

# Automated Classification and Analysis of Internet Malware

Michael Bailey<sup>1</sup>, Jon Oberheide<sup>1</sup>, Jon Andersen<sup>1</sup>, Z. Morley Mao<sup>1</sup>,  
Farnam Jahanian<sup>1,2</sup>, and Jose Nazario<sup>2</sup>

<sup>1</sup> Electrical Engineering and Computer Science Department  
University of Michigan  
{mibailey, jonojono, janderse, zmao, farnam}@umich.edu  
<sup>2</sup> Arbor Networks  
{farnam, jose}@arbor.net

**Abstract.** Numerous attacks, such as worms, phishing, and botnets, threaten the availability of the Internet, the integrity of its hosts, and the privacy of its users. A core element of defense against these attacks is anti-virus (AV) software—a service that detects, removes, and characterizes these threats. The ability of these products to successfully characterize these threats has far-reaching effects—from facilitating sharing across organizations, to detecting the emergence of new threats, and assessing risk in quarantine and cleanup. In this paper, we examine the ability of existing host-based anti-virus products to provide semantically meaningful information about the malicious software and tools (or malware) used by attackers. Using a large, recent collection of malware that spans a variety of attack vectors (e.g., spyware, worms, spam), we show that different AV products characterize malware in ways that are inconsistent across AV products, incomplete across malware, and that fail to be concise in their semantics. To address these limitations, we propose a new classification technique that describes malware behavior in terms of system state changes (e.g., files written, processes created) rather than in sequences or patterns of system calls. To address the sheer volume of malware and diversity of its behavior, we provide a method for automatically categorizing these profiles of malware into groups that reflect similar classes of behaviors and demonstrate how behavior-based clustering provides a more direct and effective way of classifying and analyzing Internet malware.

## 1 Introduction

Many of the most visible and serious problems facing the Internet today depend on a vast ecosystem of malicious software and tools. Spam, phishing, denial of service attacks, botnets, and worms largely depend on some form of malicious code, commonly referred to as *malware*. Malware is often used to infect the computers of unsuspecting victims by exploiting software vulnerabilities or tricking users into running malicious code. Understanding this process and how attackers

use the backdoors, key loggers, password stealers, and other malware functions is becoming an increasingly difficult and important problem.

Unfortunately, the complexity of modern malware is making this problem more difficult. For example, Agobot [3], has been observed to have more than 580 variants since its initial release in 2002. Modern Agobot variants have the ability to perform DoS attacks, steal bank passwords and account details, propagate over the network using a diverse set of remote exploits, use polymorphism to evade detection and disassembly, and even patch vulnerabilities and remove competing malware from an infected system [3]. Making the problem even more challenging is the increase in the number and diversity of Internet malware. A recent Microsoft survey found more than 43,000 new variants of backdoor trojans and bots during the first half of 2006 [20]. Automated and robust approaches to understanding malware are required to successfully stem the tide.

Previous efforts to automatically classify and analyze malware (e.g., AV, IDS) focused primarily on content-based signatures. Unfortunately, content-based signatures are inherently susceptible to inaccuracies due to polymorphic and metamorphic techniques. In addition, the signatures used by these systems often focus on a specific exploit behavior—an approach increasingly complicated by the emergence of multi-vector attacks. As a result, IDS and AV products characterize malware in ways that are inconsistent across products, incomplete across malware, and that fail to be concise in their semantics. This creates an environment in which defenders are limited in their ability to share intelligence across organizations, to detect the emergence of new threats, and to assess risk in quarantine and cleanup of infections.

To address the limitations of existing automated classification and analysis tools, we have developed and evaluated a dynamic analysis approach, based on the execution of malware in virtualized environments and the causal tracing of the operating system objects created due to malware's execution. The reduced collection of these user-visible system state changes (e.g., files written, processes created) is used to create a fingerprint of the malware's behavior. These fingerprints are more invariant and directly useful than abstract code sequences representing programmatic behavior and can be directly used in assessing the potential damage incurred, enabling detection and classification of new threats, and assisting in the risk assessment of these threats in mitigation and clean up. To address the sheer volume of malware and the diversity of its behavior, we provide a method for automatically categorizing these malware profiles into groups that reflect similar classes of behaviors. These methods are thoroughly evaluated in the context of a malware dataset that is large, recent, and diverse in the set of attack vectors it represents (e.g., spam, worms, bots, spyware).

This paper is organized as follows: Section 2 describes the shortcomings of existing AV software and enumerates requirements for effective malware classification. We present our behavior-based fingerprint extraction and fingerprint clustering algorithm in Section 3. Our detailed evaluation is shown in Section 4. We present existing work in Section 5, offer limitations and future directions in Section 6, and conclude in Section 7.

## 2 Anti-Virus Clustering of Malware

Host-based AV systems detect and remove malicious threats from end systems. As a normal part of this process, these AV programs provide a description for the malware they detected. The ability of these products to successfully characterize these threats has far-reaching effects—from facilitating sharing across organizations, to detecting the emergence of new threats, and assessing risk in quarantine and cleanup. However, for this information to be effective, the descriptions provided by these systems must be meaningful. In this section, we evaluate the ability of host-based AV to provide meaningful intelligence on Internet malware.

### 2.1 Understanding Anti-Virus Malware Labeling

In order to accurately characterize the ability of AV to provide meaningful labels for malware, we first need to acquire representative datasets. In this paper, we use three datasets from two sources, as shown in Table 1. One dataset, *legacy*, is taken from a network security community malware collection and consists of randomly sampled binaries from those posted to the community’s FTP server in 2004. In addition, we use a large, recent six-month collection of malware and a six-week subset of that collection at the beginning of the dataset collection period. The *small* and *large* datasets are a part of the Arbor Malware Library (AML). Created by Arbor Networks, Inc. [1], the AML consists of binaries collected by a variety of techniques including Web page crawling [28], spam traps [26], and honeypot-based vulnerability emulation [2]. **Since each of these methods collects binaries that are installed on the target system without the user’s permission, the binaries collected are highly likely to be malicious.** Almost 3,700 unique binaries were collected over a six-month period in late 2006 and early 2007.

**Table 1.** The datasets used in this paper: A large collection of legacy binaries from 2004, a small six-week collection from 2006, and a large six-month collection of malware from 2006/2007. The number of unique labels provided by five AV systems is listed for each dataset.

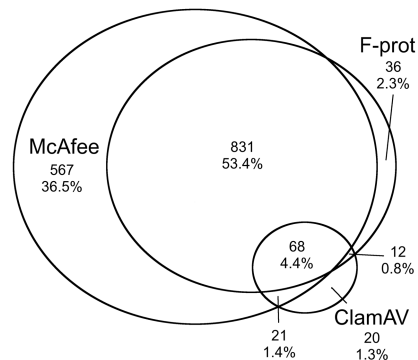
Dataset Name	Date Collected	Number of Unique MD5s	Number of Unique Labels				
			McAfee	F-Prot	ClamAV	Trend	Symantec
<i>legacy</i>	01 Jan 2004 - 31 Dec 2004	3,637	116	1216	590	416	57
<i>small</i>	03 Sep 2006 - 22 Oct 2006	893	112	379	253	246	90
<i>large</i>	03 Sep 2006 - 18 Mar 2007	3,698	310	1,544	1,102	2,035	50

After collecting the binaries, we analyzed them using the AV scanners shown in Table 2. Each of the scanners was the most recent available from each vendor at the time of the analysis. The virus definitions and engines were updated uniformly on November 20th, 2006, and then again on March 31st, 2007. Note that the first update occurred more than a year after the *legacy* collection ended and one month after the end of the *small* set collection. The second update was 13 days after the end of the *large* set collection.

**Table 2.** Anti-virus software, vendors, versions, and signature files used in this paper. The *small* and *legacy* datasets were evaluated with a version of these systems in November of 2006 and both *small* and *large* were evaluated again with a version of these systems in March of 2007.

Label	Software	Vendor	Version	Signature File
McAfee	Virus Scan	McAfee, Inc.	v4900	20 Nov 2006
			v5100	31 Mar 2007
F-Prot	F-Prot Anti-virus	FRISK Software International	4.6.6	20 Nov 2006
			6.0.6.3	31 Mar 2007
ClamAV	Clam Anti-virus	Tomasz Kojm and the ClamAV Team	0.88.6	20 Nov 2006
			0.90.1	31 Mar 2007
Trend	PC-cillin Internet Security 2007	Trend Micro, Inc.	8.000-1001	20 Nov 2006
			8.32.1003	31 Mar 2007
Symantec	Norton Anti-virus 2007	Symantec Corporation	14.0.0.89	20 Nov 2006
			14.0.3.3	31 Mar 2007

AV systems rarely use the exact same labels for a threat, and users of these systems have come to expect simple naming differences (e.g., W32Lovsan.worm.a versus Lovsan versus WORM\_MSBLAST.A) across vendors. It has always been assumed, however, that there existed a simple mapping from one system’s name space to another, and recently investigators have begun creating projects to unify these name spaces [4]. Unfortunately, the task appears daunting. Consider, for example, the number of unique labels created by various systems. The result in Table 1 is striking—there is a substantial difference in the number of unique labels created by each AV system. While one might expect small differences, it is clear that AV vendors disagree not only on what to label a piece of malware, but also on how many unique labels exist for malware in general.



**Fig. 1.** A Venn diagram of malware labeled as SDBot variants by three AV products in the *legacy* dataset. The classification of SDBot is ambiguous.

One simple explanation for these differences in the number of labels is that some of these AV systems provide a finer level of detail into the threat landscape than the others. For example, the greater number of unique labels in Table 1 for F-Prot may be the result of F-Prot’s ability to more effectively differentiate small variations in a family of malware. To investigate this conjecture, we examined the

labels of the *legacy* dataset produced by the AV systems and, using a collection of simple heuristics for the labels, we created a pool of malware classified by F-Prot, McAfee, and ClamAV as SDBot [19]. We then examined the percentage of time each of the three AV systems classified these malware samples as part of the same family. The result of this analysis can be seen in Figure 1. Each AV classifies a number of samples as SDBot, yet the intersection of these different SDBot families is not clean, since there are many samples that are classified as SDBot by one AV and as something else by the others. It is clear that these differences go beyond simple differences in labeling—anti-virus products assign distinct semantics to differing pieces of malware.

## 2.2 Properties of a Labeling System

Our previous analysis has provided a great deal of evidence indicating that labeling across AV systems does not operate in a way that is useful to researchers, operators, and end users. Before we evaluate these systems any further, it is important to precisely define the properties an ideal labeling system should have. We have identified three key design goals for such a labeling system:

- **Consistency.** Identical items must and similar items should be assigned the same label.
- **Completeness.** A label should be generated for as many items as possible.
- **Conciseness.** The labels should be sufficient in number to reflect the unique properties of interest, while avoiding superfluous labels.

## 2.3 Limitations of Anti-Virus

Having identified consistency, completeness, and conciseness as the design goals of a labeling system, we are now prepared to investigate the ability of AV systems to meet these goals.

**Table 3.** The percentage of time two binaries classified as the same by one AV are classified the same by other AV systems. Malware is inconsistently classified across AV vendors.

	<i>legacy</i>					<i>small</i>				
	McAfee	F-Prot	ClamAV	Trend	Symantec	McAfee	F-Prot	ClamAV	Trend	Symantec
McAfee	100	13	27	39	59	100	25	54	38	17
F-Prot	50	100	96	41	61	45	100	57	35	18
ClamAV	62	57	100	34	68	39	23	100	32	13
Trend	67	18	25	100	55	45	23	52	100	16
Symantec	27	7	13	14	100	42	25	46	33	100

**Consistency.** To investigate consistency, we grouped malware into categories based on the labels provided by AV vendors. For each pair of distinct malware labeled as the same by a particular system, we compared the percentage of time the same pair was classified by other AV systems as the same. For example, two binaries in our *legacy* dataset with different MD5 checksums were labeled as

**Table 4.** The percentage of malware samples detected across datasets and AV vendors. AV does not provide a complete categorization of the datasets.

Dataset Name	AV Updated	Percentage of Malware Samples Detected				
		McAfee	F-Prot	ClamAV	Trend	Symantec
<i>legacy</i>	20 Nov 2006	100	99.8	94.8	93.73	97.4
<i>small</i>	20 Nov 2006	48.7	61.0	38.4	54.0	76.9
<i>small</i>	31 Mar 2007	67.4	68.0	55.5	86.8	52.4
<i>large</i>	31 Mar 2007	54.6	76.4	60.1	80.0	51.5

W32-Blaster-worm-a by McAfee. These two binaries were labeled consistently by F-Prot (both as msblast), and Trend (both as msblast), but inconsistently by Symantec (one blaster and one not detected) and ClamAV (one blaster, one dcom.exploit). We then selected each system in turn and used its classification as the base. For example, Table 3 shows that malware classified by McAfee as the same was only classified as the same by F-Prot 13% of the time. However, malware classified by F-Prot as the same was only classified as the same by McAfee 50% of the time. Not only do AV systems place malware into different categories, these categories don't hold the same meaning across systems.

**Completeness.** As discussed earlier, the design goal for completeness is to provide a label for each and every item to be classified. For each of the datasets and AV systems, we examined the percentage of time the AV systems detected a given piece of malware (and hence provided a label). A small percentage of malware samples are still undetected a year after the collection of the *legacy* datasets (Table 4). The results for more recent samples are even more profound, with almost half the samples undetected in the *small* dataset and one quarter in the *large* dataset. The one quarter undetected for the *large* set is likely an overestimate of the ability of the AV, as many of the binaries labeled at that point were many months old (e.g., compare the improvement over time in the two labeling instances of *small*). Thus, AV systems do not provide a complete labeling system.

**Table 5.** The ways in which various AV products label and group malware. AV labeling schemes vary widely in how concisely they represent the malware they classify.

	<i>legacy</i> (3,637 binaries)		<i>small</i> (893 binaries)	
	Unique Labels	Clusters or Families	Unique Labels	Clusters or Families
McAfee	116	34	122	95
F-Prot	1216	37	379	62
ClamAV	590	41	253	65
Trend	416	46	246	72
Symantec	57	31	90	81

**Conciseness.** Conciseness refers to the ability of the labeling system to provide a label that reflects the important characteristics of the sample without superfluous semantics. In particular, we find that a label that carries either too much or too little meaning has minimal value. To investigate this property, we examined the number and types of labels and groups provided by the AV systems. Table 5 shows the number of unique labels provided by the AV systems

as well as the number of unique families these labels belong to. In this analysis, the family is a generalized label heuristically extracted from the literal string, which contains the portion intended to be human-readable. For example, the literal labels returned by a AV system W32-Sdbot.AC and Sdbot.42, are both in the “sdbot” family. An interesting observation from this table is that these systems vary widely in how concisely they represent malware. Vendors such as Symantec appear to employ a general approach, reducing samples to a small handful of labels and families. On the other extreme, FProt appears to aggressively label new instances, providing thousands of unique labels for malware, but still maintaining a small number of groups or families to which these labels belong.

### 3 Behavior-Based Malware Clustering

As we described in the previous section, any meaningful labeling system must achieve consistency, completeness, and conciseness, and existing approaches, such as those used by anti-virus systems, fail to perform well on these metrics. To address these limitations, we propose an approach based on the actual execution of malware samples and observation of their persistent state changes. These state changes, when taken together, make a behavioral fingerprint, which can then be clustered with other fingerprints to define classes and subclasses of malware that exhibit similar state change behaviors. In this section, we discuss our definition and generation of these behavioral fingerprints and the techniques for clustering them.

#### 3.1 Defining and Generating Malware Behaviors

Previous work in behavioral signatures has been based at the abstraction level of low-level system events, such as individual system calls. In our system, the intent is to capture what the malware actually does on the system. Such information is more invariant and directly useful to assess the potential damage incurred. Individual system calls may be at a level that is too low for abstracting semantically meaningful information: a higher abstraction level is needed to effectively describe the behavior of malware. *We define the behavior of malware in terms of non-transient state changes that the malware causes on the system.* State changes are a higher level abstraction than individual system calls, and they avoid many common obfuscation techniques that foil static analysis as well as low-level signatures, such as encrypted binaries and non-deterministic event ordering. In particular, we extract simple descriptions of state changes from the raw event logs obtained from malware execution. Spawned process names, modified registry keys, modified file names, and network connection attempts are extracted from the logs and the list of such state changes becomes a behavioral profile of a sample of malware.

Observing the malware behavior requires actually executing the binaries. We execute each binary individually inside a virtual machine [27] with Windows XP installed. The virtual machine is partially firewalled so that the external

impact of any immediate attack behaviors (e.g., scanning, DDoS, and spam) is minimized during the limited execution period. The system events are captured and exported to an external server using the Backtracker system [12]. In addition to exporting system events, the Backtracker system provides a means of building causal dependency graphs of these events. The benefit of this approach is that we can validate that the changes we observe are a direct result of the malware, and not of some normal system operation.

### 3.2 Clustering of Malware

While the choice of abstraction and generation of behaviors provides useful information to users, operators, and security personnel, the sheer volume of malware makes manual analysis of each new malware intractable. Our malware source observed 3,700 samples in a six-month period—over 20 new pieces per day. Each generated fingerprint, in turn, can exhibit many thousands of individual state changes (e.g., infecting every .exe on a Windows host). For example, consider the tiny subset of malware in table 6. The 10 distinct pieces of malware generate from 10 to 66 different behaviors with a variety of different labels, including disjoint families, variants, and undetected malware. While some items obviously belong together in spite of their differences (e.g., C and D), even the composition of labels across AV systems can not provide a complete grouping of the malware. Obviously, for these new behavioral fingerprints to be effective, similar behaviors need to be grouped and appropriate meanings assigned.

**Table 6.** Ten unique malware samples. For each sample, the number of process, file, registry, and network behaviors observed and the classifications given by various AV vendors are listed.

Label	MD5	P/F/R/N	McAfee	Trend
A	71b99714cddd66181e54194c44ba59df	8/13/27/0	Not detected	W32/Backdoor.QWO
B	be5f889d12fe608e48be11e883379b7a	8/13/27/0	Not detected	W32/Backdoor.QWO
C	df1cda05aab2d366e626eb25b9cba229	1/1/6/1	W32/Mytob.gen@MM	W32/IRCBot-based!Maximus
D	5bf169aba400f20cbe1b237741eff090	1/1/6/2	W32/Mytob.gen@MM	Not detected
E	eef804714ab4f89ac847357f3174aa1d	1/2/8/3	PWS-Banker.gen.i	W32/Bancos.IQK
F	80f64d342fdcc980ae81d7f8456641e	2/11/28/1	IRC/Flood.gen.b	W32/Backdoor.AHJJ
G	12586ef09abc1520c1ba3e998baec457	1/4/3/1	W32/Pate.b	W32/Parite.B
H	ff0f3c170ea69ed266b8690e13daf1a6	1/2/8/1	Not detected	W32/Bancos.IJG
I	36f6008760bd8dc057ddb1cf99c0b4d7	3/22/29/3	IRC/Generic Flooder	IRC/Zapchast.AK@bd
J	c13f3448119220d006e93608c5ba3e58	5/32/28/1	Generic BackDoor.f	W32/VB-Backdoor!Maximus

Our approach to generating meaningful labels is achieved through clustering of the behavioral fingerprints. In the following subsections, we introduce this approach and the various issues associated with effective clustering, including how to compare fingerprints, combine them based on their similarity, and determine which are the most meaningful groups of behaviors.

**Comparing Individual Malware Behaviors.** While examining individual behavioral profiles provides useful information on particular malware samples, our goal is to classify malware and give them meaningful labels. Thus malware samples must be grouped. One way to group the profiles is to create a distance metric that measures the difference between any two profiles, and then use the



**Table 7.** A matrix of the NCD between each of the 10 malware samples in our example

	A	B	C	D	E	F	G	H	I	J
A	0.06	0.07	0.84	0.84	0.82	0.73	0.80	0.82	0.68	0.77
B	0.07	0.06	0.84	0.85	0.82	0.73	0.80	0.82	0.68	0.77
C	0.84	0.84	0.04	0.22	0.45	0.77	0.64	0.45	0.84	0.86
D	0.85	0.85	0.23	0.05	0.45	0.76	0.62	0.43	0.83	0.86
E	0.83	0.83	0.48	0.47	0.03	0.72	0.38	0.09	0.80	0.85
F	0.71	0.71	0.77	0.76	0.72	0.05	0.77	0.72	0.37	0.54
G	0.80	0.80	0.65	0.62	0.38	0.78	0.04	0.35	0.78	0.86
H	0.83	0.83	0.48	0.46	0.09	0.73	0.36	0.04	0.80	0.85
I	0.67	0.67	0.83	0.82	0.79	0.38	0.77	0.79	0.05	0.53
J	0.75	0.75	0.86	0.85	0.83	0.52	0.85	0.83	0.52	0.08

metric for clustering. Our initial naive approach to defining similarity was based on the concept of edit distance [7]. In this approach, each behavior is treated as an atomic unit and we measure the number of inserts or deletes of these atomic behaviors required to transform one behavioral fingerprint into another. The method is fairly intuitive and straightforward to implement (think the Unix command *diff* here); however, it suffers from two major drawbacks:

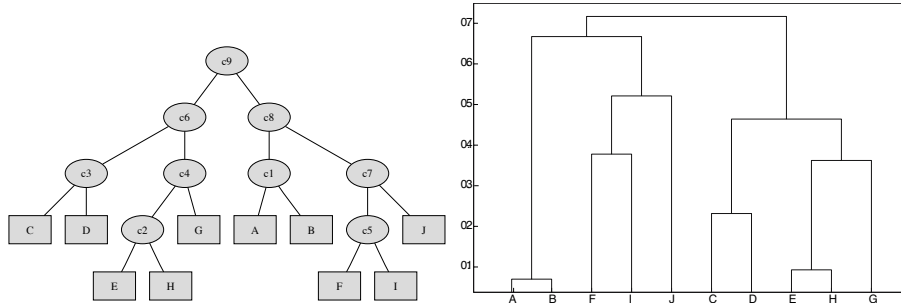
- **Overemphasizing size.** When the size of the number of behaviors is large, the edit distance is effectively equivalent to clustering based on the length of the feature set. This overemphasizes differences over similarities.
- **Behavioral polymorphism.** Many of the clusters we observed had few *exact* matches for behaviors. This is because the state changes made by malware may contain simple behavioral polymorphism (e.g., random file names).

To solve these shortcomings we turned to normalized compression distance (NCD). NCD is a way to provide approximation of information content, and it has been successfully applied in a number of areas [25,29]. NCD is defined as:

$$NCD(x, y) = \frac{C(x + y) - \min(C(x), C(y))}{\max(C(x), C(y))}$$

where "x + y" is the concatenation of x and y, and C(x) is the zlib-compressed length of x. Intuitively, NCD represents the overlap in information between two samples. As a result, behaviors that are similar, but not identical, are viewed as close (e.g., two registry entries with different values, random file names in the same locations). Normalization addresses the issue of differing information content. Table 7 shows the normalized compression distance matrix for the malware described in Table 6.

**Constructing Relationships Between Malware.** Once we know the information content shared between two sets of behavioral fingerprints, we can combine various pieces of malware based on their similarity. In our approach, we construct a tree structure based on the well-known hierarchical clustering algorithm [11]. In particular, we use pairwise single-linkage clustering, which defines the distance between two clusters as the minimum distance between any two members of the clusters. We output the hierarchical cluster results as a tree graph in graphviz's dot format [14]. Figure 2 shows the generated tree for the malware in Table 6.



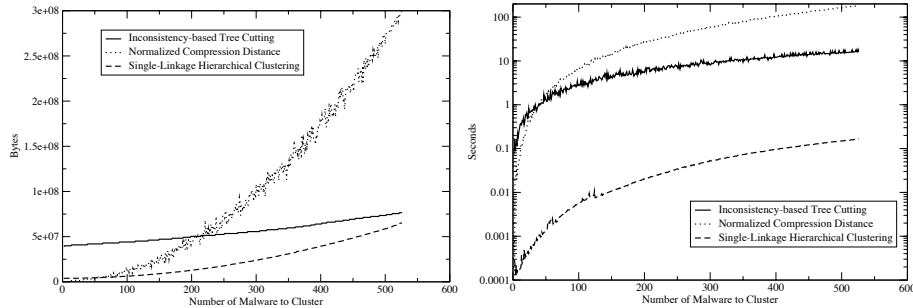
**Fig. 2.** On the left, a tree consisting of the malware from Table 6 has been clustered via a hierarchical clustering algorithm whose distance function is normalized compression distance. On the right, a dendrogram illustrating the distance between various subtrees.

**Extracting Meaningful Groups.** While the tree-based output of the hierarchical clustering algorithm does show the relationships between the information content of behavioral fingerprints, it does not focus attention on areas of the tree in which the similarities (or lack thereof) indicate an important group of malware. Therefore, we need a mechanism to extract meaningful groups from the tree. A naive approach to this problem would be to set a single threshold of the differences between two nodes in the tree. However, this can be problematic as a single uniform distance does not accurately represent the distance between various subtrees. For example, consider the dendrogram in Figure 2. The height of many U-shaped lines connecting objects in a hierarchical tree illustrates the distance between the two objects being connected. As the figure shows, the difference between the information content of subtrees can be substantial. Therefore, we require an automated means of discovering where the most important changes occur.

**Table 8.** The clusters generated via our technique for the malware listed in Table 6

Cluster	Elements	Overlap	Example
c1	C, D	67.86%	scans 25
c2	A, B	97.96%	installs a cygwin rootkit
c3	E, G, H	56.60%	disables AV
c4	F, I, J	53.59%	IRC

To address this limitation, we adopt an “inconsistency” measure that is used to compute the difference in magnitude between distances of clusters so that the tree can be cut into distinct clusters. Clusters are constructed from the tree by first calculating the inconsistency coefficient of each cluster, and then thresholding based on the coefficient. The inconsistency coefficient characterizes each link in a cluster tree by comparing its length with the average length of other links at the same level of the hierarchy. The higher the value of this coefficient, the less similar are the objects connected by the link. The inconsistency coefficient calculation has one parameter, which is the depth below the level of the current



**Fig. 3.** The memory and runtime required for performing clustering based on the number of malware clustered (for a variety of different sized malware behaviors)

link to consider in the calculation. All the links at the current level in the hierarchy, as well as links down to the given depth below the current level, are used in the inconsistency calculation.

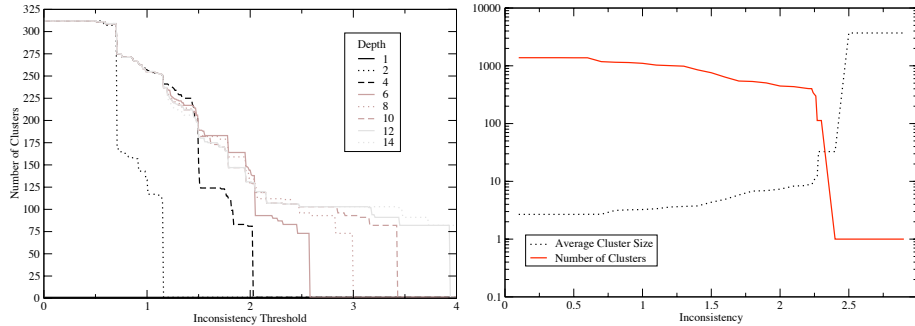
In Table 8 we see the result of the application of this approach to the example malware in Table 6. The 10 unique pieces of malware generate four unique clusters. Each cluster shows the elements in that cluster, the average number of unique behaviors in common between the clusters, and an example of a high-level behavior in common between each binary in the cluster. For example, cluster one consists of C and D and represents two unique behaviors of mytob, a mass mailing scanning worm. Five of the behaviors observed for C and D are identical (e.g., scans port 25), but several others exhibit some behavioral polymorphism (e.g., different run on reboot registry entries). The other three clusters exhibit similar expected results, with cluster two representing the cygwin backdoors, cluster three the bancos variants, and cluster four a class of IRC backdoors.

## 4 Evaluation

To demonstrate the effectiveness of behavioral clustering, we evaluate our technique on the *large* and *small* datasets discussed in section 2. We begin by demonstrating the runtime performance and the effects of various parameters on the system. We then show the quality or goodness of the clusters generated by our system by comparing existing AV groups (e.g., those labeled as SDBot) to our clusters. Next we discuss our clusters in the context of our completeness, conciseness, and consistency criteria presented earlier. Finally, we illustrate the utility of the clusters by answering relevant questions about the malware samples.

### 4.1 Performance and Parameterization

We now examine the memory usage and execution time for the hierarchical clustering algorithm. To obtain these statistics, we take random sub-samples of length between 1 to 526 samples from the *small* dataset. For each sub-sample,



**Fig. 4.** On the left, the number of clusters generated for various values of the inconsistency parameter and depth. On the right, the trade-off between the number of clusters, the average cluster size, and the inconsistency value.

we analyze its run time and memory consumption by running ten trials for each. The experiments were performed on a Dell PowerEdge 4600 with two Intel Xeon MP CPUs (3.00GHz), 4 GB of DDR ECC RAM, 146G Cheetah Seagate drive with an Adaptec 3960D Ultra160 SCSI adapter, running Fedora Core Linux.

We first decompose the entire execution process into five logical steps: (1) trace collection, (2) state change extraction, (3) NCD distance matrix computation: an  $O(N^2)$  operation, (4) clustering the distance matrix into a tree, (5) cutting the tree into clusters. We focus on the latter three operations specific to our algorithm for performance evaluation. Figure 3 shows the memory usage for those three steps. As expected, computing NCD requires the most memory with quadratic growth with an increasing number of malware for clustering. However, clustering 500 malware samples requires less than 300MB of memory. The memory usage for the other two components grows at a much slower rate. Examining the run-time in Figure 3 indicates that all three components can complete within hundreds of seconds for clustering several hundred malware samples.

Phases 1-4 of the system operate without any parameters. However, the tree-cutting algorithm of phase 5 has two parameters: the inconsistency measure and the depth value. Intuitively, larger inconsistency measures lead to fewer clusters and larger depth values for computing inconsistency result in more clusters. Figure 4 illustrates the effects of depth on the number of clusters produced for the *small* dataset for various inconsistency values. Values of between 4-6 for the depth (the 3rd and 4th colored lines) appear to bound the knee of the curve. In order to evaluate the effect of inconsistency, we fixed the depth to 4 and evaluated the number of clusters versus the average size of the clusters for various inconsistency values in the *large* dataset. The results of this analysis, shown in Figure 4, show a smooth trade-off until an inconsistency value between 2.2 and 2.3, where the clusters quickly collapse into a single cluster. In order to generate clusters that are as concise as possible without, losing important

feature information, the experiments in the next section utilize values of depth and inconsistency just at the knee of these curves. In this case, it is a depth value of 4 and an inconsistency value of 2.22.

## 4.2 Comparing AV Groupings and Behavioral Clustering

To evaluate its effectiveness, we applied our behavioral clustering algorithm on the *large* dataset from Section 2. Our algorithm created 403 clusters from the 3,698 individual pieces of malware using parameters discussed above. While it is infeasible to list all the clusters here, a list of the clusters, the malware and behaviors in each cluster, and their AV labels are available at <http://www.eecs.umich.edu/~mibailey/malware/>.

As a first approximation of the quality of the clusters produced, we returned to our example in Section 2 and evaluated the clustering of various malware samples labeled as SDBot by the AV systems. Recall from our previous discussions that various AV systems take differing approaches to labeling malware—some adopt a general approach with malware falling into a few broad categories and others apply more specific, almost per sample, labels to each binary. We expect that a behavior-based approach would separate out these more general classes if their behavior differs, and aggregate across the more specific classes if behaviors are shared. Looking at these extremes in our sample, Symantec, who adopts a more general approach, has two binaries identified as back-door.sdbot. They were divided into separate clusters in our evaluation based on differing processes created, differing back-door ports, differing methods of process invocation or reboot, and the presence of AV avoidance in one of the samples. On the other extreme, FProt, which has a high propensity to label each malware sample individually, had 47 samples that were identified as belonging to the sdbot family. FProt provided 46 unique labels for these samples, nearly one unique label per sample. In our clustering, these 46 unique labels were collapsed into 15 unique clusters reflecting their overlap in behaviors. As we noted in Section 2, these groupings have differing semantics—both Symantec labels were also labeled by FProt as SDBot, but obviously not all FProt labels were identified as SDBot by Symantec. Both of these extremes demonstrate the utility of our system in moving toward a labeling scheme that is more concise, complete, and consistent.

## 4.3 Measuring the Completeness, Conciseness and Consistency

We previously examined how the clusters resulting from the application of our algorithm to the *large* dataset compared to classification of AV systems. In this section, we examine more general characteristics of our clusters in an effort to demonstrate their quality. In particular, we demonstrate the completeness, conciseness, and consistency of the generated clusters. Our analysis of these properties, summarized in Table 9, are highlighted each in turn:

**Completeness.** To measure completeness, we examined the number of times we created a meaningful label for a binary and compared this to the detection rates of the various AV products. For AV software, “not detected” means no

**Table 9.** The completeness, conciseness, and consistency of the clusters created with our algorithm on the *large* dataset as compared to various AV vendors

AV	Completeness			Conciseness		Consistency
	Detected	Not Detected	% Detected	Unique Labels	Clusters or Families	Identical Behavior Labeled Identically
McAfee	2018	1680	54.6%	308	84	47.2%
F-Prot	2958	740	80.0%	1544	194	31.1%
ClamAV	2244	1454	60.7%	1102	119	34.9%
Trend	2960	738	80.0%	2034	137	44.2%
Symantec	1904	1794	51.5%	125	107	68.2%
<b>Behavior</b>	<b>3387</b>	<b>311</b>	<b>91.6%</b>	<b>403</b>	<b>403</b>	<b>100%</b>

signature matched, despite the up-to-date signature information. For behavioral clustering, “not detected” means that we identified no behavior. A unique aspect of this system is that our limiting factor is not whether we have seen a particular binary before, as in a signature system, but whether we are able to extract meaningful behavior. Any such behavior can be clustered in the context of any number of previously observed malware instances and differentiated, although this differentiation is clearly more valuable the more instances that are observed. In our experiments, roughly 311 binaries exhibited no behavior. The root cause of these errors, and a more complete discussion of the general limitations of dynamic path exploration, is available in the Limitations section. A striking observation from the table is that many AV software systems provide detection rates as low as 51%, compared to around 91% using behavioral clustering. It should be noted that these numbers are as much an indication of the mechanisms the vendors use to collect malware as the AV systems themselves, since signature systems can clearly only detect what they have seen before. While it would be unfair to judge the detection rates based on previously unseen malware, we hesitate to point out that our system for collection of these binaries is not unique. In fact, while individual AV system rates may vary, over 96 percent of the malware samples were detected by at least one of the AV systems. These samples are seen significantly more broadly than our single collection infrastructure and many AV systems fail to detect them.

**Conciseness.** Conciseness represented the ability of the labeling system to group similar items into groups that both reflected the important differences in samples, but were devoid of superfluous labels. As in Section 2, we evaluate conciseness by examining the characteristics of the grouping, or clusters, created by AV systems with those created by our system. We examine the number of unique labels generated by the AV systems and a heuristically-generated notion of families or groups of these labels extracted from the human readable strings. For example, the labels W32-Sdbot.AC and Sdbot.42, are both in the “sdbot” family. As we noted before, AV systems vary widely in how concisely they represent malware. Relative to other systems, our clusters strike a middle ground in conciseness, providing fewer labels than the total unique labels of AV systems, but more than the number of AV system families. This observation is consistent with the previous section—the AV system families exhibit multiple different behaviors, but these behaviors have much in common across individual labels.

**Consistency.** Consistency referred to the ability of a labeling system to identify similar or identical items in the same way. In the context of our behavioral system goals, this implies that identical behaviors are placed in the same clusters. In order to measure the consistency of the system, we examined the binaries that exhibited exactly identical behavior. In the *large* sample, roughly 2,200 binaries shared exactly identical behavior with another sample. When grouped, these 2,200 binaries created 267 groups in which each sample in the group had exactly the same behavior. We compared the percentage of time the clusters were identified as the same through our approach, as well as the various AV system. As expected, our system placed all of the identical behaviors in the same clusters. However, because consistency is a design goal of the system, the consistency value for our technique is more a measure of correctness than quality. What is interesting to note, however, is that AV systems obviously do not share this same goal. AV systems only labeled *exactly identical behavior* with the same label roughly 31% to 68% percent of the time.

#### 4.4 Application of Clustering and Behavior Signatures

In this subsection we look at several applications of this technique, in the context of the clusters, created by our algorithm from the *large* dataset.

**Classifying Emerging Threats.** Behavioral classification can be effective in characterizing emerging threats not yet known or not detected by AV signatures. For example, cluster c156 consists of three malware samples that exhibit malicious bot-related behavior, including IRC command and control activities. Each of the 75 behaviors observed in the cluster is shared with other samples of the group—96.92% on average, meaning the malware samples within the cluster have almost identical behavior. However, none of the AV vendors detect the samples in this cluster except for F-Prot, which only detects one of the samples. It is clear that our behavioral classification would assist in identifying these samples as emerging threats through their extensive malicious behavioral profile.

**Resisting Binary Polymorphism.** Similarly, behavioral classification can also assist in grouping an undetected outlier sample (due to polymorphism or some other deficiency in the AV signatures) together with a common family that it shares significant behaviors with. For example, cluster c80 consists of three samples that share identical behaviors with distinctive strings "bling.exe" and "m0rgan.org." The samples in this cluster are consistently labeled as a malicious bot across the AV vendors except Symantec, which fails to identify one of the samples. To maintain completeness, this outlier sample should be labeled similar to the other samples based on its behavioral profile.

**Examining the Malware Behaviors.** Clearly one of the values of any type of automated security system is not to simply provide detailed information on individual malware, but also to provide broad analysis on future directions of malware. Using the behavioral signatures created by our system, we extracted

**Table 10.** The top five behaviors observed by type

Network	Process	Files	Registry
connects to 80	execs cmd.exe	writes winhlp32.dat	uses wininet.dll
connects to 25	execs IEXPLORE.EXE	writes tasklist32.exe	uses PRNG
connects to 6667	execs regedit.exe	writes change.log	modifies registered applications
connects to 587	execs tasklist32.exe	writes mirc.ini	modifies proxy settings
scans port 80	execs svchost.exe	writes svchost.exe	modifies mounted drives

the most prevalent behaviors for each of the various categories of behaviors we monitor. The top five such behaviors in each category are shown in Table 10.

The network behavior seems to conform with agreed notions of how the tasks are being performed by most malware today. Two of the top five network behaviors involve the use of mail ports, presumably for spam. Port 6667 is a common IRC port and is often used for remote control of the malware. Two of the ports are HTTP ports used by systems to check for jailed environments, download code via the web, or tunnel command and control over what is often an unfiltered port. The process behaviors are interesting in that many process executables are named like common Windows utilities to avoid arousing suspicion (e.g., svchost.exe, tasklist32.exe). In addition, some malware uses IEXPLORE.EXE directly to launch popup ads and redirect users to potential phishing sites. This use of existing programs and libraries will make simple anomaly detection techniques more difficult. The file writes show common executable names and data files written to the filesystem by malware. For example, the winhlp32.dat file is a data file common to many Bancos trojans. Registry keys are also fairly interesting indications of behavior and the prevalence of wininet.dll keys shows heavy use of existing libraries for network support. The writing to PRNG keys indicates a heavy use of randomization, as the seed is updated every time a PRNG-related function is used. As expected, the malware does examine and modify the registered application on a machine, the TCP/IP proxy settings (in part to avoid AV), and it queries mounted drives.

## 5 Related Work

Our work is the first to apply automated clustering to understand malware behavior using resulting state changes on the host to identify various malware families. Related work in malware collection, analysis, and signature generation has primarily explored static and byte-level signatures [23,17] focusing on invariant content. Content-based signatures are insufficient to cope with emerging threats due to intentional evasion. Behavioral analysis has been proposed as a solution to deal with polymorphism and metamorphism, where malware changes its visible instruction sequence (typically the decryptor routine) as it spreads. Similar to our work, emulating malware to discover spyware behavior by using anti-spyware tools has been used in measurements studies [22].

There are several abstraction layers at which behavioral profiles can be created. Previous work has focused on lower layers, such as individual system calls [15,10], instruction-based code templates [6], the initial code run on malware



infection (shellcode) [18], and network connection and session behavior [30]. Such behavior needs to be effectively elicited. In our work, we chose a higher abstraction layer for several reasons. In considering the actions of malware, it is not the individual system calls that define the significant actions that a piece of malware inflicts upon the infected host; rather, it is the resulting changes in state of the host. Also, although lower levels may allow signatures that differentiate malware, they do not provide semantic value in explaining behaviors exhibited by a malware variant or family. In our work, we define malware by what it actually does, and thereby build in more semantic meanings to the profiles and clusters generated.

Various aspects of high-level behavior could be included in the definition of a behavioral profile. Network behavior may be indicative of malware and has been used to detect malware infections. For example, Ellis et al. [9] extracted network-level features, such as similar data being sent from one machine to the next. In our work, we focus on individual host behavior, including network connection information but not the data transmitted over the network. Thus, we focus more on the malware behavior on individual host systems instead of the pattern across a network.

Recently, Kolter and Maloof [13] studied applying machine learning to classify malicious executables using n-grams of byte codes. Our use of hierarchical clustering based on normalized compression distance is a first step at examining how statistical techniques are useful in classifying malware, but the features used are the resulting state changes on the host to be more resistant to evasion and inaccuracies. Normalized information distance was proposed by Li et al. [16] as an optimal similarity metric to approximate all other effective similarity metrics. In previous work [29], NCD was applied to worm executables directly and to the network traffic generated by worms. Our work applies NCD at a different layer of abstraction. Rather than applying NCD to the literal malware executables, we apply NCD to the malware behavior.

## 6 Limitations and Future Work

Our system is not without limitations and shares common weaknesses associated with dynamic analysis. Since the malware samples were executed within VMware, samples that employ anti-VM evasion techniques may not exhibit their malicious behavior. To mitigate this limitation, the samples could be run on a real, non-virtualized system, which would be restored to a clean state after each simulation. Another limitation is the time period in which behaviors are collected from the malware execution. In our experiments, each binary was able to run for five minutes before the virtual machine was terminated. It is possible that certain behaviors were not observed within this period due to time-dependent or delayed activities. Previous research has been done to detect such time-dependent triggers [8]. A similar limitation is malware that depends on user input, such as responding to a popup message box, before exhibiting further malicious behavior, as mentioned in [22]. Finally, the capabilities and environment of our

virtualized system stayed static throughout our experiments. However, varying the execution environment by using multiple operating system versions, including other memory resident programs such as anti-virus protection engines, and varying network connectivity and reachability may yield interesting behaviors not observed in our existing results. Recently, a generic approach to these and other problems associated with dynamic analysis has been suggested by Moser, et al. [21]. Their approach is based on exploring multiple execution paths through tracing and rewriting key input values in the system, which could yield additional behaviors unseen in our single execution path.

Our choice of a high level of abstraction may limit fine-grained visibility into each of the observed behaviors in our system. A path for future work could include low-level details of each state change to supplement the high-level behavior description. For example, the actual contents of disk writes and transmitted network packets could be included in a sample's behavioral profile. In addition, we plan to evaluate the integration of other high-level behavioral reports from existing systems, such as Norman [24] and CWSandbox [5], in the future. We will also investigate further clustering and machine-learning techniques that may better suit these other types of behavioral profiles. Finally, the causal graphs from Backtracker, which are used to identify the behaviors in our system, also include dependency information that is currently ignored. In future versions, this dependency information could be used to further differentiate behaviors.

## 7 Conclusion

In this paper, we demonstrated that existing host-based techniques (e.g., anti-virus) fail to provide useful labels to the malware they encounter. We showed that AV systems are *incomplete* in that they fail to detect or provide labels for between 20 to 49 percent of the malware samples. We noted that when these systems do provide labels, these labels are not *consistent*, both within a single naming convention as well as across multiple vendors and conventions. Finally, we demonstrated that these systems vary widely in their *conciseness*—from aggressive, nearly individual labels that ignore commonalities, to broad general groups that hide important details.

To address these important limitations, we proposed a novel approach to the problem of automated malware classification and analysis. Our dynamic approach executes the malware in a virtualized environment and creates a behavioral fingerprint of the malware's activity. This fingerprint is the set of all the state changes that are a causal result of the infection, including files modified, processes created, and network connections. In order to compare these fingerprints and combine them into meaningful groups of behaviors, we apply single-linkage hierarchical clustering of the fingerprints using normalized compress distance as a distance metric. We demonstrated the usefulness of this technique by applying it to the automated classification and analysis of 3,700 malware samples collected over the last six months. We compared the clusters generated to existing malware classification (i.e., AV systems) and showed the

technique's completeness, conciseness, and consistency. Through these evaluations, we showed that this new technique provides a novel way of understanding the relationships between malware and is an important step forward in understanding and bridging existing malware classifications.

## Acknowledgments

This work was supported in part by the Department of Homeland Security (DHS) under contract numbers NBCHC040146 and NBCHC060090, by the National Science Foundation (NSF) under contract number CNS 0627445 and by corporate gifts from Intel Corporation and Cisco Corporation. We would like to thank our shepherd, Jonathon Giffin, for providing valuable feedback on our submission as well as the anonymous reviewers for critical and useful comments.

## References

1. Arbor malware library (AML) (2006), <http://www.arbornetworks.com/>
2. Baecher, P., Koetter, M., Holz, T., Dornseif, M., Freiling, F.: The nepenthes platform: An efficient approach to collect malware. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, Springer, Heidelberg (2006)
3. Barford, P., Yagneswaran, V.: An inside look at botnets. In: Series: Advances in Information Security, Springer, Heidelberg (2006)
4. Beck, D., Connolly, J.: The Common Malware Enumeration Initiative. In: Virus Bulletin Conference (October 2006)
5. Willems, C., Holz, T.: Cwsandbox ( 2007), <http://www.cwsandbox.org/>
6. Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E.: Semantics-aware malware detection. In: Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005), Oakland, CA, USA, May 2005, pp. 32–46. ACM Press, New York (2005)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge, MA (1990)
8. Crandall, J.R., Wassermann, G., de Oliveira, D.A.S., Su, Z., Wu, S.F., Chong, F.T.: Temporal Search: Detecting Hidden Malware Timebombs with Virtual Machines. In: Proceedings of ASPLOS, San Jose, CA, October 2006, ACM Press, New York (2006)
9. Ellis, D., Aiken, J., Attwood, K., Tenaglia, S.: A Behavioral Approach to Worm Detection. In: Proceedings of the ACM Workshop on Rapid Malcode (WORM04), October 2004, ACM Press, New York (2004)
10. Gao, D., Beck, D., Reiter, J.C.M.K., Song, D.X.: Behavioral distance measurement using hidden markov models. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 19–40. Springer, Heidelberg (2006)
11. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, Heidelberg (2001)
12. King, S.T., Chen, P.M.: Backtracking intrusions. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, USA, October 2003, pp. 223–236. ACM Press, New York (2003)
13. Kolter, J.Z., Maloof, M.A.: Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research* (2007)

14. Koutsofios, E., North, S.C.: Drawing graphs with dot. Technical report, AT&T Bell Laboratories, Murray Hill, NJ (October 8, 1993)
15. Lee, T., Mody, J.J.: Behavioral classification. In: Proceedings of EICAR 2006 (April 2006)
16. Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.: The similarity metric. In: SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics, pp. 863–872 (2003)
17. Li, Z., Sanghi, M., Chen, Y., Kao, M., Chavez, B.: Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In: Proc. of IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos (2006)
18. Ma, J., Dunagan, J., Wang, H., Savage, S., Voelker, G.: Finding Diversity in Remote Code Injection Exploits. In: Proceedings of the USENIX/ACM Internet Measurement Conference, October 2006, ACM Press, New York (2006)
19. McAfee: W32/Sdbot.worm (April 2003), [http://vil.nai.com/vil/content/v\\_100454.htm](http://vil.nai.com/vil/content/v_100454.htm)
20. Microsoft: Microsoft security intelligence report: (January-June 2006) (October 2006), <http://www.microsoft.com/technet/security/default.aspx>
21. Moser, A., Kruegel, C., Kirda, E.: Exploring multiple execution paths for malware analysis. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland 2007), May 2007, IEEE Computer Society Press, Los Alamitos (2007)
22. Moshchuk, A., Bragin, T., Gribble, S.D., Levy, H.M.: A Crawler-based Study of Spyware in the Web. In: Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA (2006)
23. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 8–11, 2005, IEEE Computer Society Press, Los Alamitos (2005)
24. Norman Solutions: Norman sandbox whitepaper (2003), <http://download.norman.no/whitepapers/whitepaper-Norman.SandBox.pdf>
25. Nykter, M., Yli-Harja, O., Shmulevich, I.: Normalized compression distance for gene expression analysis. In: Workshop on Genomic Signal Processing and Statistics (GENSIPS) (May 2005)
26. Prince, M.B., Dahl, B.M., Holloway, L., Keller, A.M., Langheinrich, E.: Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In: Second Conference on Email and Anti-Spam (CEAS 2005) (July 2005)
27. Walters, B.: VMware virtual platform. j-LINUX-J 63 (July 1999)
28. Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., King, S.T.: Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA (2006)
29. Wehner, S.: Analyzing worms and network traffic using compression. Technical report, CWI, Amsterdam (2005)
30. Yegneswaran, V., Giffin, J.T., Barford, P., Jha, S.: An Architecture for Generating Semantics-Aware Signatures. In: Proceedings of the 14th USENIX Security Symposium, Baltimore, MD, USA, August 2005, pp. 97–112 (2005)