SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1148

RESEARCH ARTICLE

Three-phase behavior-based detection and classification of known and unknown malware

Ying-Dar Lin[1], Yuan-Cheng Lai[2]*, Chun-Nan Lu[1], Peng-Kai Hsu[1] and Chia-Yin Lee[3]

[1] Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan
[2] Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan
[3] Information & Communication Technology Laboratories, National Chiao Tung University, Hsinchu 300, Taiwan

ABSTRACT

To improve both accuracy and efficiency in detecting known and even unknown malware, we propose a three-phase behavior-based malware detection and classification approach, with a faster detector in the first phase to filter most samples, a slower detector in the second phase to observe remaining ambiguous samples, and then a classifier in the third phase to recognize their malware type. The faster detector executes programs in a sandbox to extract representative behaviors fed into a trained artificial neural network to evaluate their maliciousness, whereas the slower detector extracts and matches the LCSs of system call sequences fed into a trained Bayesian model to calculate their maliciousness. In the third phase, we define malware behavior vectors and calculate the cosine similarity to classify the malware. The experimental results show that the hybrid two-phase detection scheme outperforms the one-phase schemes and achieves 3.6% in false negative and 6.8% in false positive. The third-phase classifier also distinguishes the known-type malware with an accuracy of 85.8%. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS
malware detection; malware classification; behavior analysis; sandbox; system call

***Correspondence**
Yuan-Cheng Lai, Department of Information Management, National Taiwan University of Science and Technology, Taipei 106, Taiwan.
E-mail: laiyc@cs.ntust.edu.tw

1. INTRODUCTION

Many solutions have been proposed for preventing malware infection. Existing anti-malware solutions can be categorized into two types, that is, signature-based scanning, which has light computation overhead, and behavior-based analysis, which has heavy computation overhead. Signature-based solutions extract the unique string of a malware binary to construct a database of signatures. These solutions can then check whether a program is malicious or not by matching the binary of the file with the malware signatures. Although signature-based solutions are both efficient and effective in recognizing known malware, they fail to identify *unknown* malware. To overcome this drawback, behavior-based solutions execute malware to monitor and analyze their runtime behaviors, such as network access and memory modifications. Because most malware share some *common* behaviors, we could use such behaviors to judge whether a suspicious file is malicious or not.

Malware behaviors can be observed from two different perspectives: host behaviors and network behaviors. Host behaviors refer to all the activities that might change attributes of the system, such as modifying the registry files of the operating system (OS). On the other hand, network behaviors usually make connections to remote victims or servers, such as command and control servers. Figure 1 illustrates the relationship among different behaviors. Benign programs have only benign behaviors (BB), whereas malware exhibits suspicious behaviors (SB), which contain both benign and malicious behaviors (MB). Malware might make use of BB to shelter its malicious ones. Most malware has malicious host behaviors; however, only some kinds of malware have additional malicious network behaviors, that is, *intrusive* behaviors (IB), which influence remote victim hosts, where $IB \subseteq MB$. If we can recognize such behaviors, we could detect and classify *unknown* or *zero-day* malware more precisely.

Many behavior-based anti-malware solutions investigate malware behaviors by internal observation [1–7] or external observation [8,9]. For internal observation, it can trace the program processes by interrupting the execution or checking the registers to record certain footprints such as system calls. Forrest *et al*. [1], Mutz *et al*. [2], and

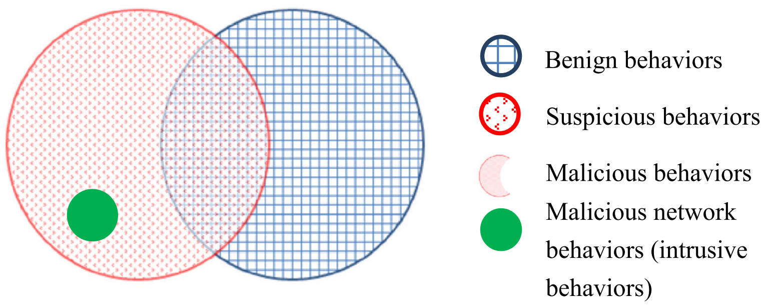


Figure 1. Relationship between different behaviors.

Warrender *et al.* [3] proposed methods to record the *normal* system call sequences so as to detect anomalous behaviors. These methods have to accumulate a large number normal behaviors with large-scale experiments; otherwise, they would generate high false positive rate. Mehdi *et al.* [4] and Rozenberg *et al.* [5] obtained sequential system calls of a *fixed* length using the N-gram segmentation model. This solution discriminates malicious behaviors from benign ones by checking whether the system call sequence is *only* invoked by malware, and gives the system call sequence a *goodness* value used to calculate the goodness of a program. However, both methods view system call sequences in *fixed* length, which is the main drawback, because different malicious behaviors might invoke various quantities of system calls. Lin *et al.* [6] built a *variable-length* system call sequences as behaviors and applied Bayes' theorem to calculate the malicious degree (MD) of evaluated behaviors and programs. The value indicates the *maliciousness* and can also be used to detect android malicious repackaged applications. Later, Lin *et al.* [7] used a similar concept to classify obfuscated bot binaries. Experimental result showed that their framework could attain high accuracy in classifying bot. These papers show that a system call sequence is actually a good feature to detect and classify malware.

Observing the aggregated external activities of malware can provide a helpful macroscopic point, such as modifying files. Liu *et al.* [8] proposed a mechanism that defines malicious behavior features (MBFs) and evaluates the malware with the MBFs. Recently, Tsai [9] proposed an effective malware detection scheme, called ANN-MD, to obtain the representative behaviors of the malware using three well-known sandboxes—the GFI sandbox, the Norman sandbox, and the Anubis sandbox—to obtain 13 representative behaviors of a program. It then employs artificial neural networks (ANNs) to calculate the MD of a program.

Although both internal and external observations can help us to find malware's trail, they still have some limitations [10]. Internal observation provides a much more fine-grained way to diagnose symptoms of malware, which makes it more time-consuming to dissect malware. By contrast, external observation is attractive because of its rapid examination but results in coarse-grained inspection. We conducted some experiments to evaluate the time cost of external and internal observations. An external observation, such as using *sandboxes* with ANNs to observe malware behaviors, took about 3 min. The time cost of the internal observation, such as using *system call* approaches with the longest common substring (LCS), took approximately 15 min. We observed that the sandbox-based approach was *faster* on analyzing a program, whereas the system call-based approach analyzed a program in a *finer-grained* way, with higher detection accuracy.

Because internal and external observations have different time costs and achieve different degrees of detection accuracy, this work proposes a *hybrid* system to combine the advantages of these two approaches. We deployed a new three-phase approach: the external observation in the first phase and internal observation in both second phase and third phase. In the first phase, we used a sandbox to obtain all behaviors and then calculated the MD values for each to-be-detected program using the ANN. In the second phase, we discovered some common behaviors between different malware by extracting the LCS of system call sequences and then employed the Bayesian model to check for likely malicious behaviors. While detecting, the second phase only processed the to-be-detected programs (DP) *not* caught or ruled out by the first phase, which reduced the loading to the slower second phase. In the third phase, on the basis of those malicious behaviors, we defined malware behavior vectors and calculated the cosine similarity to classify malware. Selecting the ANN and the Bayesian model was based on past experience. Our previous work using ANN [9] and the Bayesian model [6,7] achieved very good detection accuracy, so they were suitable choices.

Our method is based on behavior-based analysis and has the following three advantages: (i) even if malware has been encapsulated or obfuscated, our method can still capture the malicious behavior by extracting their common subsequences; (ii) our method uses a three-phase approach, rather than a one-phase method, by leveraging existing solutions to improve detection accuracy and reduce time costs on malware detection; and (iii) our method not only detects malware but also classifies the malware. Known malware is classified to the type it belongs to, whereas a new malware is classified to an unknown type.

The rest of this paper is organized as follows. In Section 2, we first review some related works and then compare existing schemes with ours. In Section 3, we present the proposed scheme in details. The experiment setup including the implementation of the three-phase scheme is described in Section 4. Section 5 presents experimental results. Finally, concluding remarks are given in Section 6.

2. RELATED WORK

In order to overcome the drawback of signature-based anti-malware solutions, many behavior-based anti-malware solutions (which observe malware behaviors by internal [1–7] or external observation [8,9]) have been proposed. In this section, we review some of the related work on malware and intrusion detection, identify particular flaws, and then propose improvements.

For internal observation, Forrest *et al*. [1] first proposed a method to record the series of normal system calls by executing benign programs into the database. When a *compromised* program executes malicious codes, the *anomalous* system call sequence patterns can be detected because those patterns do *not* exist in the database. Since then, many other system call-based approaches have been proposed. In 2006, Mutz *et al*. [2] identified *anomalous* behaviors by system calls using different criteria: string length consists of human-readable characters and rarely exceeds a hundred characters; string character distribution is based on the observation that strings have a regular structure and almost always contain only printable characters; and token finder determines whether the values of a certain system call argument are drawn from a limited set of possible alternatives, that is, elements of an enumeration. These two *anomaly-based* detection methods are required to first accumulate normal behaviors with large-scale experiments, or they would introduce high *false positive* rates. Besides, attackers might *evade* those defenses.

Recently, Mehdi *et al*. [4] came up with a method, named IMAD, which obtained sequential system call in *fixed* length using the *N-gram segmentation* model. This method discriminates malicious behaviors from benign ones by checking whether the system call sequence is *only* invoked by malware, and assigns the system call sequence a 'goodness value'. If the sequences are invoked by *both* malware and benign programs, IMAD will evaluate their goodness by *genetic* algorithm. Those goodness values can then be used to calculate an overall impression value of a process. The larger the impression value of a process, the greater probability that the process is declared as benign. Later, Rozenberg *et al*. [5] tackled system call sequences with SPADE *sequence mining* methodology and *genetic* algorithms. Those system call sequences invoked only by malware were reserved for later detection. The main drawback of both methods is that they view system call sequences in *fixed* length because different malicious behaviors might invoke *various* quantities of system calls.

Lin *et al*. [6] extracted the longest common system call subsequences from the same types of malware to represent behaviors. Subsequently, they employed Bayes probability model to assign a value for each behavior. This value indicates maliciousness and can be used to detect obfuscated programs. Although this method handled *variable-length* system call sequences, it is inclined to miss many of the sequences because there is always only one longest common subsequence (LCS) between two system call sequences. Lin *et al*. [7] used the similar concept to develop a framework for the automatic analysis and classification of bot binaries, which uses dynamic analysis to extract system call sequences from the bot binaries. The framework then classifies the binaries on the basis of the LCS similarity of system call.

The papers [1–6] noted earlier show that a system call sequence is actually a good feature to use to detect and classify malware. The key concept of using system call sequences during runtime is that, even if a malicious program can be camouflaged as a benign application or obfuscated to avoid detection, its malicious behavior would still appear in the invoked system call sequences. From observation, the system call sequences in a malicious program can be roughly divided into four stages, namely *program loader*, *unpacking handler*, *execution handler*, and *exit handler*. Program loader includes system calls related to the initialization of a new process, for example, memory allocation and loading-related library files. Unpacking handler is mainly about the decomposition of the program itself. The most important part is the third stage, execution handler, which contains the system calls made by the original malicious program itself. System calls in this stage characterizes the behavior of a program. The final stage, exit handler, contains system calls used for the deallocation of resources at the time of process termination.

For external observation, Liu *et al*. [8] extracted the *MBF* of malware by observing processes of Windows systems and then using the MBF in a sandbox to detect the malware, where MBF is a three-tuple, that is, Feature_id, Mal_level, and Bool_expression. Note that Feature_id is a string identifier that is used to uniquely represent an MBF; Mal_level divides an MBF into three malicious levels: high, warning, and low; Bool_expression is a Boolean expression that specifically defines the behavior of an MBF; and the MBF can be used to calculate the MD of a suspicious program. If the program conforms to *more* MBFs, there is a greater probability that it is malware. Tsai [9] used three sandboxes, that is, GFI sandbox, Norman sandbox, and Anubis sandbox, to obtain representative behaviors of a program. They selected 13 behaviors that malware frequently carry out but rarely benign programs do. Finally, this method constructed an MD expression by using ANN for malware detection. However, these two methods, of which one declared *only three* malicious levels and the other considered *only* those behaviors related to malware, lack precise inspection.

Because internal and external observations have different time costs and achieve different degrees of detection accuracy, this work proposes a new three-phase approach, with the external observation in the first phase and internal observation in both second and third phases. In the first phase, using the similar concept as in [9], we use a sandbox to obtain all behaviors and then calculate the MD values for each to-be-detected program using the ANN. In the second phase, using a similar concept to that in [6], we identify some common behaviors between different malware by extracting the LCS of system call sequences and then employ the Bayesian model to check likely malicious behaviors. In the third phase, on the basis of those malicious behaviors, we then classify the different types of malware, using a similar concept to that in [7].

In our architecture, the ANN and the Bayesian model used in the first and second phases, respectively, can be replaced with other similar techniques. We selected these two methods mainly for two reasons. First, they are commonly used in this field. Second, our previous work using ANN [9] and the Bayesian model [6,7] achieved very high accuracy, confirming that they are suitable choices.

Note that our major contribution is to propose a new three-phase architecture by leveraging existing malware detection mechanisms. Thus, we do not pay much attention to improving individual malware detection in each phase. Compared with existing methods, our method is based on behavior-based analysis and has the following three advantages: (i) even if malware have been encapsulated or obfuscated, our method can still capture the malicious behavior by extracting their common subsequences; (ii) our method uses a three-phase approach, rather than a one-phase method, by leveraging existing solutions to improve detection accuracy and reduce time costs on malware detection; and (iii) our method not only detects malware but also classifies it. Malware is classified to the type it belongs to, whereas new malware is classified to an unknown type. However, our method has the drawback that the detection accuracy of first phase is affected by the adopted sandbox and neural network. In the first phase, we use a sandbox to obtain all behaviors and then calculate the MD values for each to-be-detected program using the ANN. Different sandboxes and neural networks have distinct performance and limitation, and hence, adopting which sandbox and neural network will affect the final performance of our approach. The related works referred to are summarized in Table I.

3. THREE-PHASE BEHAVIOR-BASED ANALYSIS

Considering both detection accuracy and time cost, we propose a three-phase approach with the front two phases serving detection and the rear phase serving classification. Two different mechanisms are adopted: sandbox-based detection mechanism (SDM) and system call-based detection mechanism (SCDM). SDM provides a faster way to observe a program, whereas SCDM observes a program in a finer-grained way and achieves better detection accuracy. The proposed three-phase behavior-based analysis is deployed with SDM in the first phase, SCDM in the second phase, and Behavioral Classifier in the third phase.

Each sample must pass through SDM. If SDM judges a program as suspicious, that program is sent to SCDM for further examination. Finally, the intrusive or non-intrusive malware is also identified according to its exhibited behaviors. Note that there are two processing flows in our approach, that is, training flow and detection/classification flow. For the training flow, we target at digging out the *MB* together with *IB* using *BP* and *MP* and then define malware types, *T*, depending on the *MB*. For the detection/classification flow, our goal is to detect and classify malware using *MB* and *T*. The detection/classification flow is processed after the training flow is done. Notations used in this paper are summarized in Table II, and the architecture of the proposed scheme is illustrated in Figure 2.

3.1. Sandbox-based detection mechanism

As the training flow in Figure 2(a) shows, we first submitted benign and malicious samples, bp_i and mp_j, to a sandbox for training and collected all corresponding runtime behaviors from the sandbox. Next, we selected *representative* malicious behaviors into *MB* by calculating their *appearance frequency* in the malicious and benign samples and used binary vectors to keep trace of exhibited representative malicious behaviors for each sample. Afterward, those binary vectors were fed into an ANN algorithm to adjust the *weight* of each behavior in *MB* dynamically. The ANN is one kind of machine learning algorithm in the field of artificial intelligence. An ANN was composed of several interconnected artificial neurons. Each neuron

Table I. Related works on anti-malware solutions.

Category	Schemes	Core concept	Goal
System call	Forrest *et al.* [1]	Sequence miss match rate	Intrusion detection
	Mutz *et al.* [2]	String distribution, structural inference	Intrusion detection
	Mehdi *et al.* [4]	SPADE with genetic algorithm	Malware detection
	Rozenberg *et al.* [5]	N-gram with genetic algorithm	Malware detection
	Lin *et al.* [6]	LCS with Bayesian model	Obfuscated malware detection
	Lin *et al.* [7]	LCS with similarity	Obfuscated bot classification
Sandbox	Liu *et al.* [8]	Malicious behavior feature	Malware detection
	Tsai [9]	ANN	Malware detection
System call and sandbox	This work	Hybrid of ANN and Variable-length common sequence with Bayesian model	Malware detection and classification

Table II. Notations used in the proposed scheme.

Category	Notations	Mathematical properties	Descriptions
Program	BP	$\{bp_1, \ldots, bp_i, \ldots, i = 1 \ldots I\}$	A set of benign programs
	MP	$\{mp_1, \ldots, mp_j, \ldots, j = 1 \ldots J\}$	A set of malware
	DP	$\{dp_1, \ldots, dp_k, \ldots, k = 1 \ldots K\}$	A set of to-be-detected programs
Behavior	BB	$\{bb_1, \ldots, bb_m, \ldots, m = 1 \ldots M\}$	Benign behaviors obtained from BP
	SB	$\{sb_1, \ldots, sb_n, \ldots, n = 1 \ldots N\}$	Suspicious behaviors exhibited by MP
	MB	$\{mb_1, \ldots, mb_o, \ldots, o = 1..O\} = \{nb_1, nb_2, \ldots, nb_Q, ib_1, ib_2, \ldots, ib_{O-Q}\}$	Malicious behaviors of MP, where $MB \subseteq SB$
	NB	$\{nb_1, \ldots, nb_q, \ldots, q = 1 \ldots Q\}$	Non-intrusive behaviors of MP
	IB	$\{ib_1, ib_2, \ldots, ib_{q'}, \ldots, q' = 1 \ldots O - Q\}$	Intrusive behaviors of MP
	B	$\{b_1, \ldots, b_k, \ldots, k = 1 \ldots K\}$	Behaviors of DP
Type	T	$\{t_1, t_2, \ldots, t_v\}$, where $t_v = \langle nb_1, nb_2, \ldots, nb_Q, ib_1, ib_2, \ldots, ib_{O-Q} \rangle$	A set of type vectors

of ANN can receive data from multiple inputs and perform a local computation. The output of a neuron is determined by an activation function. We employed feed-forward ANN to cluster malware and benign software. Finally, an MD expression was constructed, and two MD values could be manually set as the thresholds, upper bound and lower bound, by checking the evident misjudgment. A few samples fall into the middle of the distribution and is named the *ambiguous* area.

The detection flow assesses whether a given sample, dp_k, is malware. In the same way, we collected the runtime behaviors and calculated the MD values of the sample. If its MD value is larger than the upper bound, the program is judged as malware. If its MD value is lower than the lower bound, it is judged as benign. For those suspicious samples whose values fall into the ambiguous area, the processing should be handled by SCDM rather than handled by SDM.

The sandbox-based detection mechanism was also used as the first-phase filter. We adopted three filters, that is, *benign filter*, *malware filter*, or *fuzzy filter*. A benign filter was used to filter out the benign samples. The malware filter performed exactly the other way around. A fuzzy filter was used to filter out samples in the ambiguous area.

3.2. System call-based detection mechanism

As illustrated in Figure 2(b), SCDM contains four modules and a database. The System Call Tracer traces programs' trails by recording their issued system calls and then constructs *SB* and *BB* in the training phase and behaviors (*B*) in the detection phase. The Common Sequence Extractor observes programs' common behaviors in the training phase by mining all common system call subsequences. Afterward, the Bayes Analyzer investigates those common behaviors using the Bayes probability model, together with *BB* to put the likely malicious common system call subsequences into *MB*. Then, in the detection phase, the Sequence Detector evaluates dp_k by comparing dp_k's system call sequences with *MB*. The database serves as a storage pool for *SB*, *BB*, *B*, and *MB*.

3.2.1. System Call Tracer

Malware cannot alter the invocations of system calls while carrying out some critical actions. Hence, for all of programs to be analyzed in *BP*, *MP*, and *DP*, each program can be individually executed to record the runtime-issued system calls. The recorded system calls for each process would be ordered by timeline to form a system call sequence. Then, the system call sequence can be regarded as the trail of bp_i, mp_j, or dp_k, and stored in *BB*, *SB*, or *B*, respectively.

3.2.2. Common Sequence Extractor

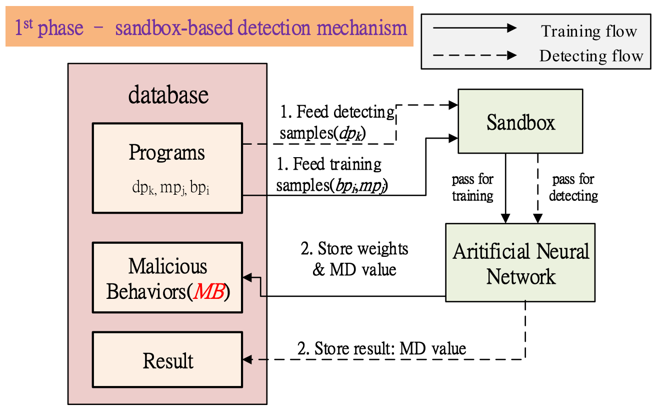
Because common malicious behavior is the great identifier of malware, the system call sequences recorded from the malware should have similar system call subsequences. LCS is a sequence mining algorithm for extracting the longest common consecutive subsequence, also named as substring. For any two sequences, only one LCS can be extracted. In order to dig out *all* common subsequences, we *recursively* extract the LCS between any two sequences to dig out all common subsequences between any two sequences. Whenever mp_j in *MP* is fed into the extraction procedure, we can extract all common subsequences between mp_j and any other malware in *MP*.

3.2.3. Bayes Analyzer

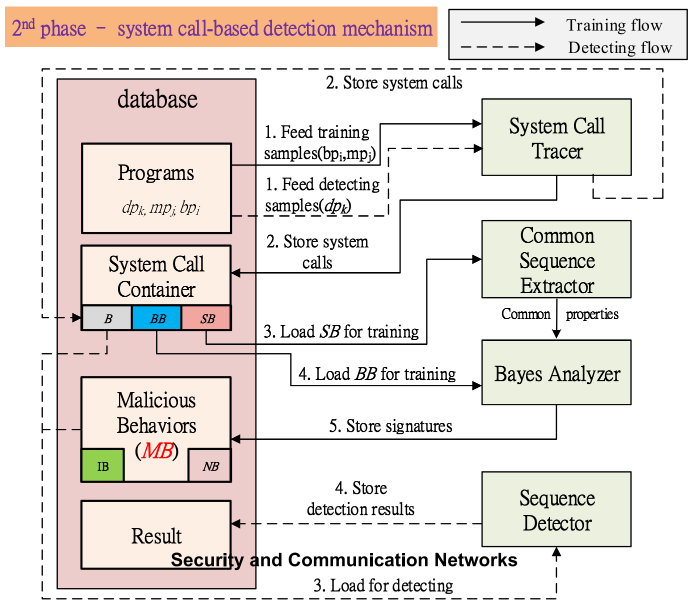
We acquired many system call subsequences after the extraction procedure. Not all of them can be regarded as malicious behaviors because malware might exhibit both benign and malicious behaviors. Accordingly, we have to evaluate every extracted substring in terms of the probability of appearance under the Bayes probability model. The formula is

$$P(M|sb_n) = \frac{p(sb_n|M)p(M)}{p(sb_n|M)p(M) + p(sb_n|B)p(B)} \tag{1}$$

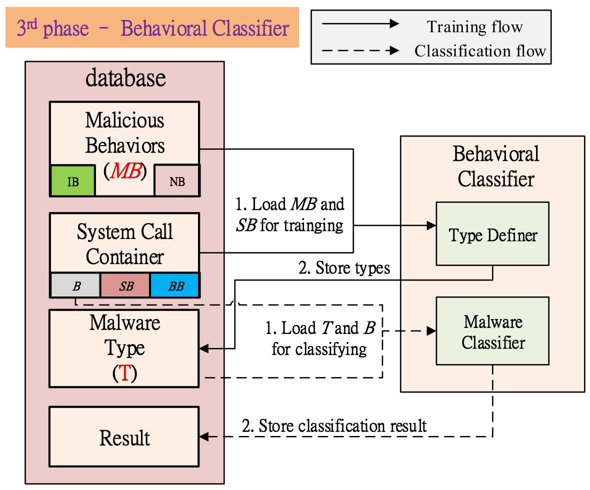
where sb_n is the extracted system call substring to be evaluated. $P(B)$ denotes the probability that the given programs are benign programs, whereas $P(M)$ denotes the probability for malware. $P(sb_n|B)$ and $P(sb_n|M)$ represent the probability that sb_n is carried out by benign programs and malware,


1st phase – sandbox-based detection mechanism
Training flow
Detecting flow
database
Programs
dpk, mpj, bpi
Malicious Behaviors(MB)
Result
1. Feed detecting samples(dpk)
1. Feed training samples(bpi,mpj)
Sandbox
pass for training
pass for detecting
Aritificial Neural Network
2. Store weights & MD value
2. Store result: MD value

(a) Architecture of SDM



(b) Architecture of SCDM



(c) Architecture of behavioral classifier

Figure 2. The proposed three-phase behavior-based analysis.

respectively. Then, we can obtain the probability $P(M|sb_n)$, which implies the probability of a program being malware given the condition that the program practiced sb_n. On the basis of the experimental results of Lin *et al.* [6], sb_n with a probability of 1 can achieve the best representation for a malicious behavior. Because intrusive behaviors are only exhibited by intrusive malware, we also label mb_o with its owner to tell intrusive behaviors apart. Finally, *MB* and *IB* can be obtained.

3.2.4. Sequence detector

As mentioned earlier, the calculated probability of mb_o is 1. It is inferred that only malware carries out mb_o. For each dp_k, we first record its runtime-issued system call sequence as b_k and then compare the b_k with all elements in *MB*. Once b_k matches mb_o, dp_k is deemed to be malware.

3.3. Behavioral Classifier

Behavioral Classifier is designed to distinguish different types of malware. We describe two major modules, that is, Type Definer and Malware Classifier, as shown in Figure 2(c).

3.3.1. Type Definer

We use a behavior vector t_v to represent the type of malware, where t_v is defined as

$$\begin{aligned} t_v &= < mb_1, mb_2, \ldots, mb_o > \\ &= < nb_1, nb_2, \ldots, nb_q, ib_1, ib_2, \ldots, ib_{o-q} > \end{aligned}$$

where nb_q and ib_{o-q} are non-intrusive behavior and intrusive behavior, respectively. Behavior vectors are identified with o-tuple of Boolean numbers, where '0' denotes the absence of mb_o, and '1' denotes the presence of mb_o. A behavior vector t_v is built by noting down what malicious behaviors a piece of malware performs. Subsequently, in order to make sure that each t_v retained in the set of representative behavior vectors, T, *resemble* close enough to the vectors of the same type but *differ* far enough from the vectors of the other types, we need a training flow to remove less-representative t_v. We calculate the cosine similarity [11] between t_v and t_v' by

$$\frac{t_v \cdot t_v'}{|t_v| \times |t_v'|} = \frac{\sum_{i=1}^{o} t_{v_i} \times t_{v_i}'}{\sqrt{\sum_{i=1}^{o} (t_{v_i})^2} \times \sqrt{\sum_{i=1}^{o} (t_{v_i}')^2}} \tag{2}$$

where t_v' is any other behavior vector of the same type as t_v. According to formula (2), all the calculated similarity values are between 0 and 1. The higher the similarity value is, the more similar two behavior vectors are. We keep the calculated maximal similarity value T_α and set a value as the threshold T_τ, where T_τ is decided heuristically, to determine whether to retain t_v. If the T_α is lower than the T_τ, t_v is not sufficiently representative and should be discarded. Otherwise, we calculate the cosine similarity between t_v and the behavior vectors of other types, t_v'', and then keep the calculated maximal similarity value T_β. If the T_β is larger than the T_τ, t_v does not differ far enough from the vectors in the other types and should be discarded too. In this way, we can retain only representative behavior vectors in T for classification. The aforementioned training flow is depicted in Figure 3.

3.3.2. Malware Classifier

In the classification flow, we build a behavior vector for each dp_k to keep track of its exhibited malicious behaviors and then calculate the cosine similarity between dp_k's vector and all behavior vectors in T. We classify dp_k into the type that its behavior vector represents, where the calculated similarity between dp_k's vector and t_v is *greatest*. In addition, a threshold is also set as the lower bound for the calculated similarity to determine whether the dp_k belongs to a known type or an unknown type. Note that if we want to classify the malware caught by SDM, we should also collect the runtime-issued system calls of that malware by System Call Tracer.

4. EXPERIMENT SETUP

We collect 1800 executable programs as the sample space, comprising 1000 malicious and 800 benign programs, with 900 programs (500 malicious and 400 benign programs) for training and detection, separately. Note that malware samples are collected from 'VX Heaven' (http://vxheaven.org/) and belong to five well-known types, namely Backdoor, Bot, Hoax, Trojan, and Worm. 'VX Heaven' contains a massive collection of malware. People who are interested in this kind of work can find useful resources, including documents, samples, and source codes.

These types were used to evaluate the proposed classification method. The benign programs were collected from Windows OS system directories and CNET.com (http:// www.cnet.com/). All benign programs were checked by several online anti-virus tools [12].

4.1. SDM implementation

We input our training samples to the GFI sandbox to monitor the external behaviors. The malware collected are Trojan, Backdoor, Worm, Bot, and Hoax. After a large statistical calculation and eliminating those behaviors that malware and benign programs rarely perform, such as *Opens Physical Memory* and *Modifies Local DNS*, only 12 representative behaviors were kept, where their appearance frequencies in malware or benign programs were above 5%. The representative behaviors included *Starts EXE in System*, *Starts EXE in Documents*, *More than 5 Processes*, *Makes Network Connection*, *Injected Code*, *Hooks Keyboard*, *Deletes Original Sample*, *Deletes File in System*, *Creates Mutex*, *Creates Hidden File*, *Copies to Windows*, and *Checks for Debugger*.

Next, we used Matlab 8.0.0 'Neural Network' toolbox to implement the ANN. The input layer of the ANN consists

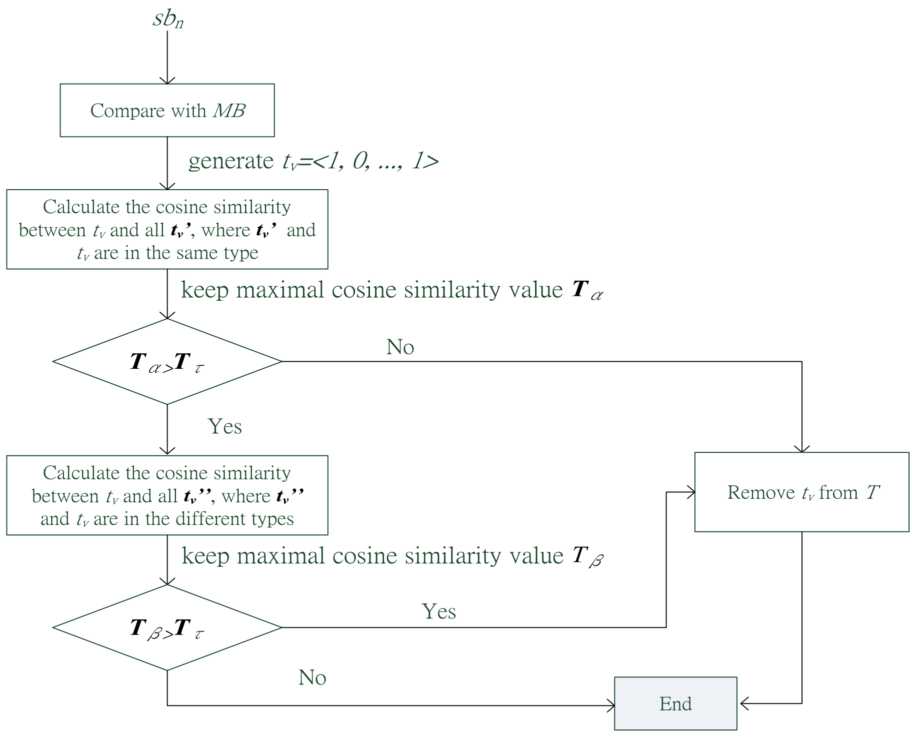

sb_n
Compare with MB
generate t_v=<1, 0, ..., 1>
Calculate the cosine similarity between t_v and all t_v', where t_v' and t_v are in the same type
keep maximal cosine similarity value T_α
$T_\alpha > T_\tau$
No
Yes
Calculate the cosine similarity between t_v and all t_v'', where t_v'' and t_v are in the different types
keep maximal cosine similarity value T_β
$T_\beta > T_\tau$
Yes
No
Remove t_v from T
End

Figure 3. Type training flow.

of 12 representative behaviors. We adopted the built-in function *initnw* to distribute the initial weight for each neuron. Choosing enough numbers of samples and neurons, the resultant ANN with weights can be obtained after the training. Also the MD value of each sample was also obtained and stored in the database.

4.2. SCDM implementation

System Call Tracer comprises two components, the Controller and the Recorder. The Controller coordinates the execution and recording environment; it fetches a program from the database and then executes the program with the Recorder. The Controller is also responsible for the integrity of the recording environment. Before a program is processed, the Controller restores the snapshot to preclude the possibility that a program is influenced by the other programs' execution results.

The Recorder relies on the dynamic instrumentation tool PIN (http://www.pintool.org/) to record the system calls invoked by a program during its execution. In later experiments, the length of each recording is set to 3 min because the number of system calls invoked does apparently not increase after this time length. The just-in-time compiler of PIN can monitor the SYSENTER instruction. We use PIN API *PIN_AddSyscallEntryFunction*() to instrument the monitoring routine *callback_before*() immediately before each SYSENTER/INT 2Eh instruction, so that the Recorder registers the instant that the program invokes a system call. The PIN API provides us the information including the thread ID, system call ID, and the system call arguments. We only retain the thread ID and system call ID while ignoring the arguments of system call.

For Bayes Analyzer, on the basis of Lin *et al*. [6], we set a probability of 100% as the threshold for Equation 1, which implies that only the a program that meets a probability of 100% is retained; otherwise, it will be filtered out.

For Sequence Detector, as long as the program matches any sample in our *MB* database, it must be malware because the probability of Bayes Analyzer is set to 100%.

5. EXPERIMENT RESULTS

We present our results in terms of four metrics, that is, true positive ratio (TPR), true negative ratio (TNR), false positive ratio (FPR), and false negative ratio (FNR). TPR and TNR denote the ratio that we truly identify malware and benign programs, respectively. FPR and FNR mean that benign programs or malware are mistakenly identified. We measured the time cost for SDM as well as SCDM and compared three strategies that two detection mechanisms serve in different phases. Furthermore, we give results for malware classification and intrusive malware recognition.

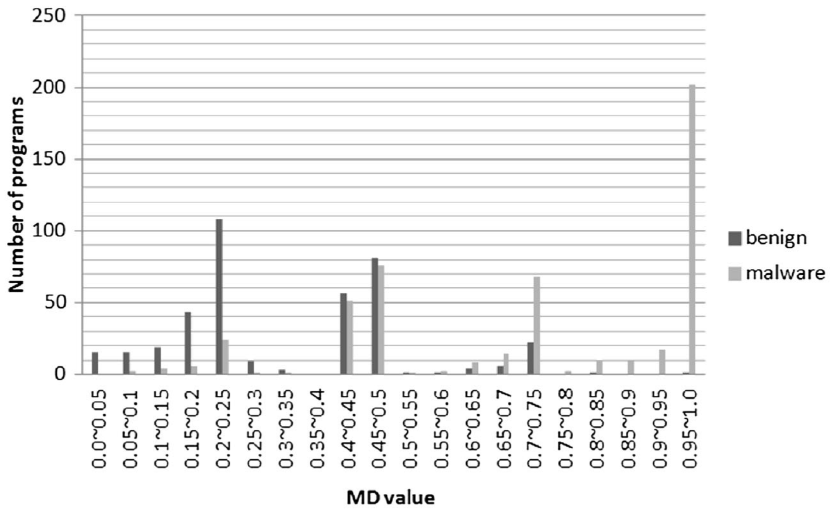
5.1. Impact of MD threshold

This experiment evaluates sandbox-based detection ability. As discussed earlier, we can obtain the MD value for each program using the ANN. In order to delimit the ambiguous area and determine the optimum MD threshold, the distribution for MD values is shown in Figure 4.

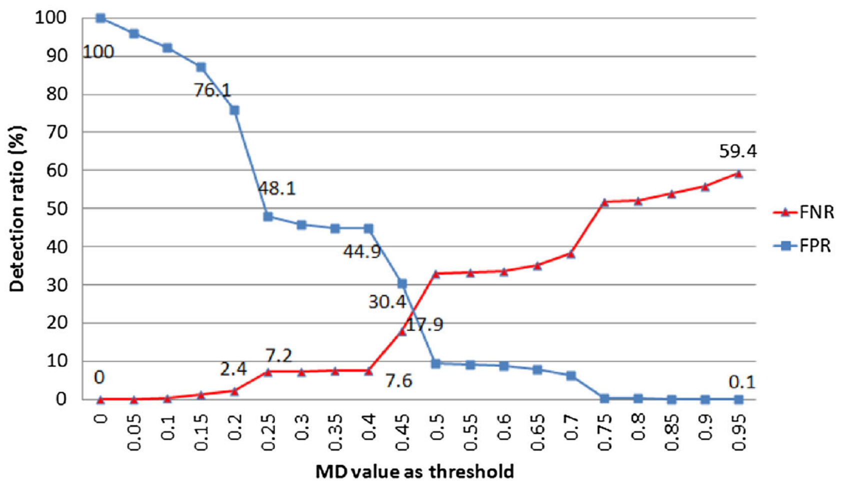
As seen in Figure 4(a), there is an obviously ambiguous area from 0.2 to 0.75. We observed that SDM could not discriminate between malware and benign programs within the ambiguous area, whereas SDM could discriminate between them well outside the ambiguous area. Therefore, we investigated detection ratio versus different MD thresholds in Figure 4(b), and we observed that FNR increased sharply within two intervals, that is, from 0.2 to 0.25 and from 0.4 to 0.5. We looked into the cause of the trend and noticed that a large quantity of benign programs and malware practice the same suspicious behaviors and their calculated MD values would thus be similar. This might result from the inability of the GFI sandbox to dissect programs in a fine-grained way. If we wanted to retain the FNR at less than 10%, the best MD threshold was 0.4.

5.2. Detection ability versus time cost

Two different detection approaches, that is, SDM and SCDM, were adopted in our system. Table III(a) shows a big gap in the FPR produced by SDM and SCDM. Observing behaviors with a sandbox could catch suspicious programs but might result in higher false positive. We also measured the time cost in observing time and detecting time. Observing time means the time consumed by the GFI sandbox or the System Call Tracer to observe the behaviors of a program, and detecting time refers to how much time the ANN or the Sequence Detector takes to process a program. SCDM spends much more time than SDM, and the observing time for both mechanisms dominates the time cost. Because of the larger number of system call sequences observed, SCDM obtains better detection ability.



(a) Number of programs vs. MD values



(b) Detection ratio vs. MD threshold

Figure 4. Impact of MD values.

Table III. SDM and SCDM: comparison and combination.

(a) Detection ratio versus time cost

Mechanism	FNR (%)	FPR (%)	Time cost (s)	
			Observing	Detecting
SDM	7.6	44.9	180	0.068
SCDM	7.4	7.5	900	0.35

(b) Decision on first phase

Strategy		% of processed programs		Total time cost (s/program)	FNR for first phase	FPR for first phase
		First phase	Second phase			
SDM → SCDM (fuzzy filter)	Benign	100	76	731	2.4%	0.4%
	Malware	100	49.4			
SDM → SCDM (benign filter)	Benign	100	76.1	973	2.4%	—
	Malware	100	97.6			
SDM → SCDM (malware filter)	Benign	100	99.4	837	—	0.4%
	Malware	100	51.8			
SCDM → SDM (benign filter)	Benign	100	19.5	1015	1.2%	—
	Malware	100	98.8			
SCDM → SDM (malware filter)	Benign	100	80.5	966	—	19.5%
	Malware	100	1.2			

5.3. Detection: two-phase based on time cost, FN, and FP

Our major contribution is to leverage existing malware detection mechanisms and to propose a new three-phase architecture (two-phase detection, SDM + SCDM) for improving detection accuracy and reducing time costs. We thus pay more attention to the feasibility of this architecture and compare the performance of our two-phase detection and one-phase detection (SDM, SCDM). The advantages and drawbacks of individual existing malware detection are not covered in this work and can be found in other work.

Because we take a two-phase behavior-based approach for malware detection, we have to decide which detection mechanism should be put in the first phase. As the first phase defense, both accuracy and time cost should be considered. As shown in Table III(b), given the condition that FNR of both SDM and SCDM remains within the tolerance, if SDM plays the front, there are 76% of benign programs and 49.4% of malware sent to second phase, compared with 19.5% and 98.8% for SCDM as a benign filter. SCDM as a malware filter introduces such high FPR that we should abandon it. In considering the time cost for analyzing a program, the first strategy took 731 s to analyze a program, which was better than all the other strategies. We deduced that the difference in time cost was a result of SDM ignoring the programs whose calculated MD values are in the ambiguous area. SDM can act as both malware and a benign filter, that is, fuzzy filter. On the other hand, SCDM cannot generate such a fuzzy decision so that it can serve as a malware filter only. We conclude that SDM as a fuzzy filter should serve the first-phase defense, that is, the first row in Table III(b).

Comparing SDM + SCDM with the first-phase SDM and SCDM, the FNR/FPR is 7.7%/7.5%, 7.3%/48.1%, and 3.6%/6.8%, respectively. Although SDM alone does not perform well, it complements SCDM in the two-phase combination.

5.4. Classification by behavior vectors

To classify detected malware into defined types, we extracted the behavior vector of a malware and calculated its cosine similarity with the representative behavior vectors of well-known types of malware. If the cosine similarity of this malware does not exceed a threshold, it belongs to a *new* type. To find the representative behavior vectors of each known type among the 500 training malware samples divided into five types, we took four types as known types and the remaining type as the unknown type, and calculated the cosine similarity of the malware in the four known types to identify which malware could be representatives. The other 500 malware samples, also divided into five types, were classified into the four known types or the new type. We evaluated the classification accuracy to find that we could distinguish malware of known types from the unknown type, and malware of the unknown type from known types with an accuracy of 85.5% and 80%, respectively. The behavior vector does characterize the type of malware.

5.5. Intrusive versus non-intrusive

According to the statistics of all the malware from VX Heaven (http://vx.netlux.org/index.html), intrusive malware accounts for only 3.8% of the total. We observed that the proposed Behavioral Classifier could differentiate

Table IV. Behaviors carried by non-intrusive or intrusive malware.

	Non-intrusive malware				Intrusive malware
Malware type	Worm	Backdoor	Trojan	Hoax	Bot
Overlapping behavior ratio (%)	28.9	27.8	22.0	17.5	44.4

intrusive and non-intrusive malware with an accuracy of 85.8%. There is still some malware incorrectly recognized, probably because some intrusive behaviors were mistaken, or the intrusive malware hid its intrusive behaviors. We then compared the behaviors that non-intrusive malware and intrusive malware practices and give the results in Table IV. We observed that *overlapping behavior ratio* indicates the percentage of the behaviors carried out by this type of malware and also carried out by other types of malware. For example, among all behaviors that Worm carries out, 28.9% is also carried out by other types of malware. In other words, 71.1% is carried out by worms only. We observed that intrusive malware has the greatest overlapping behavior ratio. More behaviors carried out by intrusive malware are also carried out by other types of malware. Therefore, we can deduce that intrusive behaviors are a minor part of the overall behaviors.

6. CONCLUSIONS

In this paper, we propose a three-phase behavior-based approach, with the front two phases serving detection and the rear phase serving classification. Compared with existing methods, our method is based on behavior-based analysis and has the following three advantages: (i) even if malware have been encapsulated or obfuscated, our method still has the chance to capture the malicious behavior by extracting their common subsequences; (ii) our method uses a three-phase approach, rather than a one-phase method, by leveraging existing solutions to improve detection accuracy and reduce time costs on malware detection; and (iii) our method not only detects malware but also tries to classify malware. Malware would be classified as the type it looks most like, whereas new malware would be classified as an unknown type. However, our method has a drawback that the detection accuracy of first phase would be constrained by the detection accuracy of adopted sandbox and neural network.

We summarize some insights as follows. First, the first phase takes about 180 s to analyze a program, whereas the second phase takes approximately 900 s. However, the first phase introduces 7.6% FN and 44.9% FP, compared with 7.4% FN and 7.5% FP of the second phase. The second phase takes more time to achieve a better performance. Next, the integrated two-phase detection approach performs better than any one-phase approach alone in both detection accuracy and time cost, where it produces 3.6% FP and 6.8% FP and spends 731 s on analyzing a sample. Finally, the proposed approach can distinguish malware of known types from unknown type with the accuracy of 85.8% and discriminate the malware of unknown type from known types with the accuracy of 80.0%. It can also recognize intrusive malware with an accuracy of 82.7% and non-intrusive malware with an accuracy of 88.9%.

There are further issues to pursue. How to further improve the accuracy in the behavior-based detection through the sandbox and system call sequences requires more studies. Clustering behavior vectors into self-defined types beyond the current five well-defined types is another interesting topic. In our architecture, the ANN and the Bayesian classifier can be replaced with other similar classifiers. Thus, we will try other classifiers to know whether they can outperform the current approach using ANN and Bayesian. Finally, the contribution of this paper is the three-phase approach to improve detection accuracy and reduce time costs on malware detection. It is proven that this approach is effective on detecting malware for Windows OS. We will further investigate detecting mobile malware using this proposed approach in the future.

REFERENCES

1. Forrest S, Hofmeyr SA, A Somayaji, Longstaff TA. A Sense of Self for Unix Process, Proceedings of the 1996 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 1996; pp. 120–128, .
2. Mutz D, Valeur F, Vigna G, Kruegel C. Anomalous system call detection. *ACM Transactions on Information and System Security* 2006; **9**(1):61–93.
3. Warrender C, Forrest S, Pearlmutter B. Detecting Intrusions Using System Calls: Alternative Data Models, *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, 1999; pp. 133–145.
4. Mehdi SB, Tanwani AK, Farroq M. IMAD: In-Execution Malware Analysis and Detection, *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation*, Montreal, Canada, July 2009; pp. 1553–1560.
5. Rozenberg B, Gudes E, Elovici Y, Fledel Y. A Method for Detecting Unknown Malicious Executables, *Proceedings of the 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Nov. 2011; pp. 190–196.

6. Lin YD, Lai YC, Chen CH, Tsai HC. Identifying android malicious repackaged applications by thread-grained system call sequences. *Computers & Security* 2013; **39**(B):340–350.
7. Lin Y-D, Chiang Y-T, Wu Y-S, Lai Y-C. Automatic analysis and classification of obfuscated bot binaries. *International Journal of Network Security* 2014; **16**(6):506–515.
8. Liu W, Ren P, Liu K, Duan HX. Behavior-based malware analysis and detection. *Proceedings of International Workshop on Complexity and Data Mining*, Nanjing, China Sept. 2011; 39–42.
9. Tsai HY. Suspicious behavior-based malware detection using artificial neural network. *Master thesis, Institute of Network Engineering College of Computer Science*. National Chiao Tung University: Hsinchu, Taiwan, June 2012.
10. Moser A, Kruegel C, Kirda E. Exploring multiple execution paths for malware analysis, *IEEE Symposium on Security and Privacy*, May 2007.
11. Manning C, Raghavan P, Schütze H. *Introduction to Information Retrieval*. Cambridge University Press: Cambridge, United Kingdom, 2008.
12. Virus total. [online], Available: http://www.virustotal.com/ [accessed on March 2013].