J Comput Virol Hack Tech (2013) 9:193–204
DOI 10.1007/s11416-013-0186-3

ORIGINAL PAPER

Abstracting minimal security-relevant behaviors for malware analysis

Ying Cao · Qiguang Miao · Jiachen Liu · Lin Gao

Received: 4 July 2012 / Accepted: 11 April 2013 / Published online: 26 April 2013

Abstract Dynamic behavior-based malware analysis and detection is considered to be one of the most promising ways to combat with the obfuscated and unknown malwares. To perform such analysis, behavioral feature abstraction plays a fundamental role, because how to specify program formally to a large extend determines what kind of algorithm can be used. In existing research, graph-based methods keep a dominant position in specifying malware behaviors. However, they restrict the detection algorithm to be chosen from graph mining algorithm. In this paper, we build a complete virtual environment to capture malware behaviors, especially that to stimulate network behaviors of a malware. Then, we study the problem of abstracting constant behavioral features from API call sequences and propose a minimal security-relevant behavior abstraction way, which absorbs the advantages of prevalent graph-based methods in behavior representation and has the following advantages: first API calls are aggregated by data dependence, therefore it is resistent to redundant data and is a kind of more constant feature. Second, API call arguments are also abstracted particularly, this further contributes to common and constant behavioral features of malware variants. Third, it is a moderate degree aggregation of a small group of API calls with a constructing criterion that centering on an independent operation on a sensitive resource. Fourth, it is very easy to embed the extracted behaviors in a high dimensional vector space, so that it can be processed by almost all of the prevalent statistical learning algorithms. We then evaluate these minimal security-relevant behaviors in three kinds of test, including similarity comparison, clustering and classification. The experimental results show that our method has a capacity in distinguishing malwares from different families and also from benign programs, and it is useful for many statistical learning algorithms.

Y. Cao
School of Computer Science and Technology, Xidian University, Mailbox 85, 2 South Taibai Road, Xi'an 710071, Shaanxi, China
e-mail: yingcao@stu.xidian.edu.cn

Q. Miao (✉)
School of Computer Science and Technology, Xidian University, Mailbox 167, 2 South Taibai Road, Xi'an 710071, Shaanxi, China
e-mail: qgmiao@126.com; qgmiao@mail.xidian.edu.cn

J. Liu
School of Computer Science and Technology, Xidian University, Mailbox 169, 2 South Taibai Road, Xi'an 710071, Shaanxi, China
e-mail: jcliu@stu.xidian.edu.cn

L. Gao
School of Computer Science and Technology, Xidian University, Mailbox 163, 2 South Taibai Road, Xi'an 710071, Shaanxi, China
e-mail: lgao@mail.xidian.edu.cn

1 Introduction

Malicious software, so called malware, is one of the most serious security threats facing the Internet today. A malware can disrupt computer system, collect sensitive information, steal bank accounts and passwords and so on, bringing about great loss to computer users. To combat with malwares, current security products employ a static pattern-matching approach which looks for context-based characteristic byte sequences. This method has efficiency in detecting known malwares, for they have high detection accuracy and a high speed than other heuristic ways. However, to evade detection, code obfuscation, such as polymorphism and metamorphism, is frequently used by malware writers, resulting in that relying solely on static signature is inefficiency for malware variants and has no use for unknown malwares [1]. These characteristics threaten the availability of classic security products. To compensate these deficiencies, dynamic

behavior-based analysis [2,3] is introduced with a motivation that using comparatively constant behavioral patterns to detect unknown and varietal malwares whose binary characteristics are ever-changing. Compared to that static analysis takes program source or assembly codes as analysis objects, dynamic behavior-based analysis concerns about run-time characteristics of a program. A standard procedure of such analysis approach usually consists of three parts: capturing program behaviors, abstracting behavioral features, and designing a behavior-based detection or classification algorithm. In this paper, we focus on the step of behavioral feature abstraction, which is an important foundation of behavior-based malware analysis.

In Windows system, API calls are interfaces between operating system and program, so they are the first choice to describe program behaviors. However, in most cases, unless behavioral feature abstraction is performed, API calls is not appropriate to be processed by algorithm directly. The reasons are as follows. First, any input data processed by the algorithm should be formalized. Second, the semantic granularity of API calls are too trivial, strictly speaking, they are activities, not behaviors. Third, It is ambiguous that one API call can complete multiple functionalities. Fourth, it is unavoidable that API calls collected while program is running contain a lot of redundant data, which may be caused by operating system management, API monitoring principle, or just loop calling. These redundant API calls could overwhelm core behaviors of a program. Fifth, API call sequence in time order is not a stable characteristic, they may change with different execution paths, API rearrangement or redundant API insertion. In conclusion, a way to abstract comparatively stable behavioral characteristics from original API calls is necessary.

In this paper, we design a generic way to abstract comparatively constant behavioral features from original API call sequence and embed it in a high-dimensional vector space, thus it can be processed by almost all of the prevalent statistical learning algorithms. This representation mainly has the following characteristics. (1) By absorbing the advantages of graph-based methods, data dependences of API calls are abstracted in a moderate aggregation granularity to form a minimal security-relevant behavior, which is an independent operation on a sensitive resource. (2) A careful API arguments abstraction strategy should also be designed to further reduce the dimensions of behavioral features, thus the extracted behavioral features are easy to be embedded in a high dimensional vector space.

2 Related work

Existing researches have proposed different methods to abstract behavioral features from original API call sequence, among them, the most popular ones are methods revolved around n-gram [4] and methods based on graph [5,6].

First introduced into malware analysis area for automatic signature generation [7], n-gram gradually evolves to a basic technique to extract characteristics of a malware [8], such as byte sequence n-gram [9], OpCode n-gram [10] and system call n-gram [11,12]. For all these methods, two parameters greatly affect the model: n which determines the length of the subsequence under analysis, and L which determines the number of n-gram. Usually, there is no universal principle to determine the optimal value of n and L directly. They are chosen through cross validation or other tests. When analyzing API call sequence, drawbacks related to this kind of method mainly are threefold, including loss of information, poor interpretability and *"curse of dimensionality"*. API call name only indicates the occurrence of an event. Its arguments contain ample information to judge whether the event is benign or malicious, thus is very important in malware analysis. However, n-gram only depicts temporal relations of a specific API call subsequence, and does not take their arguments into consideration, causing a great loss of information. In addition, one important aspiration of behavior-based malware analysis is to understand the intentions and functionalities of a program, namely, we expect that the analysis results are interpretable. However, n-grams of API calls take a statistical view of different invocation patterns between malwares and benign programs. Quite often, not every n-gram has exact behavior semantics. Last and most important, n-gram always confronts a dilemma between completeness of features and *"curse of dimensionality"*. In general, API call n-gram is not a stable characteristic to represent program behaviors. The instability violates the original intention of behavior-based malware analysis, which is focusing on comparatively constant features of malwares.

Like n-gram style methods, graph-based ones are also categorized into OpCode graph [13] and API call graph [5,6,14]. They overcome the above weaknesses to a large extent and keep a dominant position in API call sequence analysis, on which we focus in this paper. An output argument of an API call can be the input argument of another API call. This is a constant data dependence relationship and it builds up the core design idea of graph-based methods. A group of API calls aggregated together constitutes a real behavior, not a trivial activity. This is what is expected in behavior-based malware analysis. Based on this graphic specification, most significant graphs [15] or contrast subgraphs [5] are built from sets of malicious and benign programs to detect a malware. Furthermore, in some algorithms, the aggregation granularity can be customized to meet different type of demands. To detect or classify malwares algorithmically, a moderate degree aggregation will be more appropriate, but to generate analysis report for normal users, a high aggregation will be welcome. Because algorithms require mathematical,

informational and formalized input data to do statistical analysis, while normal users prefer a report easy to understand. A moderate degree aggregation offers more freedom for algorithm design and better reveals the statistical nature of data. By comparison, semantics expressing by highly aggregated API calls are very close to natural language in explaining purpose of a malware, which is a major concern of normal users.

Nevertheless, there are still challenges along the way of graph-based methods. In many cases, in order to reduce the complexity, only near-optimum solution is available. Besides, how to specify program behaviors formally determines what kind of detection algorithm can be used to a large extend. Considering great progress of statistical learning theory made in recent years, a graph specification restricts detection or classification algorithm to be chosen from graph mining algorithms, putting new learning models and techniques to no use, such as transfer learning, incremental learning, ensemble learning, boosting and so on. Actually all these new models, concepts or techniques potentially facilitate to combat with malwares. Some can help detect unknown malwares or malware variants, some can help select suitable discriminant features, and some can help improve the detection accuracy. To widen the range of choices in designing behavior-based malware detection or classification algorithm, a more generic representation of program behavior that can be processed by most of the prevalent machine learning algorithms is valuable.

3 Data collection

To capture behavior data of a program, mainly referring to API call sequence and OS state changes, automatically, transparently and accurately, many famous systems are put forward. These systems fall into two categories according to the distinguishing feature that at which layer the behavior monitoring is implemented.

In the first approach, behavior monitoring is implemented at the VMM (virtual machine monitor) layer. Through modifying the source codes of an open-source system emulator or virtual machine, a behavior monitor framework is inserted into the instruction virtualization process to collect behavior data. Attributing to the very fact that behavior monitoring is implemented at the most privileged layer in a virtual machine system, it is much harder to detect the existence of the monitoring system. Anubis [16] is the best example belonging to this type. After all its merits, semantic gap between operating system and VMM is the most troublesome difficulty encountered by this method. A VMM conscientiously virtualizes the whole computer system and is completely unconscious about process, which is a concept in operating system. However, it is a process running in the Guest OS that need to be monitored, thus, reconstructing processes information must be the first prerequisite. Unfortunately, it is not a so easy task. To a large extent, it depends on the design details of a specific operating system version. For windows, not everything is made public.

In the second approach, taking CFISandbox [17] as an example, behavior monitoring is implemented at the layer of Guest OS. In other words, virtual machine is taken just as a virtual execution environment, with other debugger or reverse engineering tools deployed in it to capture behavior data. Advantages of this method is that monitoring program behavior at the Guest OS layer makes it possible to directly utilize services or interfaces provided by the operating system. Besides, for they are obtained from a higher layer compared to the first approach, behavior data have better semantics to express high-level behaviors of a program. However, the disadvantages are also obvious. It has a higher risk in disturbing the normal execution of the program under analysis, thus it is easier to employ an anti-detecting technology for malwares.

In this paper, to capture API calls, we follow the core idea of Anubis, that is, modifying source code of the system emulator Qemu [18] to monitor program behaviors at the virtual machine monitor layer, in this way, full control of the computer system can be taken, which guarantees a more accurate behavior data.

Furthermore, to stimulate and then capture network behaviors of a malware, we also build a multi-virtual machine analysis environment, of which a core technology is network traffic redirection. From a security perspective, in malware analysis, any network traffic migrating from the virtual machine environment to Internet is undesired, for this will potentially make more computers in the local network and Internet infected. However, from a perspective of basic requirements of malware analysis, network behaviors of a program are bound to fail if no network environment provides, leading to a great loss of necessary information to evaluate whether the program is malicious or not. Thus, to analyze network behaviors of a malware, all kinds of online services in the Internet should be simulated within a constrained environment, including DNS, FTP, HTTP, Email service and so on. We realized network redirection with the help of MikroTik Router OS[1]. As illustrated in Fig. 1, a modified Qemu is taken as compromised computer in which behaviors of the analysis program are monitored. To speed up analysis process, more than one such Qemu can be distributed to form an analysis network to perform parallel analysis. In the analysis network, any network traffic from compromised computer is forwarded to ROS first, and then ROS will redirect all these network packages to another virtual machine, in which almost all of the common network services in the Internet are simulated.

[1] http://www.mikrotik.com/software.html.

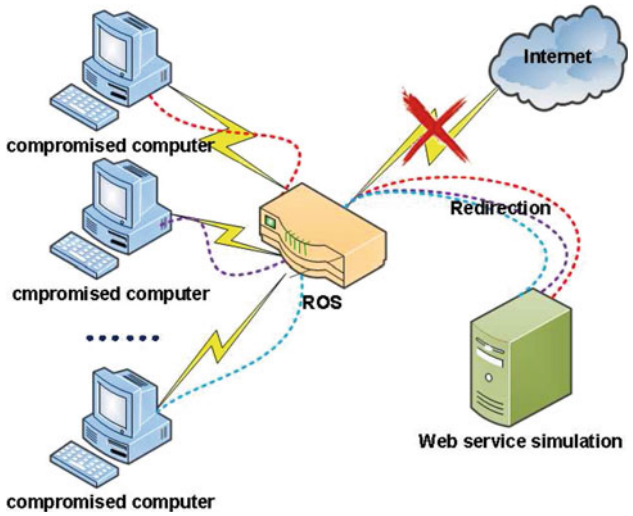

Fig. 1 System overview of data collection system in this paper

In addition, in the Internet services simulating virtual machine, instead of mere use of server softwares to build the Internet services, we also construct bogus responses to different type of service requests so that network behaviors of analysis program are stimulated to the maximum extend. The work mainly includes: (1) DNS hijacking: any DNS request will get a successful response, pointing to the IP address of the Internet service simulating virtual machine. (2) FTP downloading: if the program under analysis logs into any FTP server to upload or download files with a user name and password or anonymously, it will successfully get the very type file as it want, but of course it is a fake file. (3) Web page browsing redirection: if the program under analysis visits any web page, this visiting will be logged and redirected to a pre-prepared web page. (4) Email sending: request of sending an Email to any Email server with any user name and password will be redirected to the Email server set up ourself and successfully done, in addition to this, email content and attachment if any exists are also obtained and stored for further analysis. This is to address a situation that some malwares send out sensitive data they collected to their writers by sending an Email. (5) Port redirection: without being aware by analysis program, TCP and UDP packages sending to any IP address and any port will be redirected to a fixed and open port on the Internet service simulating virtual machine. This is an important step to stimulate network behaviors, because it is common for a backdoor or a trojan to connect to its client or server end in remote computer, but for malware analysis, quite often we can only get a server or a client end, hardly both, leading to the failure of establishing the initial connection. After network package and port redirection, despite control commands remain unknown, the first few TCP packages can be successfully sent out and we can analyze them, so network behaviors of a malware are greatly enriched.

4 Abstracting minimal security-relevant behaviors of a program

At the end of data collection step, not only API calls of main process corresponding to the analysis file are intercepted, but also API calls of child processes created by main process, injected processes if process injection happens, and service processes if any service is registered to operating system. These API calls cover file activities, registry activities, memory activities, process and thread activities, GUI activities, network activities and other security relevant activities, and make up the original data for behavior-based malware analysis. However, these API calls are too tedious for automatic analysis as analyzed in Sect. 1. Directly taking them into analysis consumes too much time and may not reach a satisfactory result, for there is a plethora of redundant data, leading to the most concerned API calls are paled in masses of trivial ones. To get a more constant and accurate representation of malware behaviors, we design a procedure called minimal security-relevant behavior abstraction to give a formal description of malware behaviors.

4.1 Criterion and representation

There are data dependences between API calls as mentioned in Sect. 2. This kind of dependence can be represented via a directed graph of a form that is adapted from and extends AND/OR graphs. Vertices in the graph are names of API calls, while edges are data dependences between them. This graphic model to specify malware behaviors is prevalent in malware analysis and more implementation details about it can be found in [5,6,14]. However, in this paper, we propose to abstract minimal security-relevant behaviors. Two key words of it are: security-relevant and minimal. We will explore these two points separately and go into the representation details in this section.

In windows system, thousands of API calls exit, as a consequence, monitoring all of them is impossible and unnecessary. Recall our objective is to detect malware samples, thus what concerns analyst most should be API calls that manipulate sensitive resources, such as file (including removable storage), registry, memory, network connection, security token, mutex or synchronization object, GUI component and so forth. Whichever kind of the resource an API call manipulates, the action mainly belongs to one of the following types: creation, opening, query, modification and deletion. We take these resource-centric behaviors as security-relevant. Moreover, program behaviors can be divided into different categories: file, registry, memory, process and thread, network, and GUI (not restricted to these). A group of API calls that manipulates on a resource and belongs to just one of these categories, we call it a minimal security-relevant behavior. For instance, it is common for a trojan or backdoor to release

a file and set it as an autostart program. This is a higher level behavior, not a minimal security relevant behavior, for it consists of a file behavior of creating a new file and a registry behavior of setting the registry value.

From a technique perspective, in windows system, a handle type argument marks a resource and can be shared across multiple API calls which as a whole constitute an independent operation on the resource. The same principle applied to some pointer type API call arguments. Besides these integer type data dependence, another frequently encountered data dependence is built on string type API arguments. Take executing the released file as an example, such behavior is obtained by establishing a data dependence on a string type argument: file name. This is a high-level program behavior. In this paper, we define that integer type data dependence is help to build minimal security-relevant behaviors while string type data dependence is help to build high level behavior.

The representation of minimal security-relevant behavior extends from [14] and some changes are made to facilitate abstraction algorithm design. In our representation, each graph is made up of four different types of nodes: start node, internal node, stop node and submitting node, of which, only the internal node and stop node are real API calls, others are analyst defined events to guide abstraction algorithm to take certain actions rather than to match to an API call. The necessity of such virtual nodes is that they help formulate the representation and make the graphs easier to be stored in a linear table. Figure 2 gives an example of modification of a registry key in our specification language. The start node indicates initializing a matching template and is not a real API call. The internal node represents a security-relevant API call needed to be analyzed. The stop node represents a status that the execution of a minimal security-relevant behavior fails half-way. This may caused by program crash or for a certain special purpose, we record such failure, and meanwhile, matching an edge from the penultimate node to submitting node is no more need, thus is ceased. Once a submitting node is reached, it means a minimal security-relevant behavior is successfully detected, and then an output event is generated.

4.2 Abstraction algorithm

This section gives design details of the minimal security-relevant behavior abstraction algorithm with emphasis on a basic data structure we used, for it is a fundamental part. The abstraction algorithm can be generalized as bounding API calls to pre-defined behavior graphs. In the bounding process, a match unit does not only involve a single API call, but names of two calls and dependence between them, which is a unique matching pattern determined by practical meanings of the behavior graph.

In the abstraction process, collected API calls are partitioned into three sets. The first one is the active API call set, in which are already matched API calls. The second one is the to-be-matched API call set. Given an already matched API call, its successors can be deduced by the edge constrain. All the successors of the already matched API calls make up the to-be-matched API call set. The last one is the discarded API call set, API calls in which are unhelpful for analysis. Based on the above analysis, we realize the algorithm with a help of an auxiliary two dimensional linear table, which keeps trace of adjustments of these three sets. It can be said simplistically that the algorithm is a three-part process: part initialization the table, part looking up the table and part updating and clearing up the table.

Let $X\{x_1, x_2, \ldots, x_k\}$ be the set of raw API calls, A be the set of active API calls, P be the set of to-be-matched API calls, $\{G_1, G_2, \ldots, G_n\}$ be the set of pre-defined minimal security-relevant behavior graphs and M is the output of the abstraction algorithm, in which are minimal security-relevant behaviors. A dependence is a logic formula expressing the conditions placed on two API calls, which is annotated as $\gamma(x, y)$. Because P is a set deduced by A, for any $p_k \in P$, we keep a record of which element in A deduced p_k and which graph in G they belong to. Thus operator $\tau(p)$ returns the element in A that deduces P, and operator $gid(x)$ returns the index number of minimal security-relevant behavior graph to which x belongs. Operator $pre_i(x)$ and $suc_i(x)$ fetches all predecessors and successors of x in G_i. Operator $rel(X)$ checks the relationship among elements in set X and its return value may be "OR" or "AND". Operator $bev(G_i)$ returns the minimal security-relevant behavior depicted by G_i, consisting by a string-described behavior name and a string-described parameter. Pseudo-code for the algorithm is sketched in Table 1.

Challenges in this algorithm lie in two aspects. Since different behaviors may be processed at intervals, half-matched graph cannot be discarded until all API calls are handled. It means that before one graph is thoroughly matched, another API call may appear to interfere with the current bounding process, which will lead to bounding API calls to a new graph. Moreover, some graphs share a same API call sequence, there are chances that half-matched graphs actually indicates a behavior which does not exist in the analyzed program. These call for well-designed memory management and conflict eliminating policies, which we solve technically.

4.3 Abstracting API call arguments

In Sect. 4.2 we discuss aggregating API calls together by data dependences, this helps detect the occurrence of a security-relevant behavior. However, whether this behavior is malicious or not depends on its argument, leading to that the mere focus on data dependences is not enough. Nevertheless arguments of API calls play an important role in malware analysis. The problem of abstracting them is not made clear

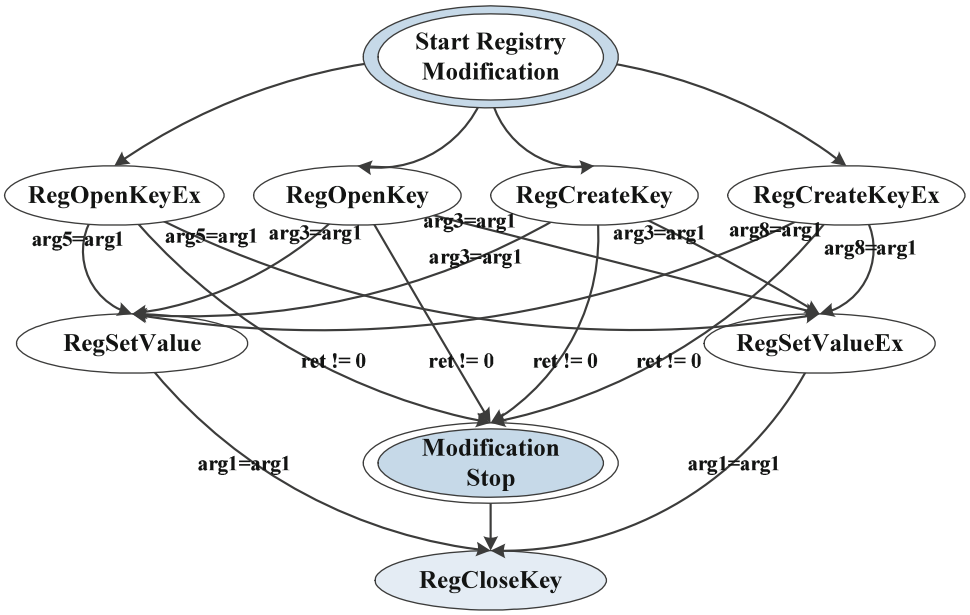

Fig. 2 An example of minimal security-relevant behavior: modifying a registry key

Table 1 Algorithm to abstract minimal security-relevant behaviors

Input: A set of collected API call set $X\{x_1, x_2, \ldots, x_n\}$
Output: A set of minimal sevurity relevant behaviors $M\ \{\ldots, m_j, \ldots\}$
begin
  for i=1 to N do
    $A \leftarrow$ start node of G_i
    $P \leftarrow \text{suc}_i(\text{start node of } G_i)$
  $M \leftarrow \varnothing$
  foreach $j = 1$ to N do
    foreach($x_j \in P$)
    /*check the dependence between x_j and $\tau(x_j)$*/
    if ($\gamma(\tau(x_j), x_j)$ is compromised)
      $\text{Pre} \leftarrow \text{pre}_{gid(x_j)}(x_j)$
      /*check the relationship between x_j and its predecessors*/
      if(rel(Pre)==OR)
        if($\exists m \wedge p_m \in$ Pre)
          $A \leftarrow x_j, P \leftarrow \text{suc}_{gid(x_j)}(x_j)$
          delete $\text{pre}_{gid\ x_j}$ from A
      if(rel(Pre)==AND)
        if($\forall m \wedge p_m \in$ Pre)
          $A \leftarrow x_j, P \leftarrow \text{suc}_{gid(x_j)}(x_j)$
          delete $\text{pre}_{gid\ x_j}$ from A
    if (TYPE($\text{suc}_{gid(x_j)}(x_j)$)==submit node)
      $M \leftarrow bev(G_{gid(x_j)})$
return $M\ \{\ldots, m_j, \ldots\}$

in many graph-based methods. In this paper, we work on this topic explicitly. Some related works can be found in [15].

API call argument abstraction is performed on two types of API call arguments: strings and pre-defined Windows constants. For strings, various strings of different literal values but having a same meaning, functionality or nature are grouped into one abstract description, and we call it string family. For example, there are dozens of registry locations to set a program start automatically once the computer boosts [19]. We translate all these literally different registry locations into a common one "registry location of autostart" to represent the characteristic that they share. For pre-defined constants, in behavior monitoring, we barely get their values which are ambiguous to understand, decoding them back to the meanings they actually represent is more valuable. For example, when the second argument of SetFileAttributes is 0x00000002, it will be translated into "FILE_ATTRIBUTE_HIDDEN", indicating that the file is hidden and are not included in a typical directory listing. Tables 2 and 3 list a small part of abstraction rules for string type arguments and pre-defined constants to illustrate argument abstraction process.

The role of argument abstraction is twofold: more constant behavior features and to avoid excessive growth of dimension of the feature space. Recall the *n*-gram representation suffers from a limitations of *"curse of dimensionality"*, for it depends on too trivial information, resulting in the feature number is always too large, in worse cases, it may be unacceptable. To some extend, arguments abstraction avoids the unlimited expansion of feature dimension.

5 Experiment and evaluation

In this paper, we propose abstracting minimal security-relevant behavior to do malware analysis. To determine the validity of our method, the following two main questions should be answered.

1. Do minimal security-relevant behaviors have a capability of specifying discriminant behavioral features of malwares from different families and benign programs?

Table 2 Abstraction rules for some string type arguments

Originial string value	After abstraction
C:\Documents and Settings\Administrator\Application Data\Tencent;	Directory of instant messaging tool
C:\Program Files\Tencent;	
C:\Program Files\AliWangWang;	
C:\Program Files\Messenger;…	
C:\Documents and Settings\Administrator\Cookies;	Registry location of Internet temporary files
C:\Documents and Settings\Administrator\Local Settings\Temp\Cookies;	
C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files; …	
SVCH0ST.exe, kernel32.exe, Rundll.exe, Expiorer.exe, Windows.exe, …	Process names that intentionally obfuscated with normal program
**.inf, *.ini,…	Configuration file
**.bat, *.js, *.vbs, *.scp, *.jsp, …	Script file

Table 3 Abstraction rules for some pre-defined constants in Windows system

Rule name	Pre-defined value	Description
File Attributes	0x00000001	FILE_ATTRIBUTE_READONLY
	0x00000002	FILE_ATTRIBUTE_HIDDEN
	…	…
Registry Key	0x80000000	HKEY_CLASSES_ROOT
	0x80000001	HKEY_CURRENT_USER
	0x80000002	HKEY_LOCAL_MACHINE
	…	…
Socket Type	0x00000001	SOCK_STREAM
	0x00000002	SOCK_DGRAM
	…	…
Socket Protocol	0x00000000	IPPROTO_NOT_SPECIFY
	0x00000006	IPPROTO_TCP
	…	…

2. Are minimal security-relevant behaviors valuable for detecting and classifying varietal and unknown malwares when using the prevalent statistical learning algorithms?

To answer these questions, we perform three kinds of experiment including similarity comparison, clustering and classification, since they are the most concerned applications in behavior-based malware analysis.

5.1 Experimental setup

This section first describes the experimental setup used in our study. To construct minimal security-relevant behaviors, the first and most basic step has bearing on API calls. In Windows system there are thousands of API calls and monitoring all of them is impossible. As a result, only a small subset could be chosen. Once a program escapes monitoring by invoking API calls beyond the range of surveillance, it cuts the ground on which API call sequence analysis style methods base. Therefore, the choices of monitored API calls play an important role in behavior-based malware analysis.

In this paper, we count API calls from imported table of PE format file of 3545 malwares and 208 benign programs. 251 most frequent ones are chosen. By adding possible data dependences of these API calls, 90 minimal security-relevant behavior graphs which function as rules to guide the abstraction algorithm in processing individual API calls are constructed. Besides, argument abstraction rules with 44 categories for string type arguments, 6 categories for pre-defined Windows constants are also defined. All together, 443 minimal security-relevant behaviors are abstracted. The detail statistical information is listed in Table 4.

To answer these two questions, we collected 1018 malwares from nine families and 293 benign programs from a fresh installed Microsoft Windows XP sp3. Composition information of the test dataset can be found in Table 5 and it is used in all the following experiments.

Moreover, we compare our method against a standard n-gram model. For the standard n-gram model, top 500 2-grams are chosen by computing the information gain.

5.2 Similarity comparison

Similarity comparison plays an key role in malware analysis and detection, for it is an important foundation for clustering, detection and classification [20]. A good specification language to depict malware behaviors should present high similarity inner a same class and low similarity between different classes. With this in mind, we perform similarity comparison for our method and standard n-gram to determine the ability of minimal security-relevant behavior in depicting discriminant behavioral features of malwares and benign

Table 4 Statistical information of minimal security-relevant behaviors that constructed in our experiments

Category	Number of monitored API calls	Number of minimal security-related behavior graphs	Number of minimal security-relevant behaviors
File	65	24	301
Registry	34	6	51
Process and thread	32	18	49
Service	14	3	14
GUI	12	6	3
Network	56	14	6
Others	38	19	19
Total	251	90	443

Table 5 Test dataset

Malware family	Number
Trojan-Spy.Win32.Carberp	52
IM-Flooder.Win32	66
Backdoor.Win32.Kbot	86
Trojan-Spy.Win32.Lydra	174
Net-Worm.Win32.Kolabc	138
Net-Worm.Win32.Aspxor	74
Trojan-Spy.Win32.Ardamax	118
HackTool.Win32.QQMima	126
Trojan-PSW.Win32.Dybalom	184
Benign Program	293

programs. The method we used to compute similarity of n-gram is from [12] and Euclidean distance is taken as similarity measurement in both models.

In this experiment, similarities of 1018 malwares and 293 benign programs are counted pointwise. The results then form a similarity matrix and we use heap map to visualize it. Each point along X and Y axis represents a sample file. A point with coordinate (x, y) in the graph indicates similarity between the xth and the yth test sample. A special case is that (x, x) indicates similarity of a sample compared to itself. The result is always 1 after normalization. By considering that the lighter the point, the higher similarity are shared between the two samples, the line across the main diagonal is always white. We permute test samples from a same class one by one together, so that squares along main diagonal represent similarities inner a same class. Squares outside main diagonal represent similarities between different classes. Figure 3 is the result of similarity comparison for minimal security-relevant behavior specification (left) and 2-gram method (right).

From Fig. 3 we can see that, in minimal security-relevant behavior specification, squares along the diagonal are much lighter than squares outside the diagonal. There are clearer boundaries between squares with comparison to 2-gram specification. This indicates higher similarity inner a same class and lower similarity between different classes and our methods has a capability of depicting discriminant behaviors of malwares and benign programs.

5.3 Clustering

Clustering is an important way to evaluate the validity of behavioral feature abstraction. A good clustering result that samples from a same class are clustered into one group means distinguishing features of different classes are abstracted. Therefore after similarity comparison, we perform clustering on minimal security relevant behaviors to further test whether our method is useful in this application. Likewise, we compare our method with 2-gram. For clustering algorithm, we choose K-means and use Euclidean distance as similarity measurement. For the target cluster number, we set it the same as class number of the training samples, namely ten.

After clustering, a matrix can be used to show the composition of each cluster. Its columns are the obtained clusters, and rows are sample families. In the ideal situation, after clustering, samples from one class are still in a same cluster. This means high similarity is shared inner class and low similarity is shared between different classes. Numbers of this kind of samples are just elements along main diagonal of the matrix. To visualize, we normalize the matrix by row so that the (x, y) element in the matrix indicates the rate that samples of xth category are distributed in the yth cluster. Then heat map of this matrix is showed in Fig. 4. The darker the squares on diagonal and meanwhile the lighter the squares outside diagonal, the better the way to depict discriminant features.

In Fig. 4, we can see that, samples from a same class can be successfully clustered together in minimal security-relevant behavior representation. It means that behavioral features extracted by minimal security-relevant behavior method have a capacity in telling malwares from benign programs, and

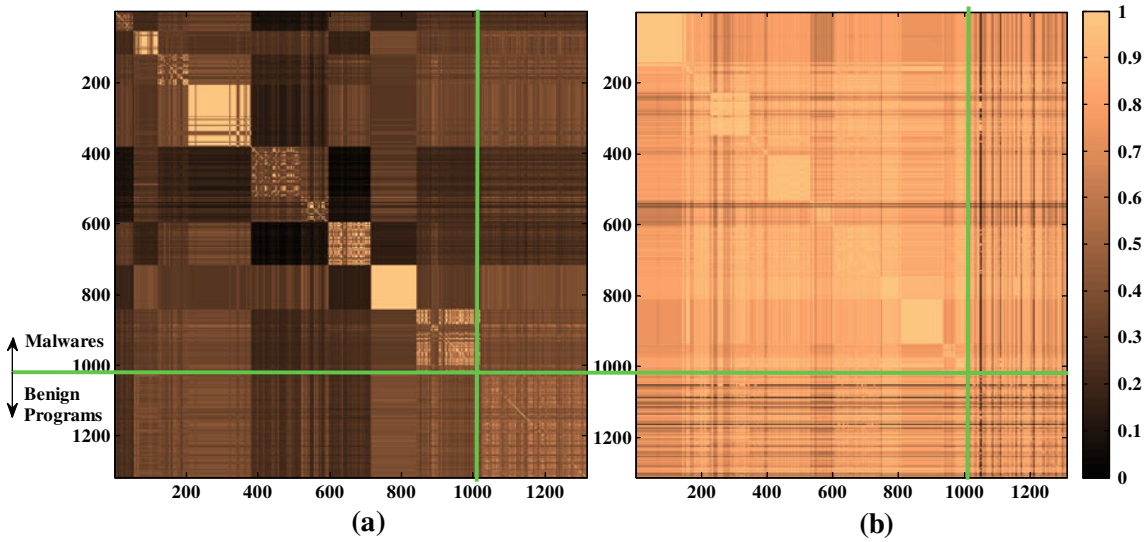

Fig. 3 The heat maps of the similarity matrices for benign programs versus malwares. **a** Minimal security-relevant behavior. **b** 2-gram

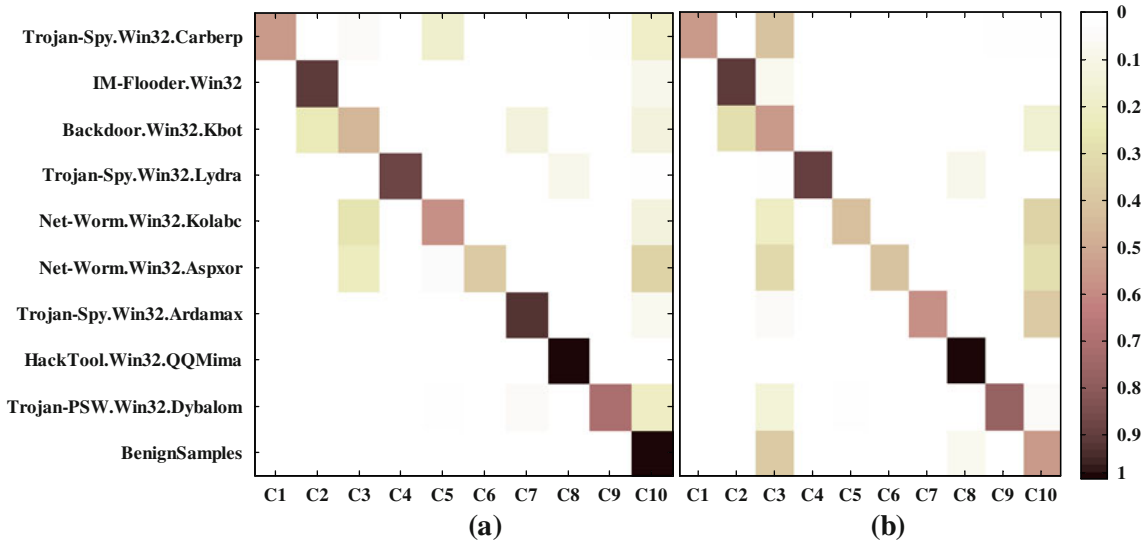

Fig. 4 The heat maps of clustering results. **a** Minimal security-relevant behavior. **b** 2-gram

telling malwares from different families. In particular, all the benign programs are clustered together in our method with comparison to 2-gram method.

5.4 Classification

To answer the second question, we further perform different classification algorithms on minimal security-relevant behavior specification.

First of all, direct visualization of the sample distributions is tested in Fig. 5, because almost all of the statistical learning algorithms are based on a hypothesis that training and testing samples are generated from a fixed but unknown distribution. If malware samples under minimal security-relevant behaviors do not present a fixed distribution, or distributions of different malware families and benign programs are heavily overlapped, no algorithm could give a good classification result. However, because only spaces of three dimensions or fewer can be directly visualized, PCA analysis is performed at first on both minimal security-relevant behavior specification (443 dimensions) and 2-gram specification (500 dimensions) to reduce feature dimensions. From intuitionistic vision effects, it can be seen that in our methods, samples from different classes present better distributions. For in the area marked by the green circle, sample distributions overlap heavily, this will cause difficulty for the following detection or classification algorithm.

In the classification test, all the algorithms used are taken from Weka [21], and we use two ways to calculate the testing classification accuracy, FP rate and AUC. The

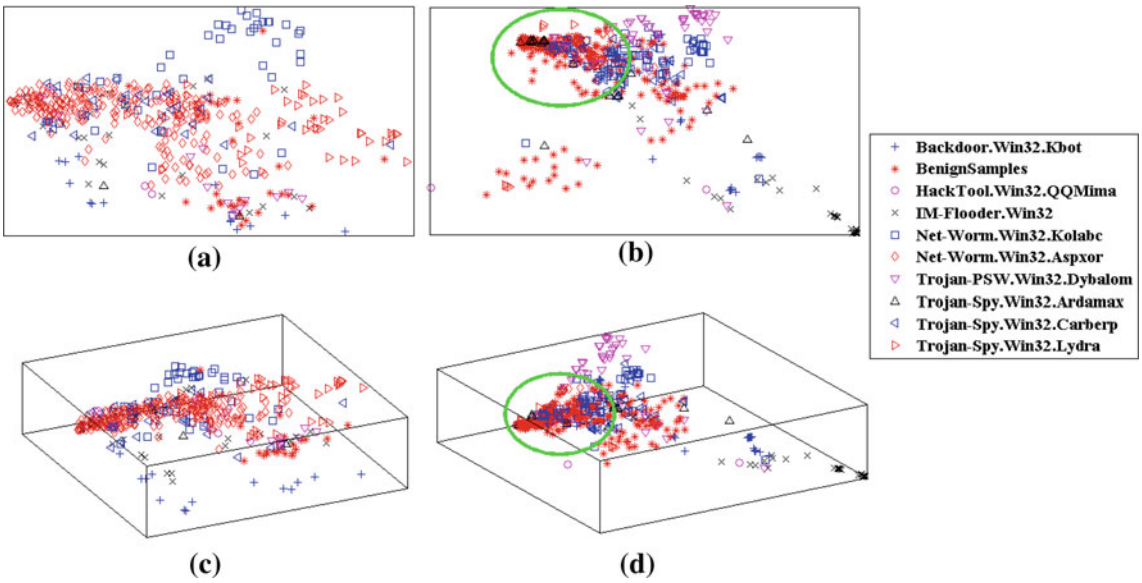

Fig. 5 Distributions of 9 malware families and benign programs after PCA analysis. **a** Sample distribution in 2D under minimal security-relevant behavior specification. **b** Sample distribution in 2D under 2-gram specification. **c** Sample distribution in 3D under minimal security-relevant behavior specification. **d** Sample distribution in 3D under 2-gram specification

Table 6 Classification results for minimal security-relevant specification and 2-gram specification

Learning algorithm	Our method			API 2-Gram		
	Classification accuracy (%)	FP (%)	AUC	Classification accuracy (%)	FP (%)	AUC
Naive Bayes (CV)	86.25	3.06	0.962	**86.65**	4.09	**0.977**
Naive Bayes (Splitting)	88.21	3.05	0.973	84.92	7.69	0.965
Bayes Network (CV)	86.78	3.74	0.964	**87.11**	**3.41**	**0.979**
Bayes Network (Spliting)	87.64	3.05	0.976	85.69	9.38	0.97
SMO (CV)	94.00	4.08	0.981	93.70	4.42	0.976
SMO (Splitting)	94.68	3.94	0.985	93.32	**3.28**	0.98
J48 (CV)	92.63	7.48	0.962	92.45	**6.55**	0.956
J48 (Split)	91.83	5.34	0.945	88.93	7.43	0.919
Random Forest (CV)	94.60	1.70	0.998	93.90	2.39	0.996
Random Forest (Splitting)	94.49	0	0.992	92.75	1.69	**0.994**
Bagging + J48 (CV)	94.17	5.12	0.986	93.59	6.12	0.984
Bagging + J48 (Split)	92.78	4.58	0.993	91.41	5.98	0.986
AdaBoost + J48 (CV)	93.38	6.85	0.994	93.06	6.85	0.992
AdaBoost + J48 (Splitting)	93.35	7.63	0.995	91.60	9.02	**0.976**

first one is ten-fold cross validation which is frequently used to evaluate the generalization ability of a learning algorithm. The second one is that we split test samples in Table 5 with a proportion of 6 to 4, so that in every class, 60 % of the samples are used for training and the left 40 % are used for testing. This process is repeated 10 times and the average results are recorded. The test results are listed in Table 5. For overall classification accuracy and AUC, the larger the value, the better the performance is. For FP rate, the situation goes opposite. We highlight the results that 2-gram are better than our method in bold.

From Table 6 we can see, in most cases, the overall classification accuracy and FP rate are better under minimal security-relevant behavior specification. Only in one experiment (Bayes Network with ten-fold cross validation), three evaluations of 2-gram are all better than our method. Moreover, we can also find that AdaBoost is not comparable to

random forest, and is almost equal to Bagging. AdaBoost boosted decision trees is known as the best "off-the-shelf" learning algorithm. However, the original AdaBoost algorithm has a property that it is sensitive to noisy data. By contrast, random forest is an algorithm with better anti-noise property. In the case that there are less noisy data, AdaBoost will always have a higher classification rate than Bagging, for it is specially designed to reduce the classification error. Once random forest and bagging have a close or even better classification error, this means the input data contains noisy data. In behavior-based analysis, because of execution error, hidden behaviors, lack of necessary requirements for execution, or other reasons, behavior data always contain noisy data. To get a better detection or classification performance, the following behavior-based algorithm should have an anti-noise ability.

6 Conclusions and future work

In this paper, we build a complete virtual environment to capture malware behaviors and study the problem of extracting constant behavioral features from API call sequences. We propose a minimal security-relevant behavior abstraction algorithm and then experimentally evaluate this method in three kinds of test: similarity comparison, clustering and classification. The experiment shows that operations on sensitive resources can be used as discriminant behavioral features to detect malwares from benign programs. Such an operation is obtained by a moderate degree aggregation of a small group of API calls, thus is easy to implement. Let m be the number of pre-defined behavior graphs and n be the number of API calls to be analyzed. The complexity of our abstracting algorithm is $O(mn)$, therefore it is easy to implement. Besides, the abstracted behavioral features are easy to be embedded in a high dimensional vector space, so that it can be directly used by almost all of the prevalent statistical learning algorithms.

At meanwhile, there are still some work we plan to do in the following research. In current work, from API calls to minimal security-relevant behaviors, three links require human involvement, including choosing monitored API calls, building rules to abstract minimal-security relevant behaviors and API call arguments. Any work with manual intervention is at a risk of bias, which can be exploited by a malware to escape detection. This problem is addressed as functional polymorphic in [22] and will be confronted by many behavior-based malware detection methods. In future work, by doing statistical analysis and mining executions of malware samples, we plan to extend our work to automate these human works. The automatic process also helps improve the practical use of our method. Besides, we plan to pay more concerns on mining distinguishing features of different malware classes according to their functionalities and families in future work. This is a crucial factor for accurate multi-classification of malwares, which is more difficult than binary classification. We also have great interests in utilizing different statistical learning models such as boosting, incremental learning, transfer learning and so on to better solve the invariable problem of unknown and varietal malware detection.

Acknowledgments The work was jointly supported by the National Natural Science Foundations of China under grant No. 61072109, 61272280, 41271447, 61272195, the Program for New Century Excellent Talents in University (NCET-12-0919), the Fundamental Research Funds for the Central Universities under grant No. K5051203020 and K5051203001.

References

1. Filiol, E.: Malware pattern scanning schemes secure against black-box analysis. J. Comput. Virol. **2**(1), 35–50 (2006)
2. Jacob, G., Debar, H., Filiol, E.: Behavioral detection of malware: from a survey towards an established taxonomy. J. Comput. Virol. **4**(3), 251–266 (2008)
3. Filiol, E., Jacob, G., Le Liard, M.: Evaluation methodology and theoretical model for antiviral behavioural detection strategies. J. Comput. Virol. **3**(1), 23–37 (2007)
4. Shabtai, A., Moskovitch, R., Elovici, Y., Glezer, C.: Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey. Inf. Secur. Tech. Rep. **14**(1), 16–29 (2009)
5. Christodorescu, M., Jha, S., Kruegel, C.: Mining specifications of malicious behavior. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Softerware Engineering (ESEC/FSE '07), Cavat, Croatia, pp. 5–14. ACM, New York, USA (2008)
6. Kolbitsch, C., Comparetti, P.M., Kruegel, C., Kirda, E., Zhou, X., Wang, X.F.: Effective and efficient malware detection at the end host. In: Proceedings of the 18th conference on USENIX security symposium (USENIX Security'09), pp. 351–366. USENIX Association, Springer, Heidelberg (2009)
7. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., White, S.R., Watson, T.J.: Biologically inspired defenses against computer viruses. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), Quebec, Canada, pp. 985–996. Lawrence Erlbaum Associates LTD (1995)
8. Reddy, D.K.S., Pujari, A.K.: N-gram analysis for computer virus detection. J. Comput. Virol. **2**(3), 231–239 (2006)
9. Reddy, K. S., Dash, S. K., Pujari, A. K.: New malicious code detection using variable length n-grams. In: International Conference on Information Systems Security (ICISS), Lecture Notes in Computer Science, vol. 4332, pp. 276–288 (2006)
10. Santos, I., Brezo, F., Sanz, B., Laorden, C., Bringas, P.G.: Using opcode sequences in single-class learning to detect unknown malware. IET Inf. Secur. **5**(4), 220–227 (2011)
11. Ravi, C., Manoharan, R.: Malware detection using Windows API sequence and machine learning. Int. J. Comput. Appl. **43**(17), 12–16 (2012)
12. Rieck, K., Trinius, P., Willems, C., Holz, T.: Automatic analysis of malware behavior using machine learning. J. Comput. Secur. **19**(4), 639–668 (2011)

13. Anderson, B., Quist, D., Neil, J., Storlie, C., Lane, T.: Graph-based malware detection using dynamic analysis. J. Comput. Virol. **7**(4), 247–258 (2011)
14. Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., Mitchell, J.: A layered architecture for detecting malicious behaviors. In: Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08), Cambridge, MA, USA, pp. 78–97. Springer, Berlin, Germany (2008)
15. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near-optimal malware specifications from suspicious behaviors. In: Proceedings of the 31st IEEE Symposium on Security and Privacy, Berkeley, CA, pp. 45–60. IEEE, New York, USA (2010)
16. Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: a tool for analyzing malware. In: Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference, Hamburg, Germany (2006)
17. Willems, C., Holz, T., Freiling, F.: Toward automated dynamic malware analysis using cwsandbox. IEEE Secur. Priv. **5**(2), 32–39 (2007)
18. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the USENIX 2005 Annual Technical Conference, California, USA, pp. 41–46. USENIX Associations, Springer, Heidelberg (2005)
19. Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., Kruegel, C.: Insights into current malware behavior. In: Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET 2009), Boston, USA USENIX Associations, Springer, Heidelberg (2009)
20. Apel, M., Bockermann, C., Meier, M.: Measuring similarity of malware behavior. In: Proceedings of the 34th Conference on Local Computer Networks (LCN'09), Zrich, Switzerland, pp. 891–898. IEEE, New York, USA (2009)
21. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explor. Newsl. **11**(1), 10–18 (2009)
22. Jacob, G., Filiol, E., Debar, H.: Functional polymorphic engines: formalisation, implementation and use cases. J. Comput. Virol. **5**(3), 247–261 (2009)