious behaviors. The best subgraphs that should be extracted are those able to distinguish the malicious API call graphs from the benign ones. Thus, our goal is to isolate the few relevant subgraphs from the nonrelevant ones. This problem can be seen as an Information Retrieval problem, where the goal is to retrieve relevant items and reject nonrelevant ones. The Information Retrieval community has been working on this problem for a long time. Over the past 35 years, it has accumulated a large amount of experience on how to efficiently retrieve information. Thus, it would be interesting to adapt the knowledge and experience of the Information Retrieval (IR) community to our malicious behavior extraction problem. This is the main goal of the first part of this poster. One of the most popular techniques that was shown to be very efficient in the IR community is the TFIDF scheme that computes the relevance of each item in the collection using the TFIDF weight that is computed from the occurrences of terms in a document and their appearances in other documents. We show in this poster how to adapt this approach that was mainly applied for text and image retrieval for malicious API graph extraction. For that, we associate to each node and each edge in the API call graphs of the programs of the collection a weight. Higher weight implies higher relevance. Then, we compute the malicious API graphs by taking edges and nodes that have the highest weights.

We implemented our techniques in a tool and obtained encouraging results: We first applied our tool to automatically extract a malicious API graph from a training set of malwares and benign programs. Then, we used this malicious API graph for malware detection on a test set of malwares and benign programs. We obtained a detection rate of 99.04%. Moreover, we used this malicious API graph to detect 180 newly generated malwares. We were able to detect all these malwares, whereas none of the well-known and widely used antiviruses such as Panda, Avira, Kaspersky, Avast, Qihoo-360, McAfee, AVG, BitDefender, ESET-NOD32, F-Secure, and Symantec were able to detect all of them.

III. Automatic Learning of Malicious Behaviors

Using the graph representation above, we apply, in the second part of this poster, machine learning techniques on graphs to learn malicious behaviors, and detect malwares. Support Vector Machine (SVM) is one of the most successful techniques in machine learning. It has been applied to several fields in pattern recognition including text analysis and bioinformatics. In this work, we apply Support Vector Machine based learning techniques for malware detection. The choice of Support Vector Machine is motivated by the fact that they are very suitable for nonvectorial data (graphs in our setting), whereas the other well-known learning techniques like artificial neural network, k-nearest neighbor, decision trees, etc. can only be applied to vectorial data. This SVM method is highly dependent on the choice of kernels. A kernel is a function which returns similarity between data. Standard kernels (including linear, polynomial, etc) handle vectorial data. However, for nonvectorial data such as graphs, these kernels become non suitable. That is the reason why we need to use specific kernels for graphs. In this work, we use a variant of the random walk graph kernel that measures graph similarity as the number of common paths of increasing lengths.

The second main contribution of this poster is the application of graph kernel based learning techniques for malware detection in a completely static way (no dynamic analysis). As far as we know, this is the first time that these techniques are applied for malware detection in a static manner.

We implemented our technique and tested it on a dataset of 6291 malwares, that are collected from Vx Heavens[1], and obtained encouraging results. Our tool can achieve a high detection rate with only few false alarms (98.93% for detection rate with 1.24% of false alarms).

Moreover, we show that our techniques are able to detect several malwares that could not be detected by well-known and widely used antiviruses such as Avira, Kaspersky, Avast, Qihoo-360, McAfee, AVG, BitDefender, ESET-NOD32, F-Secure, Symantec or Panda.

IV. Related Work

This poster is based on extensions of works presented in [8]–[10].

References

[1] J. Bergeron, M. Debbabi, M. Erhioui, and B. Ktari, "Static analysis of binary code to isolate malicious behaviors," in *WET ICE '99*, 1999.

[2] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," ser. SSYM'03, 2003.

[3] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Proactive detection of computer worms using model checking," *Dependable and Secure Computing*, 2010.

[4] F. Song and T. Touili, "Ltl model-checking for malware detection," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2013.

[5] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, "Synthesizing near-optimal malware specifications from suspicious behaviors," ser. SP '10, 2010.

[6] D. Babić, D. Reynaud, and D. Song, "Malware analysis with tree automata inference," ser. CAV'11, 2011.

[7] H. Macedo and T. Touili, "Mining malware specifications through static reachability analysis," in *ESORICS 2013*, 2013.

[8] K. Dam and T. Touili, "Extracting malicious behaviours," *Int. J. Inf. Comput. Secur.*, vol. 17, no. 3/4, pp. 365–404, 2022.

[9] ——, "STAMAD: a static malware detector," in *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019*. ACM, 2019, pp. 25:1–25:6.

[10] ——, "Precise extraction of malicious behaviors," in *2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC*, 2018, pp. 229–234.

[1] `http://vxheavens.org`

Static Malware Detection

Khanh Huu The DAM and Tayssir TOUILI

UNIVERSITE PARIS 13

USPC Université Sorbonne Paris Cité

Motivation

Symantec reported :

- —**317M** malwares in 2014 vs. **431M** malwares in 2015.
- —The number of malwares increased by **36%** in one year.
- —More than **1M** new malwares released everyday.

Malware Detection is a big challenge.

The Problem is ...

Extracting malicious behaviors requires a huge amount of engineering effort :

- —a tedious and manual study of the code.
- —a huge time for that study.

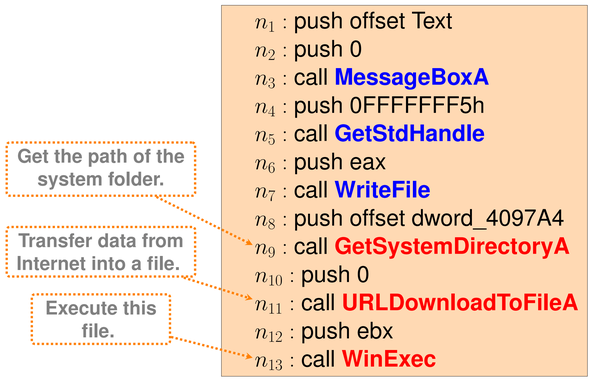
The main challenge is to avoid this manual work.

Our goal is to automatize this step

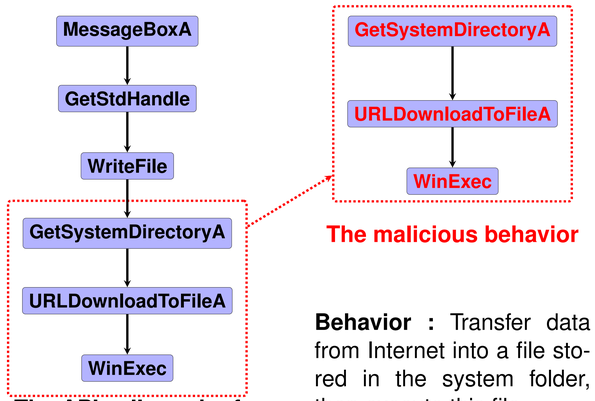
This can be done by the following two approaches :

1. Extract the malicious behaviors automatically.
2. Apply machine learning to detect malwares without extracting the malicious behaviors.

Modeling Malicious Behaviors



Fragment of Trojan Downloader.



The API call graph of the program.

The malicious behavior

Behavior : Transfer data from Internet into a file stored in the system folder, then execute this file.

An API call graph represents the order of execution of the different API functions in a program.

The program is represented by an API call graph.

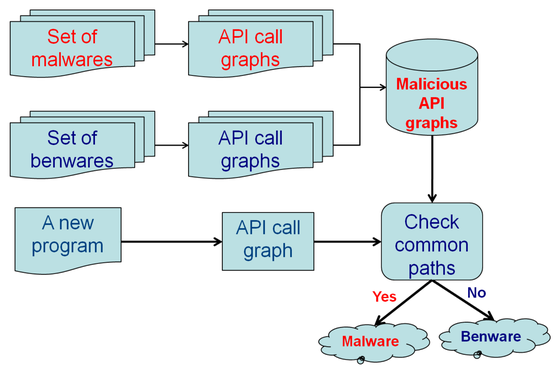
The malicious behaviors are represented by a malicious API graph.
that keeps the API functions needed to perform the malicious behaviors.

Approach 1 : Automatic Extraction of Malicious Behaviors

Our goal : Isolate the few relevant subgraphs (in malwares) from the nonrelevant ones (in benwares).

This is an Information Retrieval (IR) problem.

We adapt and apply the knowledge and experience of the IR community to our malicious behavior extraction problem.



Experiments

Apply on a dataset of 1980 benwares and 3980 malwares collected from Vx Heaven.

- —Training set consists of 1000 benwares and 2420 malwares → to extract malicious graphs.
- —Test set consists of 980 benwares and 1560 malwares → for evaluating malicious graphs.

We obtained the detection rate of 99.04% with 98.16% for the precision.

Comparison with well-known antiviruses

Detect new unknown malwares

- —180 new malwares generated by NGVCK, RCWG and VCL32 which are the best known virus generators (from VX Heaven).

Our approach achieves a detection rate of 100%.

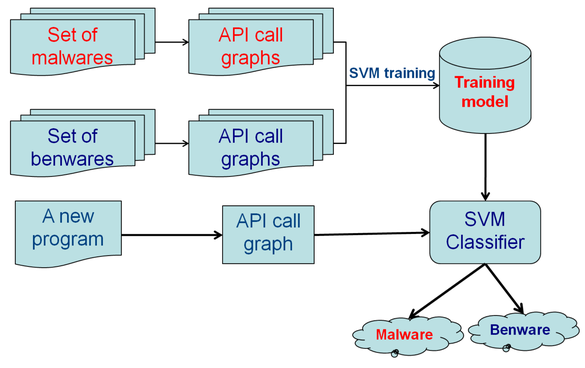
Antivirus	Detection Rates	Antivirus	Detection Rates
Our tool	**100%**	Panda	19%
Avira	16%	Kaspersky	81%
Avast	87%	Qihoo-360	96%
McAfee	96%	AVG	82%
BitDefender	87%	ESET-NOD32	87%
F-Secure	87%	Symantec	14%

Approach 2 : Learning Malicious Behaviors

Our goal : Implement machine learning for malware detection.

We need to use a learning technique for graphs.

Random Walk Graph Kernel based Support Vector Machines is the best learning technique that can be applied for API call graphs.



Experiments

We evaluate this technique on a dataset of 2323 benwares and 6291 malwares.

- —Training set of 2000 malwares and 2000 benwares → to compute the training model.
- —Test set of 4291 malwares and 323 benwares → for evaluating the performance of the training model.

We obtained the detection rates of 98.93% with 1.24% false alarms.

Comparison with well-known antiviruses

Detect new unknown malwares

- —180 new malwares generated by NGVCK, RCWG and VCL32 which are the best known virus generators (from VX Heaven).

Our approach achieves a detection rate of 100%.

Antivirus	Detection Rates	Antivirus	Detection Rates
Our tool	**100%**	Panda	19%
Avira	16%	Kaspersky	81%
Avast	87%	Qihoo-360	96%
McAfee	96%	AVG	82%
BitDefender	87%	ESET-NOD32	87%
F-Secure	87%	Symantec	14%