2010 34th Annual IEEE Computer Software and Applications Conference

AntiBot: Clustering Common Semantic Patterns for Bot Detection *

Younghee Park Qinghua Zhang Douglas Reeves Vikram Mulukutla
Cyber Defense Laboratory
Department of Computer Science Department
NC State University, Raleigh, NC, USA
{ypark3,qzhang2,reeves,vmuluku}@ncsu.edu

Abstract

Among malicious software (malware), autonomous malicious programs, called bots, are a serious problem in the Internet. The bot writers have developed a variety of techniques to evade simple signature-based detection. Concise representations of malware behavior, or semantic patterns, are much harder to evade or obfuscate. However, generating a semantic pattern for every program instance is time-consuming, and comparing with a large number of patterns creates a challenge for timely identification of bots.

This paper proposes an automated approach to generate semantic patterns for bot detection. Unlike previous approaches, it is intended to find one pattern that accurately represents the important behavior of an entire class of bots, rather than of individual instances. Doing so has advantages for fast malware identification, and for distinguishing new classes of attacks from previously-seen attacks. The work uses static analysis to characterize bot behaviors , and proposes to use hierarchical clustering of the resulting semantic patterns from a set of bot programs. The goal is to identify critical, common semantic behavior that represents the functions of an entire class of the malware. This method has been prototyped and evaluated on real-world malicious bot software. Depending on parameter choices, our approach can achieve more than 95% detection rates and less than 5% false positive rates on a large set of bot programs and non-bot executables.

1. Introduction

The detection of bots, malware instances that run autonomously on a compromised host, has been a challenging problem since the bots, played a key role in launching all Internet threats such as DDoS, spam, information extortion, click fraud, etc [4, 12, 14, 22]. Signature-based malware detection has been the major defense against any malware including bots. However, this approach lacks insight into the malware program semantics (malicious behaviors) because it typically uses signatures (fixed byte strings) that represent characteristic instruction sequences derived from collected malware samples [5, 29, 7, 20]. Furthermore, this shortcoming permits attackers to avoid the detection by using program obfuscation techniques such as polymorphism (self-decrypting code), and metamorphism (code rewriting techniques).

Several techniques [28, 19] focus on characterizing some bot behaviors for bot detection. None of them includes comprehensive malicious behaviors to detect all of bot malware class such as bots, their variants, even different bot families, etc. Above all, the dependency of partial behaviors can be easily defeated since bots can be easily extensible by adding other functionalities [4, 8]. On the other hand, as alternative methods, we can use advanced malware detection techniques [24, 7, 15, 16] that focus on specifying and identifying malicious behaviors such as self-unpacking, hooking into the web browser, modifying critical data at load time, etc. Since these techniques are not dependent on specific instruction sequences, they are more robust against code obfuscation. However, the specification of malicious behaviors can be arduous, even and identifying the common behaviors that characterizes an entire class of bot malware requires manual inspection and processing, a hit-or-miss proposition.

In this paper, we propose a novel technique, called AntiBot, which identifies bots based on *common semantic behavior* mined from a large set of bot malware samples. The common semantic behavior is representative semantic patterns of an entire family of bot malware. These patterns can effectively detect metamorphic bots since the code obfuscation normally does not affect such semantic patterns in any significant way. The proposed approach is based on the recent advances in metamorphic malware detection [32]. By

*This material is based upon work supported by the the National Science Foundation under Award No. 0831081. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of NSF.

0730-3157/10 $26.00 © 2010 IEEE
DOI 10.1109/COMPSAC.2010.33

characterizing individual semantic patterns through static analysis, our method automatically extracts the common semantic behavior among an entire family of bot malware through a hierarchical clustering technique [9]. Specifically, first, the semantic patterns based on system calls are derived from each of entire bots. The extracted semantic patterns are classified into separate clusters according to a generated way of each system call. Second, the similarity measurement among the system calls in each cluster is performed by a maximum weighted algorithm [32]. Based on the two intermediate results, the common semantic patterns are automatically mined depending on several important parameters and criterions, which can determine the quality of the common semantic behavior for bot detection.

The proposed method has been implemented and evaluated on real-world bot programs, which are obtained from two trusted sources: the Cyber-TA project [1] and the Johns Hopkins University Botnet research project [23]. A set of experiments was performed to test the quality of the common behaviors which were generated with different parameter configurations. The detection rates of 97.62% and the false positive rates of 1.6% can be achieved under an optimized parameter setting. Our experimental results showed that our approach can successfully extract the common semantic behavior from a large set of bots, and validate the worthiness for the identification of the malware class collected from well-known public repositories. In addition, an 80% reduction was measured in the number of semantic patterns needed to detect the malware instances accurately.

Our contribution in this paper is four-fold. First, we develop a new approach to extract common semantic behaviors of a large set of bot malware class automatically, and use it for detection of such malware class. Second, the detection based on the common semantic behavior achieves high resilience to the deliberate insertion of *junk* code, which is not important to the exploit's intended function. Third, our experimental results demonstrate a major reduction in the number of common semantic behavior needed to identify bot mutants than a previous approach. Lastly, the proposed technique can be flexibly applied for other malware classes (e.g. rootkits, keyloggers, etc) in addition to the bot malware.

The rest of this paper is organized as follows. Section 3 introduces the proposed approach for automatically mining common semantic behavior of bot malware samples, and for employing the method to detect bot instances. Section 4 presents the results of evaluating the proposed method with real-world bot binaries. Section 5 discusses the method with limitations. Section 6 briefly reviews related work. Lastly, section 7 concludes the proposed work.

2. Motivation

The bot detection have been a hard problem because bots can include all kinds of malicious behaviors such as backdoors and trojan horses. To identify possible malicious behaviors of such malware can be a tedious and challenging task.

We discover that most bot programs have similar structures and functionalities since most bot mutants are derivatives from other existing bots or different bot families. In other words, they are developed by imitation or adaptation than innovation. Most of them have something in common with the following actions: a server connection, automatic start-up, system registry access, anti-virus software turn-off, bot software upgrade/uninstallation, local information extortion, bot process copy/hiding, various attack abilities, etc [4, 19, 28, 12]. Furthermore, to perform the specific behaviors, they have a tendency to use the same or similar function calls with a similar structure. From the characteristics, the mining of similarity among system calls from many bot binaries derives the common semantic behavior to represent entire of bot malware for bot detection. These commonalities are unique enough for bot identification.

The goal of this work is to identify a (metamorphic) bot malware by *automatically* deriving common semantic behavior for the malware family. An obvious advantage of using such common semantic behavior is ability to cope with metamorphic bots and provide forward detection of their mutations. It can potentially greatly reduce the labor of generating malicious behaviors that would be needed for each single instance of the malware. In this paper, the program behavior of interest is captured by the system calls that are made by the program. System calls are the primary interactions with the operating system and services it provides to applications. It has previously been suggested that they can effectively characterize a program's behavior [30]. In addition, data-mining techniques are frequently used to detect patterns in a large set of data and have already been applied for mining benign behavior models for intrusion detection [18, 30, 13, 26]. Therefore, our work takes a different approach in that the data-ming is used to find common malicious behaviors automatically, based on static analysis for the extraction of malicious behaviors.

3. AntiBot: System Design

In this section, the system design of AntiBot is presented. First, we generate common semantic behaviors from many bot binaries, which are unique enough for bot identification. Then, the mined common semantic behaviors can be directly used by a malware detector to detect new malicious programs of this malware class. Mining the common semantic behavior is performed by three key steps: static an-

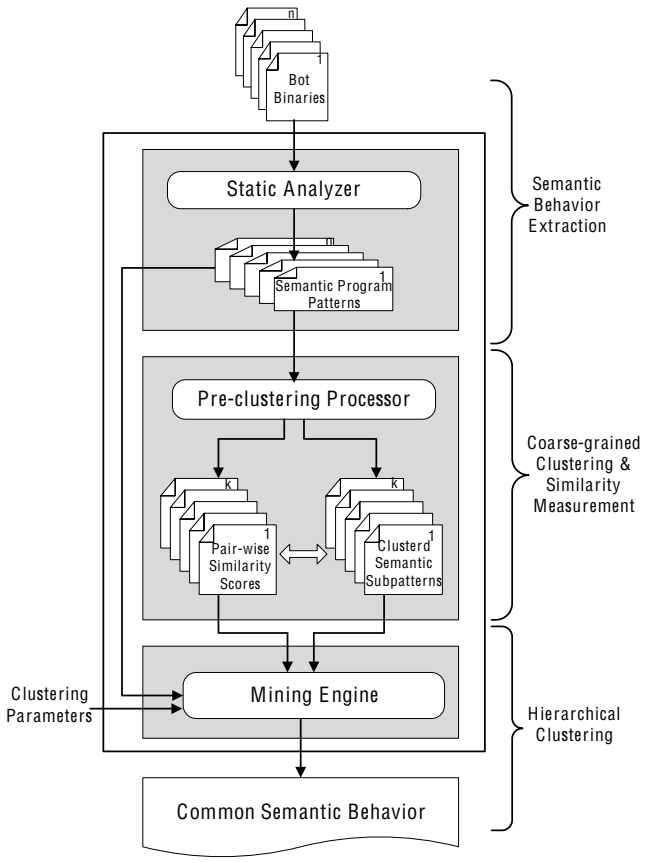


Figure 1. A System Architecture of Common Semantic Behavior Generation

alyzer, pre-clustering processor, and mining engine. These individual steps are first described in the following sections. Lastly, we discuss the detection method based on the common semantic behavior resulted from the three steps.

3.1 Overview

Figure 1 shows the major steps to produce common semantic behavior automatically: static analyzer, pre-clustering processor, and mining engine. Input to the procedure is a set of bot binaries, and output from the procedure is the common semantic behavior of these malware samples.

A static analyzer is responsible for processing the primary bot samples to derive their semantic patterns related to system calls. These patterns are written into separate files, one per malware instance. The static analyzer is modeled on the approach of [32], which statically analyzes obfuscated malware programs to characterize system call behaviors. The next step is a pre-clustering processor, which accepts the semantic patterns from the result of the static analyzer. The pre-clustering processor coarsely clusters a set of the sub-patterns representing these malware programs, and also outputs a set of similarity scores between pairs of sub patterns in the same cluster. The resulting classified data sets from the second step are intended to improve the performance and accuracy of the final functional step, a mining engine. The mining engine finds popular, or common, behavior patterns from these sub-patterns and their similarity scores by using hierarchical clustering depending on various clustering parameters .

The proposed method has three notable features. First, it is fully automated to find shared program malicious behaviors without any human intervention. Second, it works completely on binary code and requires no source code of the malware. Finally, it looks for semantically similar behaviors, not just identical behaviors. Focusing only on identical behaviors makes the mining procedure easy, but potentially misses shared malicious behaviors that involve intentional or non-intentional program obfuscation. This feature is reflected in the similarity computation for each cluster. From now, we elaborate on the proposed method with each step.

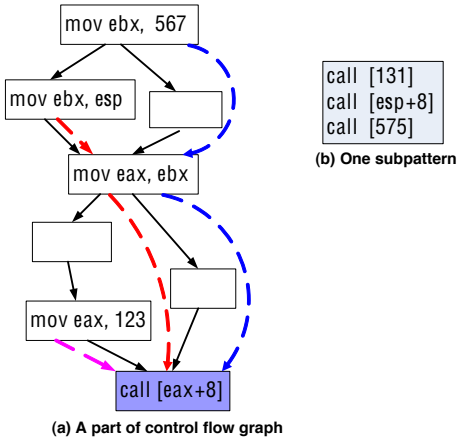


Figure 2. An Illustration of Semantic Pattern Generation. (a) A snippet of the control flow graph after disassembly of a program. Since a sub-pattern is based on a system call instruction and the instructions Each block indicates data flow dependency to the `call` instructions that affect the call target addres. Three maximal instruction traces, or program slices, for the `call` are shown as dashed lines. They are derived using backward data flow analysis, starting from the call instruction. (b) The generated sub-pattern. It is the result of the simple symbolic execution of the slices and is stored in an intermediate representation format.

3.2 Static Analyzer

The static analyzer processes a bot executable to extract its important program behavior which consists of seman-

tic patterns based system calls. We adopt the technique of MetaAware [23] to summarize semantic program behavior through static analysis. The semantics of the program are less affected by program obfuscation techniques. The highlights of the adopted static analysis method are briefly introduced below, since it is key to understanding our work.

A *semantic behavior* of the executable is derived based on the system calls the program makes. This pattern characterizes the program's behavior; for each program a single pattern is derived. A pattern itself consists of a set of sub-patterns. Each sub-pattern characterizes how a single system call is made as shown in Figure 2. This is done through control and data flow analysis of the disassembled program. The goal of analysis is to extract the system call instructions, and the instructions that prepare the parameters used by them. For each system call, a set of maximal instruction traces ending with that *call* are found. This set of traces is aggregated to form a sub-pattern. A maximal instruction trace, or program slice, is the longest loop-free sequence of instructions on which a system call instruction is dependent. Figure 2 illustrates sub-pattern generation with a simple example. Consequently, the semantic patterns of the program produced by this static analysis technique are input to the next step, which performs a coarse classification of the sub-patterns.

This method of analysis and program characterization has several benefits. System calls represent key functions / services provided by the operating system. Program obfuscation does not eliminate the need for such services by the application. The way in which systems calls are made is also represented in the sub-pattern. Program patterns can be compared based on their sub-patterns.

3.3 Pre-clustering Processor

Before a data-mining technique is applied, pre-clustering processor aims to classify the semantic patterns of the static analysis into separate clusters with similar sub-patterns. It is called a coarse-grained clustering. The clustered data sets are expected to improve both the speed of data mining and the quality of the produced results. The result of the pre-clustering processor is coarse-grained clusters of sub-patterns that potentially represent similar semantic patterns of the programs. From the coarse-grained clusters, the pre-processor also computes the similarity between pairs of sub-patterns contained in the same coarse-gained cluster. The two outputs in this step, such as the coarse-grained clusters and the similarity scores, are directly used in the next step (Mining Engine). In this section, the method of coarse-grained clustering to produce these two outputs are described more specifically.

3.3.1 Pre-clustering Semantic Sub-patterns

A pre-clustering is made based on the information that is available about the system call represented by the sub-pattern. There are basically four types of sub-patterns as shown in Figure 3. They are all relevant to *call*s using absolute addressing. Relative addressing calls are usually generated during compilation, and result from calls to internal program procedures, rather than functions of the operating system.

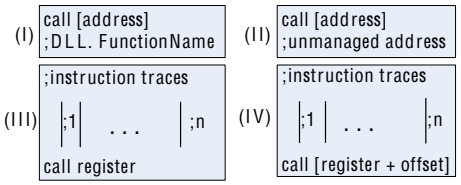


Figure 3. Four Types of Sub-patterns. (a) Direct (known) system calls (b) Direct (unknown) system calls (c) Indirect system calls with call address stored in register (d) Indirect system calls with call address stored in memory as specified. This way usually represents a *method* dispatch.

As shown in Figure 3, there are four types of a sub-pattern. A sub-pattern of type (a) involves a system call that can be clearly identified; the call target has a known DLL name and a known function name. A sub-pattern of type (b) involves a system call whose target address is the only identifiable information. For sub-patterns of type (c) and (d), not even the target address is fully known. This is due to the limitations of static analysis, e.g., the call target is computed at runtime, or the results of instruction execution are incorrectly predicted. For two sub-patterns of type (c) and/or (d), it is not possible to determine accurately by static analysis whether these represent similar program behaviors. The conservative assumption made by this method is that they are dissimilar.

Pre-classification begins by decomposing semantic program patterns into sub-patterns, and categorizing the sub-patterns based on their types in the Figure 3. Sub-patterns of type (a) are clustered together if they contain the same system call with the same DLL name and function name. We assume that two system calls to different DLLs, or to different functions of the same DLL, represent different behaviors. This assumption can be relaxed if the proposed method is provided with information about groups of system calls that represent similar behaviors. For instance, Unicode and non-Unicode versions of Windows library functions may be regarded as implementing the same behaviors. In our prototype implementation, we do not make use of such information, and assume they are different. Therefore, the sub-patterns of type (a) are classified into a set of coarse-grained clusters depending on the names.

Sub-patterns of other types excluding (a) types of Figure 3 are categorized into each coarse-grained clusters. However, because the call target can not be identified in static analysis, we coarsely classify the sub-patterns of the other types into *single cluster* in view of implementation. The final decision to group them into similar semantic patterns rely on the next step (Mining Engine). For instance, for sub-patterns of type (b), system calls casted at different addresses could still represent "similar" semantic patterns. Thus, the mining engine will be decided if the sub-patterns of other types are classified into the similar system calls or similar semantic patterns in the single cluster.

3.3.2 Pair-wise Similarity Measurement

After pre-clustering sub-patterns (SP), the similarities of sub-patterns for each same coarse-grained cluster are measured. The result is a similarity matrix ($S_{SP_xSP_y}$), one per coarse-grained cluster. An entry in the xth row and yth column is a numerical value between 0 and 1 representing the similarity between the xth and the yth sub-patterns in the coarse-grained cluster. In this scheme, a value of 0 represents complete dissimilarity, and a value of 1 represents identical sub-patterns.

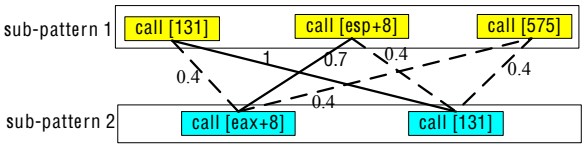


Figure 4. An Example of Sub-patterns Similarity Computation. The similarity score ($S_{SP_xSP_y}$) is $\frac{(1+0.7)}{max(2,3)}=\frac{1.7}{3}=0.57$.

Equation (1) indicates the function that computes the similarity (S) of a pair of sub-patterns [1]. This function uses a maximum weighted matching algorithm to find an optimistic match between the elements of the two sub-patterns (SP_x, SP_y). An element (U) of a sub-pattern corresponds to one maximal instruction trace of the sub-pattern. A heuristic scoring function is used to calculate the similarity of two elements (U_k^x, U_l^y) (i.e. k and l are the number of elements in one sub-pattern). This score function considers several factors including instruction operation type, operand addressing mode and value, and assigns the score based on heuristic estimation of their closeness. Partial credits are given and have different weight for each component's match. This process is implemented as a decision tree algorithm. Details may be found in [31]. Figure 4 illustrates the concept with an example.

[1]Note the formulas to calculate the similarity score are slightly different from MetaAware with different heuristic instruction scoring

Algorithm 1 Mining Engine

```
// Input: A coarse-grain cluster with N sub-patterns,
// C = {SP_1, SP_2, ..., SP_N}
// Regard each sub-pattern as a fine-grain cluster
for i = 1 to N do
    F_i ← {SP_i}; F ← F ∪ F_i
end for
// Perform the hierarchical clustering
// DIST function returns the average distance of Eq.(2)
for two nearest fine-grain clusters, say, F_i and F_j do
    if DIST(F_i, F_j) ≥ Threshold_cutoff then
        break
    end if
    Merge F_i and F_j
end for

// Get popular fine-grain clusters by applying popularity
for all F_i in F do
    if Popularity(F_i) < Threshold_popularity then
        Remove F_i
    end if
end for

// Find the representative common sub-patterns
// |F_i| is the number of sub-patterns in F_i
Pattern_common ← {}
for all F_i in F do
    D_min ← ∞; SP_min ← NULL
    for all SP_x in F_i do
        D = 1/(|F_i|-1) Σ_{y=1}^{|F_i|-1} (1 - S_{SP_x,SP_y})
        if D < D_min then
            D_min ← D
            SP_min ← SP_x
        end if
    end for
    Pattern_common ← Pattern_common ∪ {SP_min}
end for
```

$$S_{SP_xSP_y} = \frac{\sum_{\langle U_k^x, U_l^y\rangle \in W} score(U_k^x, U_l^y)}{max(|SP_x|, |SP_y|)} \quad (1)$$

3.4 Mining Engine for Common Behavior Pattern Generation

As the last step, the hierarchical clustering [9], one of well-known data-mining techniques, mines common semantic behaviors from three inputs of the previous steps: the coarse-grained clusters (C) of sub-patterns, the similarity score matrices ($S_{SP_xSP_y}$), and the original malware semantic patterns, as shown in Figure 1. The hierarchical clustering method enables groups of similar data to be discovered over a variety of scales. To decide what scale of clustering is most appropriate in our application, we define a clustering parameter (c), called clustering *cutoff* value.

The mining engine has three stages for hierarchical clustering, as shown in Algorithm 1. In the first stage, it finely re-clusters sub-patterns in each coarse-grained cluster from the results of the pre-clustering processor. The outputs are

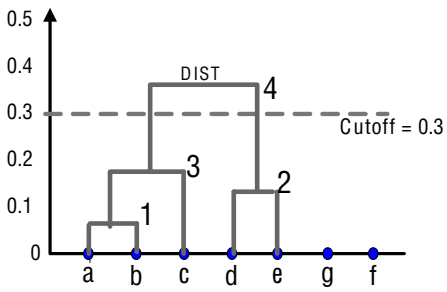


Figure 5. A Dendrogram of Hierarchical Clustering. A coarse-grained cluster (C) contains 7 sub-patterns represented from 'a' to 'f'. From the cluster, two fine-grained clusters (F) are created by *cutoff* = 0.3, such as F_1 = {a, b, c} and F_2 = {d, e}. The height of a sub-tree stands for the average distance (DIST) of that cluster. The number indicates the order of merging. After that, each representative sub-pattern are chosen for F_1 and F_2, respectively, if the popularity exceeds a pre-defined popularity threshold.

called fine-grained clusters. In the second stage, it identifies the most popular fine-grained clusters (i.e., those occurring in the most programs). In the last stage, for each of fine-grained clusters, it chooses representative sub-patterns to stand for all the sub-patterns in that cluster. Therefore, the common semantic behavior as output from the mining engine consists of a set of representative sub-patterns from the popular fine-grained clusters.

In detail, firstly, to find fine-grained clusters (F) from the coarse-grained clusters (C), we use an *average-linkage agglomerative hierarchical clustering* algorithm [9]. The average-linkage clustering algorithm considers all similarities among sub-patterns in C to create fine-grained clusters. In other words, the objective of the average-linkage clustering algorithm is to find groups of similar sub-patterns in C by evaluating all similarities among sub-patterns.

As shown in Algorithm 1, we define the average distance (DIST) between all pairs of sub-patterns between two clusters, for all pairs where one sub-pattern comes from the first cluster, and the other sub-pattern comes from the second cluster in Equation(2). The dissimilarity between two sub-patterns is measured as $(1 - S_{SP_xSP_y})$. In Algorithm 1, the procedure of clustering to find fine-grained clusters repeatedly takes the closest two clusters and then merges them into a single, larger cluster. The example of the procedure is shown in Figure 5. This iterative fine-grained clustering stops when the average distance (DIST) is greater than a pre-defined cutoff value (c) or there is only one cluster remaining. Accordingly, several fine-grained clusters for each coarse-grained cluster are created. However, as shown in Figure 5, the singleton clusters (e.g. 'g' and 'f') consisting of only single sub-patterns that were not close enough to cluster with any other sub-patterns, are considered not to represent the behavior of multiple programs. Such singleton clusters are discarded from further consideration. Therefore, the result constitutes the fined-grained clusters remaining except for such singleton clusters.

$$DIST(F_i, F_j) = \frac{1}{|F_i||F_j|} \sum_{SP_x \in F_i} \sum_{SP_y \in F_j} (1 - S_{SP_xSP_y}) \quad (2)$$

Secondly, to create the common semantic behavior from the fine-grained clusters (F), we need to define another threshold, popularity which can determine the popularity of each cluster (F). The popularity of the cluster (F) is the number of programs in the set for which there is at least one sub-pattern in the cluster (F), divided by the number of programs in the set. For example, if there are 5 programs being processed, and the cluster (F) contains sub-patterns from the first, second, and fourth programs, the popularity of the cluster (F) is 3/5 (.6). After fine-grained clustering, each cluster (F) has its own popularity. Accordingly, we select the most popular fine-grained clusters which are greater than the pre-defined popularity threshold (p), as shown in Algorithm 1.

Lastly, each of the remaining fined-grained clusters choose an representative sub-pattern which has the minimum average distance (D) to all other sub-patterns in the same fine-grained cluster as in Algorithm 1. The common semantic behavior is constructed from representative sub-patterns in the most popular fine-grained clusters. Therefore, the set of exemplar sub-patterns is used as the common semantic behavior that characterizes some of the critical, common functions of a set of the given malware instances.

3.5 Bot Detection

The generated common semantic behavior can be directly used to detect new bot instances. The detection requires a *pattern matching* to compute a matching score (MS) between the semantic patterns of the new instance and the common semantic patterns of the bot malware class. If the matching score exceeds a pre-defined *decision threshold* (d), the new suspicious program is an instance of this bot malware class. Otherwise, the suspect program is regarded not being a member of this malware class. In detail, the method for computing the matching score of the two semantic patterns is derived from a maximum weighted matching algorithm in [32]. However, in this current work, the similarity is defined to be the sum of the weights from the maximum weight matching, divided by the number of sub-patterns from the generated common semantic behaviors. For instance, suppose the pattern of a suspected malware instance consists of four sub-patterns, the pattern derived for the malware class consists of three sub-patterns, and the sum of the weights from a maximum weighted matching of

the two is .6+.7+.9 = 2.2. Then the matching score of the two patterns is 2.2 / 3 = .73. Therefore, if the matching score is greater than the *decision threshold*, the new suspicious instance is identified as the new member of the bot malware class.

4. Empirical Evaluation

In this section, the experimental results are shown to demonstrate the effectiveness of our approach for bot detection. A set of experiments on real-world bot malware and benign executables were performed to evaluate the proposed approach according to various views.

Data Collection. A total of 110 bot programs were obtained from different sources. Among 110 bots, 65 bots come from two trusted sites: the Cyber-TA project website [1] and the Johns Hopkins University bot research project [23]. These executables were successfully disassembled, and had non-empty program patterns, which mean that they contain more than one sub-pattern. We discarded some binaries which are suspected of using encryption or self-decrypting code due to the limitation of static analysis [2]. These programs were detected and labeled as malicious bot programs using Symantec anti-virus software [2], such as `agobot spybot`, `ircbot`, `ircbot.gen`, `linkbot`, `gobot`, etc, as mentioned in [4].

The remaining 45 bots were collected from the public sites[3]. These executables from the public repositories were first selected if their file names include "bot" text with PE format. We verify the relevance of these 45 bots to the 65 bots downloaded from the trusted sites in order to qualify them for the detection capability evaluation. We used the MetaAware tool to calculate all similarities of the pairs, one being from the set of 45 bots and the other being from the set of 65 bots, and their similarities are not 0. These 45 bots also have non-empty patterns.

Out of the 110 programs, two different test datasets are created. The first dataset (DATASET_1) includes randomly-chosen 50 programs from the two trusted source. The chosen 50 bot samples were used to derive common semantic behavior. All of the bot samples (60 bots) excluding the 50 bots are classified in the second dataset (DATASET_2).

For detection rates, the resulting common semantic behavior was evaluated to validate the quality with the first and the second datasets. In other words, the common semantic behavior was matched with the semantic patterns of DATASET_1 and DATASET_2. In the case of DATASET_1, it is intended to show how the generated common semantic behavior was mined well. With DATASET_2, the common semantic behavior was evaluated to demonstrate a detection capability after estimating the best clustering parameter from the result of the first dataset. Lastly, the common semantic behavior was also matched with the semantic patterns of 313 benign programs to evaluate a false positive rate. These 313 non-bot programs, called DATASET_3, are collected from directory `C:\windows\system32` on Windows XP on the test machine.

Quality of Generic Patterns. As shown in Figure 6, after generating the common semantic behavior from DATASET_1, the detection rate was evaluated with the first dataset according to various values of clustering parameters. In addition, the false positive rate with the resulting common semantic behavior is evaluated with DATASET_3, as shown in Figure 7. For each of the clustering parameters, such as cutoff (c), popularity (p), decision threshold (d), we evaluate the generated common semantic behavior according to different values, from 0.1 to 0.9 with 0.1 interval. However, due to space limitations, only the results for the cases where c = 0.1 or c = 0.3, and $p \in \{0.2, 0.4, 0.6, 0.8\}$ are plotted in these figures. As a result, when c = 0.1 or 0.2, p = 0.2 or 0.4, and d = 0.5, the resulting common semantic behavior achieves more than 95.2% detection rates and less than 2.2% false positive rates. In addition, the case of c = 0.2, p = 0.6, and d = 0.5 shows a detection rate of 92.9% and a false positive rate of 0.32%.

From these experiments, the quality of the common semantic behavior is demonstrated that our approach can detect bots with a small set of semantic patterns mined from the large dataset. Based on the detection rates and the false positive rates in Figure 6 and Figure 7, we can choose the best values of clustering parameters which result in a false positive rate of less than 5% and a detection rate of more than 95%. In detail, the qualified results from these experiments are shown in Table 9. The locally optimal results, and their parameter configurations, are shown as the shadowed entries in these tables. These optimal results will not be inferior to any others; two cases were chosen for the next experiment with DATASET_2. The first case is c = 0.2, p = 0.2, and d = 0.4, for the detection rate and false positive rate 97.6% and 1.6%, respectively. With the second case, the detection and false positive rates were 95.2% and 0.06%, for c = 0.2, p = 0.2, and d = 0.5.

Detection Capability. Based on the common semantic behavior generated from DATASET_1, the detection capability is evaluated with DATASET_2, which is totally different binaries from DATASET_1. As shown in Figure 10, our approach still shows more than 90% detection rate for the configuration of clustering parameters in the first experiments. In detail, for $d = 0.4$ with c = 0.1 or 0.2 and p = 0.2, the detection rate of 98% was achieved from the complete different dataset. With the best values of the clustering parameters from the first experiments, more than 90%

[2] A future work to address this limitation is to combine dynamic analysis to improve the coverage of code disassembly.

[3] http://vx.netlux.org, http://securitydot.net

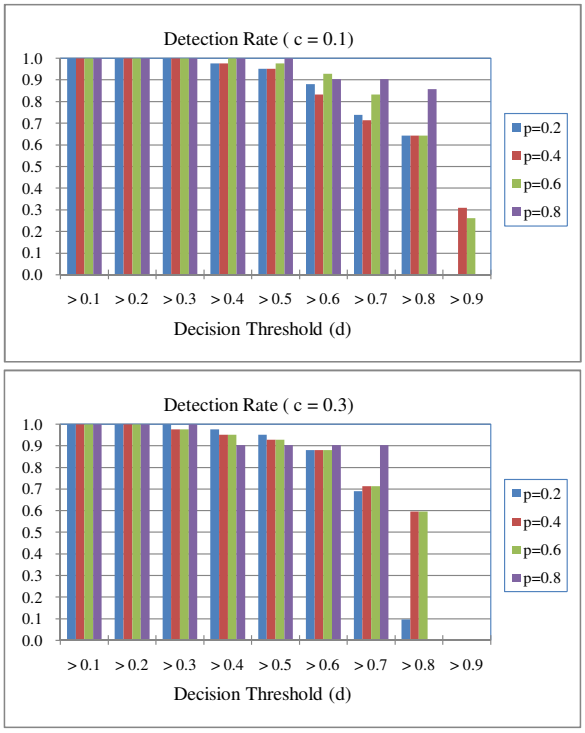


Figure 6. Detection Rate with DATASET_1.(c = cut-off, p = popularity, d = decision threshold)

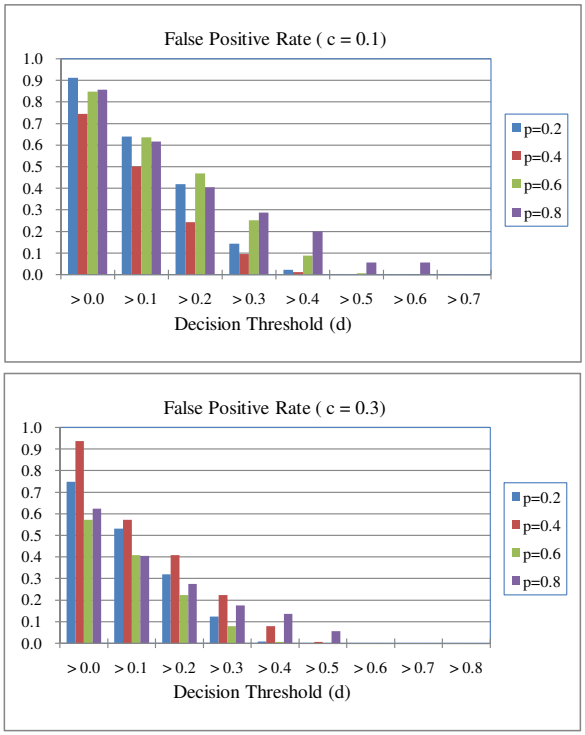
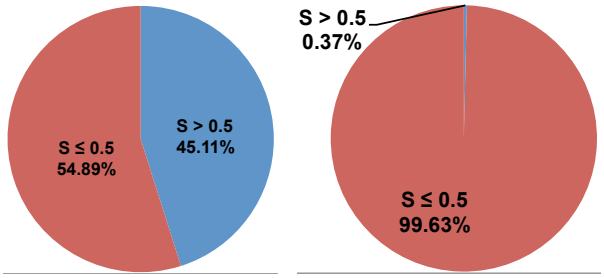


Figure 7. False Positive Rate with DATASET_3.(c = cutoff, p = popularity, d = decision threshold)

detection rate was achieved with DATASET_3. Moreover, from this experimental results, the number of the bot samples for generating common semantic behavior was considered to be a sufficient number of programs to identify common semantic behavior, but small enough to avoid performance problems during derivation of the common semantic behavior. However, note that the selection of test set may represent a wide variety of malware behaviors, sources, and coding styles. The only thing known to be common about them is that they represent bot programs. This is therefore considered to be a challenging test set.

Figure 4 shows distribution of similarity for sub-patterns of bot programs or sub-patterns of benign programs by comparing with the common semantic patterns. For this purpose, we selected 10 bot programs in the data set and 10 benign programs. While only 0.37% of sub-patterns in the benign programs showed more than 0.5 similarity (S), 45.11% of sub-patterns in the bot programs exhibited more than 0.5 similarity with the common semantic patterns. Consequently, this obvious distinction leads AntiBot to detect any variant of bots.

Selection of Clustering Parameters. Based on detection rates and false positives, we can generally suggest a standard to choose good values of clustering parameters



(a) Bot Programs (b) Benign Programs

Figure 8. Distribution of Similarity (S) for Sub-patterns comparing with the Common Semantic Patterns

through the extensive experiments. The cutoff (c) values should be less than 0.5 because the similarity is decreased according to the increase of cutoff. In other words, the high cutoff values indicate the high dissimilarity among clusters. Our experiments showed the results of c = 0.4, or c = 0.5 were not far from the results of c = 0.3. In the case of popularity (p), the high values of the popularity with the high vales of the cutoff increase a false positive rate. Ac-

Decision (d)	Cutoff (c)	Popularity (p)	Dt (%)	Fp (%)
0.4	0.1	0.2	97.62	6.39
0.4	0.1	0.4	97.62	9.58
0.4	0.1	0.6	100.00	9.27
0.4	**0.2**	**0.2**	**97.62**	**1.60**
0.4	0.2	0.4	95.24	8.63
0.4	0.2	0.6	97.62	8.95
0.4	0.3	0.4	95.24	7.99
0.4	0.3	0.6	95.24	7.99
0.5	**0.1**	**0.2**	**95.24**	**2.24**
0.5	**0.1**	**0.4**	**95.24**	**1.28**
0.5	0.1	0.6	97.62	8.95
0.5	**0.2**	**0.2**	**95.24**	**0.06**
0.5	**0.2**	**0.4**	**95.24**	**0.64**
0.6	0.2	0.8	95.24	6.39
0.5	**0.3**	**0.2**	**95.24**	**0.96**

Figure 9. Several Qualified Clustering Parameters from DATASET_1. (c = cutoff, p = popularity threshold, d = decision threshold, D_t = detection rate, F_p = false positive rate)

c	p	d > 0.3	d > 0.4	d > 0.5	d > 0.6	d > 0.7
0.1	0.2	1.00	0.98	0.98	0.88	0.42
0.1	0.4	1.00	0.98	0.93	0.86	0.40
0.1	0.6	1.00	1.00	0.95	0.91	0.84
0.1	0.8	1.00	1.00	0.98	0.91	0.86
0.2	0.2	1.00	0.98	0.91	0.86	0.47
0.2	0.4	0.98	0.95	0.91	0.72	0.40
0.2	0.6	1.00	0.93	0.88	0.65	0.40
0.2	0.8	1.00	0.98	0.91	0.91	0.86

Figure 10. Detection Capability with DATASET_2. (c = cutoff, p = popularity threshold, d = decision threshold, The entries in this table indicate detection rate)

cording to the experiments, the popularity (p) should have between 0.2 and 0.6 when c is less than 0.5. The decision threshold (d) should be between 0.4 and 0.7 to achieve high detection rates as well as low false positive rates. For automatically setting these parameter values, the proposed method can stop itself at the best performance defined, such as more than 95% detection rates and less than 5% false positive rates, as shown in Figure 9. With the resulting parameter values, the mined common patterns demonstrated around 90% detection rates with different dataset, DATASET_2. In general, the best performance achieve at $c \leq 0.4$, $0.2 \leq p \leq 0.8$, and $d = 0.5$ for the training set.

Comparison with MetaAware. Our approach in this paper focuses on generalizing the semantic pattern generation and matching approach. It aims to use a single *common* malware behavior pattern to detect an entire category of bot malware as well as those are from a different category. Therefore, the technique presented in this paper can potentially result in a substantial reduction in semantic pattern population for the detection of known and new malware instances. We performed an experiment to quantify this amount.

To compare our approach with the previous work [32], we highlight the difference between the two techniques through another experiments. MetaAware [32] addresses the problem of how to summarize and compare program semantics between *a pair* of programs (or executables). The technique can serve as a basis for the detection of metamorphic malware which has equivalent or updated functionalities. This technique works by using one malware instance's pattern to detect the variants of that malware. However, this technique is not intended to detect other malware instances from a totally different malware category.

For the experiment, we randomly select a set of 25 training programs from DATASET_1. The 25 training programs are used to derive the common semantic behavior, and their common semantic behavior was evaluated with DATASET_1. From the experiment, the detection rate and the false positive rate was 78% and 0.32%, respectively, for c = 0.2, p = 0.4, and d = 0.6. For comparison, the number of semantic patterns are measured to achieve the same detection rate using the same data set under the same testing condition (i.e., d = 0.6). Based on MetaAware, the minimum number of semantic patterns were calculated from the 25 training programs when the similarity score between the common behavior pattern of the 25 training set and the patterns of DATASET_1 was greater than 0.6. As a result, using the *five* best semantic patterns, the target detection rate was achieved. To summarize, for this experiment, the technique presented in this paper resulted in an 80% reduction (from 5 to 1) in the number of semantic patterns that were needed to achieve a target detection rate.

5. Limitations

There are two limitations to the proposed work. The first limitation stems from the limitations of static analysis. Malware self-defense techniques can challenge and thwart static analysis. There are several such techniques, as pointed out in [27]. Some of them, including using packed malware binaries and blocking access to files, were encountered in our experimental evaluation. Other program obfuscation techniques, like delicate system call obfuscation, are not handled by the static analyzer. For this problem, solutions have been proposed by A. Lakhotia et al. [17]. A more sophisticated technique would integrate these approaches to strengthen malware analysis and defense.

The second limitation comes from the data-mining approach, whose accuracy depends on the training data set to

derive the common behaviors. It is an undecidable problem to determine the semantic equivalence of two programs. If the training set could insert unnecessary, atypical behaviors in all of them, the resulting common behaviors would include such behavior, which would not be found in other malware instances. The result would not invalidate our approach, but would degrade its performance. However, small errors can be filtered by the mining technique, and the careful selection from the large training datasets can avoid such situation, as shown in our experiments.

Data set selection in general is an important problem for data mining. A popular solution to is based on the use of *genetic algorithms* [11] with domain specific individual selection functions to improve the overall fitness of the population. In our case, a special consideration arises if the data set consists of malicious binaries that are developed and compiled using the same tool set or development environment. Such binaries may contain significant common behavior not related to their malicious function, due simply to having been generated using the same tools. Such behavior may also be found in benign programs coded and compiled in the same environment, causing an increase in the false positive rate of the proposed method. A possible solution is to remove or exclude behaviors found to be shared in common between benign and malicious programs from the common behavior pattern for the malicious programs.

6. Related Work

Autonomous and automatic malicious bot programs have threatened current networks, however, several method to detect them were proposed [19, 28]. Signature based malware detection methods have been the major anti-malware methods used in industry. However they are frequently criticized for being easily bypassed by advanced attacks such as polymorphic and metamorphic malware [5, 29, 7, 20]. More, the signatures are poor at providing forward detection to new malware instances. Regardless to whether a malware signature is generated manually or automatically, the signature repository requires constant maintenance for updates.

Heuristic behavior based detection methods [24, 7, 15, 16], which detect specific malicious behaviors such as self-unpacking, have been more robust against advanced attacks. They detect entire families of malware exhibiting that specific behavior. However, to derive such malicious behavior specifications manually can be a slow, difficult process.

Data mining approaches [18, 30, 13, 26] have been frequently used to build behavior models for intrusion detection purposes. These methods are applied to system calls or network data to learn how to detect new intrusions. These methods constructed benign behavior models and detect deviating behavior via run-time monitoring. A recent paper [6] developed a novel mining technique to specify malicious program behaviors by mining differences of execution traces between a malware sample and a set of benign programs. A general limitation to this dynamic execution based approach is its difficulty of simulating the malicious execution environment to obtain high coverage of malicious execution traces. The malware detection using a common malware behavior pattern proposed in this paper is, in contrast, a static pattern matching process.

The work by Schultz et al. [25] is closest to our work. It built a framework that uses three data-mining algorithms (i.e., RIPPER, Naive Bayes, and Multi-Naive Bayes) to train multiple classifiers on a set of malicious and benign executables to detect new malware instances. The training binaries are statically analyzed to extract the properties of DLLs used, fixed strings, and byte sequences in the binaries. According to [25], the most accurate algorithm for predicting a program to be malicious was the Multi-Bayes algorithm. This is based on the probability or frequency calculation of program byte sequences and strings.

However, to deal with advanced attacks that deliberately utilize program obfuscation techniques, program semantics should be examined [5, 29, 7]. There are methods that can manipulate code byte sequences distribution to evade intrusion detection based on frequency computations [3, 20, 21, 10].

Our approach is different from [25] by using a different feature extraction. Our approach considers the program semantics and is based on static analysis to extract and compare the way that a program makes system calls. Previous evaluation has shown effectiveness of the method's resilience to program obfuscation [32]. Moreover, the mined common behavior patterns are a byproduct that can be utilized in assisting malware analysis.

7. Conclusion

In this paper, we presented AntiBot, a new host-based bot detection, to identify bots or their new mutants. It is achieved by automated generation of common semantic behavior from a large set of bot malware. This idea start from the fact that most bot programs are derivatives from other bot family. Because of it, they use the same or similar functions to perform their objectives [28]. The proposed approach newly combines static analysis with a data-mining technique. The technique is used to identify the most promising patterns to represent the whole behaviors from the given bot class. From the experiments, AntiBot showed a high detection rate with a small set of patterns. Moreover, this approach is more robust to a type of data attack that deliberately inserts junk code (not important to the exploit's intended function) than other malware detectors using semantic patterns.

References

[1] Cyber-Threat Analytics (Cyber-TA) Project. http://www.cyber-ta.org/releases/malware/.

[2] Symantec anti-virus software. http://www.symantec.com.

[3] The CLET polymorphism engine. http://www.phrack.org.

[4] P. Barford and V. Yegneswaran. An inside look at botnets. pages 171–191, 2006.

[5] M. Christodorescu and S. Jha. Testing malware detectors. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 34–44, 2004.

[6] M. Christodorescu, S. Jha, and C. Kruegel. Mining Specifications of Malicious Behavior. In *Proceedings of the the 6th ESEC/FSE*, pages 5–14, September 2007.

[7] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-Aware Malware Detection. In *Proceedings of 2005 IEEE Symposium on Security and Privacy (S&P'05)*, pages 32–46, May 2005.

[8] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23th Annual Computer Security Applications Conference (ACSAC'07)*, pages 325–339, December 2007.

[9] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[10] P. Fogla and W. Lee. Evading Network Anomaly Detection systems: Formal Reasoning and Practical Techniques. In *Proceedings of the 13th ACM Conference on Computer and Communications Security(CCS'06)*, pages 59–68, 2006.

[11] S. Forrest. Genetic algorithms. *ACM Computing Surveys*, 28(1):77–80, 1996.

[12] F. C. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *ESORICS*, pages 319–335, 2005.

[13] J. T. Giffin, S. Jha, and B. P. Miller. Efficient Context-Sensitive Intrusion Detection. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, 2004.

[14] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and detecting fast-flux service networks. In *15th Network and Distributed System Security Symposium (NDSS)*, Feb 2008.

[15] E. Kirda and C. Kruegel. Behavior-based Spyware Detection. In *Proceedings of the 15th USENIX Security Symposium (Security'06)*, pages 273–288, Auguest 2006.

[16] C. Kruegel, W. Robertson, and G. Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 91–100, December 2004.

[17] A. Lakhotia, E. U. Kumar, and M. Venable. A Method for Detecting Obfuscated Calls in Malicious Binaries. *IEEE Transactions on Software Engineering*, 31(11):955–968, 2005.

[18] W. Lee and S. J. Stolfo. Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th conference on USENIX Security Symposium (Security'98)*, January 1998.

[19] L. Liu, S. Chen, G. Yan, and Z. Zhang. Bottracer: Execution-based bot-like malware detection. In *Proceedings of the 11th Information Security Conference(ISC 08)*, Taipei, Taiwan, September 2008.

[20] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting Signature Learning By Training Maliciously. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, September 2006.

[21] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. I. Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection. In *Proceedings of 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 17–31, May 2006.

[22] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52. ACM, 2006.

[23] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC '06)*, pages 41–52, October 2006.

[24] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *Proceedings of the 22th Annual Computer Security Applications Conference (ACSAC'06)*, December 2006.

[25] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P'01)*, page 38, May 2001.

[26] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors. In *Proceedings of 2001 IEEE Symposium on Security and Privacy (S&P'01)*, pages 144–155, May 2001.

[27] A. Shevchenko. The Evolution of Self-Defense Technologies in Malware. July 2007. http://www.net-security.org/article.php?id=1028.

[28] E. Stinson and J. C. Mitchell. Characterizing bots' remote control behavior. In *The 4th GI International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment(DIMVA)*, pages 89–108, Lucerne, Switzerland, 2007.

[29] G. Vigna, W. K. Robertson, and D. Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the 11th ACM Conference on Computer and Communications Security(CCS'04)*, pages 21–30, October 2004.

[30] D. Wagner and D. Dean. Intrusion detection via static analysis. In *Proceedings of 2001 IEEE Symposium on Security and Privacy (S&P'01)*, pages 156–169, May 2001.

[31] Q. Zhang. *Polymorphic and Metamorphic Malware Detection*. PhD thesis, North Carolina State University, Raleigh, North Carolina, 2008.

[32] Q. Zhang and D. Reeves. MetaAware: Identifying Metamorphic Malware. In *Proceedings of the 23th Annual Computer Security Applications Conference (ACSAC'07)*, pages 411–420, December 2007.