Host-Rx: Automated Malware Diagnosis Based on Probabilistic Behavior Models

Jian Zhang, Phillip Porras and Vinod Yegneswaran
zhang@csc.lsu.edu, {porras, vinod}@csl.sri.com

Abstract

We explore a new approach to using a VM-based honeyfarm for harvesting complex infection forensics live from the Internet and rapidly applying this gained knowledge to develop a new probabilistic methodology for diagnosing the presence of malware in host computer systems. Our approach builds on a rich model of infection representation that captures the complexities in host forensic attribute priorities and the observed interdependencies among these attributes. We use the model to design an automated host-based malware diagnosis system called *Host-Rx*, which employs probabilistic inference to prioritize symptoms and identify the most likely contagion among a suite of competing diagnosis models. The *Host-Rx* system, and the underlying analytics we employ for symptom prioritization and host-side diagnosis conflict resolution (potentially in the presence of hundreds of malware disease profiles) are inspired by the foundations of abductive-based diagnosis algorithms. Our experimental results illustrate the potential utility and viability of such a system.

1 Introduction

The battle to produce high-performance binary pattern recognition systems or single-event heuristics to detect modern Internet malware binaries has been largely lost by the current generation of Antivirus technologies. Strategies such as polymorphic and metamorphic restructuring of binaries now produce monthly binary sample corpora in the millions, and antivirus companies themselves suggest the average lifetime of a malicious binary may be as little as six hours and two infections. Thus, there is high motivation to explore new generalizable strategies to detect the underlying functionality of malware using techniques that produce high detection rates, are agnostic to malware binary structural modifications, and do not impose excessive system overhead.

It is important to understand that the corpora described above do not represent millions of unique malicious applications, but are rather algorithmically altered variants of orders of magnitude fewer malware programs. Thus, an important question is whether one can reliably detect the presence of the underlying malicious program, regardless of how its code may be restructured or mutated to avoid antivirus detection. One promising avenue is the use of behavioral-oriented detection paradigms that model and diagnose malware infections based on their forensic impact. In such an approach, rather than reliance on signatures that attempt to recognize a binary, multiple behavioral attributes of the malware program itself are probabilistically modeled and used to diagnose the presence of this malware on a victim host.

Following this paradigm, we present our work toward a system for diagnosing malware infections solely through a probabilistic model of how the malware affects that state of its victim host. As with classic AV binary signatures, our diagnoses do depend on previous exposure to the malware family in order to con-

struct our behavioral model. However, this technique differs in that once the behavioral model is generated, our diagnoses are then agnostic to the current crop of binary structural perturbations that prevent detection in classic AV signature systems. We present a system that harvests malware binaries live and unattended from the Internet, and then employ these samples to *automatically* derive infection forensics using an instrumented virtual OS environment, called a malware sandbox. The sandbox is used to collect infection forensics, such as changes to the file system, registry, process, mutex, library and memory alterations and network interactions such as local DNS lookups, connections, and listen port reservations.)

Each set of execution forensics that are produced for a malware binary form a *forensic profile* for that binary. Next, forensic profiles are automatically processed using a clustering algorithm into groups of common profiles. From these common profiles we derive a probabilistic infection model that captures the broadest set of host state changes and state change relationships observed when the clustered malware samples infected their victim sandboxes. Finally, we conduct a targeted attribute sweep on an unknown computer system, and use this scan to diagnose whether this system has been infected by any of our candidate probabilistic infection models, resolving conflicts and selecting the best match when multiple models match the computer's current forensic state. We call our system the *Host-Rx* system, in analogue to a doctor who diagnoses a patient based on his symptoms.

While our work is similar, and informed by, prior studies that have applied clustering algorithm to malware forensic attributes for the purpose of classification and labeling [3, 22], our work is distinguished by its application of the forensic cluster. The goal of Host-Rx is to demonstrate how to build probabilistic diagnosis models from clustered attributes, and to further use these probabilistic models to conduct infection diagnosis on operational systems. Our objective is to construct a diagnosis model that captures a common and distinct patter of behavior, rather than attempting to express all variant behavior. Clustering serves as a condensation process in our system. Behavior patterns are enriched in each cluster which helps our knowledge extraction and model building process.

Host-Rx is composed of three components: (i) malware harvesting and infection forensic/behavior analysis, (ii) infection knowledge extraction from malware forensic/behavior profiles, and (iii) infection diagnosis using the gained knowledge. There are two main steps in conducing our forensic analysis. First, we group malware according to their behavior by clustering. Second, we extract explicit patterns exhibited by each behavior group and construct a probabilistic model for each group based on those behavior patterns. The probabilistic models encode knowledge about the malware's behavior and is later used in diagnosing malware infection. The explicit pattern extraction for each malware behavior group and the probabilistic model based on these patterns form the novel contributions introduced by our system.

We report on a set of experiments with our Host-Rx system. The experiment uses behavior profiles of the harvested malware as the true positives and profiles of clean computer systems as the true negatives. The experimental results shown in this paper suggest that behavioral modeling can produce true positive detection results with an accuracy rate close to 90%. Our limited diagnosis experiment on 20 benign system profiles produced zero false positives.

There are two main contributions in our work. First, we take malware behavior clustering to its natural next step. We use clustering to help the extraction of behavior patterns and the construction of probabilistic models. The patterns and models are incorporated into a system that diagnoses malware infection on operational computers. Second, we introduce probabilistic models for malware diagnosis that is based on the behavior patterns extracted from each cluster. Even after clustering, the malware in the same cluster may behave differently to a certain extent. Probabilistic modeling provides a way to deal with uncertainty and variants

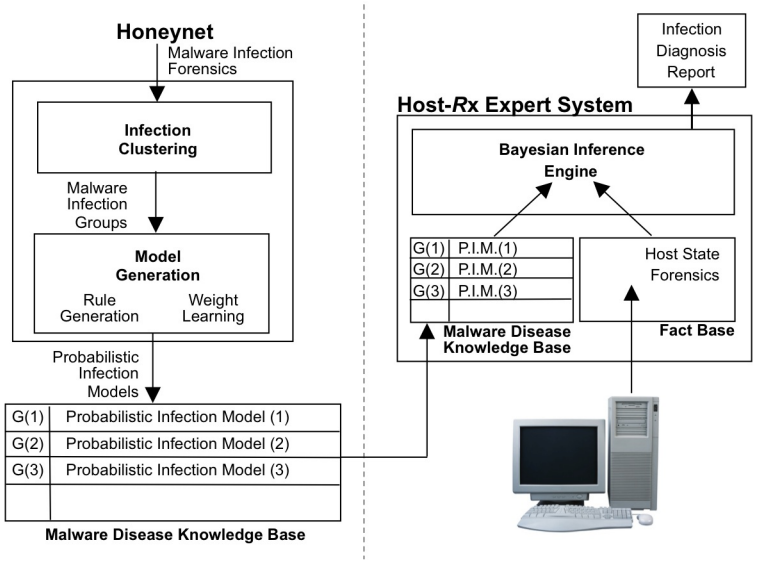

Figure 1: The Host-Rx Framework

in the clusters.

The rest of the paper is organized as follows. In Section 2, we describe our diagnosis system and its components in detail. In Section 3, we present the result of our experimentations. We discuss limitations of the system in Section 4 and related work in Section 5. We conclude with a summary in Section 6.

2 Host-Rx Diagnosis System

Host-Rx is a new form of malware infection diagnosis system, under which live Internet infections are automatically assimilated into a probabilistic infection model. These models are composed of weighted forensic detection rules, which capture the unique state changes associated with each malware infection. The models are compiled into a malware disease knowledge base, which is published to expert systems that are deployed to operational computers across the Internet. These expert systems periodically interrogate the forensic state of their hosts, gathering a corpus of relevant state attributes that are then compared against the myriad of probabilistic infection models—some of these models will potentially capture new infection behavior patterns that have emerged recently. The expert system's task is to evaluate its host's current system state against the models in the knowledge base. The expert system must determine whether the subject computer's forensic state matches any of its candidate infection diagnosis models, and to conduct a best fit analysis when multiple competing diagnostics models appear to match. We illustrate the components of the Host-Rx system in Figure 1.

The left panel of Figure 1 represents components and data flows that occur during the automated formulation of the malware disease knowledge base. Raw malware infection forensic data produced from an Internet honeynet is used to drive the creation of the knowledge base. A behavior clustering algorithm produces a set of malware infection groupings (G(1), G(2),...G(N)). Each group represents the combined forensic footprint of malware infections that are found to have a certain degree of behavioral similarity. Each malware infection grouping is then applied to a dynamic learning algorithm, which derives from the infection grouping a set of forensic detection rules (i.e., predicates that describe the set of malware-related state changes). Based on the

rules, a probability model M is derived, which defines the probability that a computer with symptom set S is infected by malware from the model M. The collection of the probability models for all the infection groups forms a malware knowledge base, which is then used for diagnosis. The right panel illustrates an instance of our Host-Rx expert system. This application can be deployed to host machines as a complementary service to the antivirus and antispyware that regularly runs on host systems. The system interrogates the host machine and collects the states and behavior of the host. The interrogation report is then compared to each model in the knowledge base, and from this comparison a probability is derived to define the likelihood of infection.

In the following subsections we describe the components that generate the probabilistic models and that subsequently use these models in the knowledge base to diagnose possible malware infection on host computer systems.

2.1 Behavior Analysis and Clustering

The objective of the infection forensics harvesting component is to derive a set of features that could be used to identify an infected host. For this, we trace the behavior of the malware by running the malware executable in a sandbox environment for several minutes. We do not assume that Host-Rx is running on the system prior to infection or that Host-Rx is able to observe the startup of the running malware process. Hence, we do not use API hooking techniques for forensic feature extraction. Instead, our approach relies on a combination of features that are based on comparing the pre-infection and post-infection system snapshots. Some of the key features collected include AUTORUN ENTRIES, CONNECTION PORTS, DNS RECORDS, DROP FILES, PROCESS CHANGES, MUTEXES, THREAD COUNTS, and REGISTRY MODS. We use a whitelisting process to downweight certain commonly occurring attribute values for filenames and DNS entries. To identify deterministic and non-deterministic features each malware is executed three different times on different virtual machines and a JSON object is generated describing the malware behavior in each execution, as illustrated in Figure 2.

We use an agglomerative hierarchical clustering algorithm to group similar malware infections into clusters. The algorithm progressively merges elements in a set to build hierarchies. It partitions a dataset S of n elements into partitions (groups of clusters) $Q_1...Q_k$, where each partition has multiple clusters. Here, Q_1 is the partition that has N clusters, each containing a single malware instance and Q_k is the partition with a single cluster containing all N malware instances. The algorithm processes from Q_{i-1} to Q_i at each stage and joins together the two clusters which are most similar. We stop the joining when the distance between all pairs of clusters exceeds a threshold (distance criterion).

To measure the similarity between two malware infections, we match the behavior attributes of the two. A profile of malware infection is the collection of its behavior attributes. We call the attributes in a behavior profile the *symptoms* of the malware infection. Let $S = \{s_1, s_2, \ldots, s_n\}$ be the set of symptoms exhibited by malware infection 1 and $T = \{t_1, t_2, \ldots, t_m\}$ the symptoms of infection 2. We use the amount of matching between S and T as the measure of similarity between the two infection. One can construct a bipartite graph for the two infections. Each s_i corresponds to a node on one side and t_j to a node on the other side. There is a weighted edge between the node for s_i and the node for t_j if their two symptoms show a certain degree of similarity. The similarity measure between the two infection profiles can then be calculated by the maximum weighted matching in the bipartite graph. This calculation also allows partial matching. For example, s_i and t_j can be two processes belonging to two infections. They may have different process names but they may use the same DLL and other resources. By adding a connection between the two nodes corresponding to the two symptoms, such a relationship is taken into consideration in calculating the final similarity.

```
{
AUTORUN_ENTRIES : [
{  Entry Location :  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run,
   Entry : eggs joy math type,
   Enabled : enabled,
   Description : Contains anave on otherede skin toplofre tismane,
   Publisher : Ithellen Reject,
   ImagePath : c:\documents and settings\all users\application data
         \bind army eggs joy\gram bias.exe,
   LaunchString : C:\Documents and Settings\All Users\Application Data\Bind
         army eggs joy\gram bias.exe,
   MD5 : 550176d229beea38bfb8154f6c85085c,
},
{  Entry Location :  HKCU\Software\Microsoft\Windows\CurrentVersion\Run,
   Entry : roam2,
   Enabled : enabled,
   Description : Range be children the clear tarno keryedin,
   Publisher : Ondu Great,
   ImagePath : c:\documents and settings\sri-user\application data\
               boltcloseseek\one enc.exe,
   LaunchString : C:\DOCUME~1\SRI-user\APPLIC~1\BOLTCL~1\One Enc.exe,
   MD5 : 68cd5d7bc0f5176c1e0788df49958a60,
},
{  Entry Location :  Task Scheduler,
   Entry : A648A72591835FA1.job,
   Enabled : enabled,
   Description : Present no cost chima wap nemb,
   Publisher : Ser Neceh,
   ImagePath : c:\documents and settings\sri-user\application data\
               boltcloseseek\kindphoneblah.exe ,
   LaunchString : c:\docume~1\sri-user\applic~1\boltcl~1\KINDPHONEBLAH.exe ,
   MD5 : 855c902ba3ffd20cdc50239d0b48c6a6,
},
],
DNS_RECORDS : [ ],
FORENSIC_DROP_NAME_LIST : [
   C:\Documents_and_Settings\SRI-user\Cookies\sri-user@ayb.host127-0-0-1[1].txt,
],
```

```
FORENSIC_PROCESS_LIST : [
{  name : iexplore.exe,
   cmdargs : "",
   execpath : C:\Program Files\Internet Explorer\iexplore.exe,
   handles : 146,
   threads : 3,
   openfiles : [ c:\scripts\, c:\docume~1\alluse~1\applic~1\bindar~1\grambi~1.exe ],
   mutexes : [hklm\system\controlset001\services\winsock2\parameters\namespace_catalog5,
              hklm\system\controlset001\control\nls\language groups,
              hklm\system\controlset001\control\nls\language groups],
   netports : [],
   regkeys : [hklm\software\microsoft\windows\currentversion\telephony\country list\1 ,
              hkcu\software\name 01 long ,
              hkcu\software\microsoft\windows\currentversion\internet settings\zonemap],
   dlls : [c:\windows\system32\mfc42.dll ,  c:\windows\system32\mstask.dll ],
},
{       name : iexplore.exe,
   cmdargs : "",
   execpath : C:\Program Files\Internet Explorer\iexplore.exe,
   handles : 148,
   threads : 9,
   openfiles : [c:\documents and settings\all users\application data\bind army eggs joy\ ,
                \device\afd\asyncconnecthlp ],
   mutexes : [hklm\system\controlset001\services\netbt\parameters ],
   netports : [],
   regkeys : [],
   dlls : [c:\windows\system32\msxml3.dll ],
},
],
REGISTRY_MODS_LIST : [ ],
NET_DNS_LIST : [
   ads.range159-195.com,  ayb.host127-0-0-1.com,
   h5323.nb.host-domain-lookup.com, n596.nb.host127-0-0-1.com,
   v2367.nb.host127-0-0-1.com, x6785.nb.host127-0-0-1.com,
],
NET_PORTS_LIST : [ 80 ]
}
```

Figure 2: Behavioral summary JSON for an unclassified malware instance

Once the similarity/distance measure is obtained, we perform hierarchical agglomerative clustering to produce a hierarchical tree with all its profiles. Then to identify meaningful clusters, we walk the tree and at each node, splitting the two branches into different clusters when the average distance between the nodes in the branches exceeds a certain distance. (We discuss this threshold later in the experimentation section.)

2.2 Malware Disease Diagnosis

Malware infection diagnosis is the process of determining, from a behavioral profile of a computer system, whether the system is infected with malware, and if so, which type of infection it has. By "type", we mean behavior type, i.e., each cluster produced by the method described in the previous subsection corresponds to a behavior type. For this purpose, we define a MIG (*malware infection group*) to be a collection of malware instances that have similar infection impact. Identifying what type of infection a computer has is essentially a search for which MIG the host most closely matches, and deciding whether this similarity has reach a threshold sufficient for declaring an infection.

Our diagnosis is based on a malware knowledge base learned from a collection of malware grouped into MIGs. The knowledge base consists of a set of probabilistic infection models, each describing a MIG. Each probabilistic model includes a set of (first order) logic rules that capture the forensic states of host machines infected by elements of the group. The rules and their associated weights together define the probability of a malware belonging to a certain MIG. (From the probabilistic model's point view, one may treat the non-infected case as a special MIG.)

Formally, let $\{s_1 s_2, \ldots\}$ be the set of symptoms the malware in a particular MIG displays. We may view s_i as a predicate, such that $s_i(x) = 1$ if the malware x exhibits symptom s_i and $s_i(x) = 0$ otherwise. A rule r is either a single symptom predicate or any logical combination of the symptom predicates, e.g., $s_1 \wedge s_2$ and $\neg s_1 \vee (s_2 \wedge s_3)$. We call a rule that is a logical combination of the symptom predicates a *combination rule*. A weight w_j is associated with each rule $r(j)$. The infection model for a particular MIG k consists of the set of rules $\{r^k(1), r^k(2), \ldots, r^k(n)\}$ and their corresponding weights $\{w_1^k, w_2^k, \ldots, w_n^k\}$. Let $S(x) = \{s_1, s_2, \ldots, s_m\}$ be the collection of symptoms in profile x. The rules and weights jointly determine the probability $P(\text{MIG(x)} = k|S(x))$, i.e., the probability that the profile belongs to MIG k given the set of symptoms S. We use a logistic regression model to define this probability:

$$P(\text{MIG(x)} = \text{k}|S(x)) = \frac{1}{Z(x, W)} \exp\left(\sum_i w_i f_i(x, k)\right) \tag{1}$$

where w_i is the weight associated with the ith rule and W is the weight vector whose ith entry is w_i. f_i is the boolean function defined by the ith rule. $Z(x, W) = 1 + \sum_k \exp(\sum_i w_i f_i(x, k))$ normalizes the probability $P(\text{MIG(x)} = k|S(x))$ to ensure that the probabilities for different MIG sum to one.

Probabilistic infection models have the ability to express complex relations between forensic features. We contrast it to linear functions, which are a simple and commonly used technique for data classification. With linear functions, a score is computed by summing the weights specified across all malware features. If the score exceeds a specified threshold, the malware is classified as belonging to a certain MIG. However, this approach has significant limitations.

Consider a malware group (say MIG I) that exhibits two distinct behavioral patterns. In one scenario, the malware instance creates a registry key A and modifies a file C. In an alternate scenario, it creates a registry key B and performs a DNS lookup D. Let events A, B, C, and D be the observed malware forensic attributes.

We state the forensic impact of this type of malware using the logic expression $(A \wedge C) \vee (B \wedge D)$. Given a host profile x, the boolean function corresponding to the rule is:

$$f(x, I) = \begin{cases} 1 & \text{if } (A(x) \wedge C(x)) \vee (B(x) \wedge D(x)) \\ 0 & \text{otherwise.} \end{cases}$$

where each letter A-D is a predicate, testing whether a host exhibits a specific forensic state change. It is hard to express this pattern using the sum-of-the-attribute-weights scheme. For example, we may assign weight 0.5 to A, B, C, and D and set a diagnosis threshold of 1. Clearly, a host found to exhibit both A and C will produce a score above the threshold, and will be diagnosed with a MIG I infection, as will any host that exhibits attributes B and D. However, hosts that exhibit attributes A and D will also be classified as MIG I, resulting in false positives. Therefore, the sum-of-weights scheme (or any linear function) is inadequate to express situations where an attribute pattern is relevant only when some precondition is satisfied.

We mine the combination rules for each MIG using a *frequent itemset mining* technique. If two symptoms A and B are correlated, i.e., they show up together in many of the profiles in a cluster, we will make a rule $A \wedge B$. Formally, let $Pf(A)$ be the set of profiles in the cluster that contain symptom A and $Pf(B)$ be the set of profiles in the cluster that contain symptom B. We form rule $A \wedge B$ if $\frac{|Pf(A) \cap Pf(B)|}{|Pf(A) \cup Pf(B)|} > \tau$ for a threshold τ. (We use $|\cdot|$ to indicate the size of a set.) Because our model is probabilistic, a rule does not necessarily apply to all the members in a cluster. In fact, one may view the rules as candidate features that may help to distinguish different clusters. In the later training process, we learn the weights such that if a rule is irrelevant, its weight becomes zero (practically eliminating the rule). Therefore, in the mining stage, one may set a loose threshold because we are identifying a candidate, not the final effective rules. (In our experiment, we set τ to be 30%.)

We use min-hash based mining to identify the combination rules. A min-hash function h_{min} maps a symptom to a number and has the following property: Given two symptoms A and B,

$$p(h_{min}(A) = h_{min}(B)) = \frac{|Pf(A) \cap Pf(B)|}{|Pf(A) \cup Pf(B)|}.$$

The larger the ratio $\frac{|Pf(A) \cap Pf(B)|}{|Pf(A) \cup Pf(B)|}$, the more likely it is that the two symptoms A and B will be hashed to the same value. In this way, correlated pairs can be identified. We also extend this technique to mine rules that involve more than two symptoms.

Once we obtain a set of rules (both single symptom rules and combination rules) for each MIG, we learn the weights for the probabilistic models. We take a maximum likelihood approach to derive the weight. Following Eq. 1, the joint conditional log-likelihood of the collection of training profiles can be written as:

$$\mathcal{L} = \sum_{j=1}^{N} \left(\sum_{i=1}^{M} w_i f_i(x_j, MIG(x_j)) - \log Z(x_j, W) \right) \tag{2}$$

where N is the number of profiles in the training set and M is the total (sum over all MIGs) number of rules. The optimal weights are those that maximize this likelihood. Taking the derivative of Eq. 2 with respect to w_i gives:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_j f_i(x_j, \text{MIG}(x_j)) - \sum_j \frac{1}{Z(x_j, W)} \sum_k \exp\left[\sum_s w_s f_s(x_j, k) \right] \cdot f_i(x_j, k)$$

$$= \sum_j f_i(x_j, \text{MIG}(x_j)) - \sum_j \sum_k p(\text{MIG}(x_j) = k | S(x_j)) \cdot f_i(x_j, k)$$
$$= \sum_j \Big(f_i(x_j, MIG(x_j)) - \mathbf{E}_{p(k|x_j)} f_i(x_j, k) \Big) .$$

where $\mathbf{E}_{p(k|x_j)} f_i(x_j, k)$ is the expected value of f_i under the model distribution of the MIG label k. It shows that, under the optimal probability distribution, the empirical number of times f_i is true is equal to the expectation of this number. We solve this optimization problem numerically using the quasi-Newton method L-BFGS [17].

After learning the weights, we have a complete set of models for the MIGs. Given a particular profile of symptoms, the diagnosis process calculates the probability of the profile belonging to each MIG as well as the probability that it is from a machine that is not infected. The final decision is the case with the largest probability over all the cases (MIGs or uninfected).

3 Experimentation

We test our system on a collection of malware. We first evaluate the system's detection capability through a set of experiments and then discuss the clustering step.

3.1 Malware Infection Diagnosis

The basic task a diagnosis system needs to perform is to decide, given the forensic profile of a host, whether the host is infect by some malware or not. To do so, Host-Rx first constructs diagnosis models from the behavior profiles of a sample of malware collections. And then using these models, it performs infection diagnosis on operational host. In a practical setting, the collection of sample malware is always incomplete (i.e., the malware we can harvest for building the diagnosis models is always a part of the whole collection of malware in the wild). To test the diagnosis capability of Host-Rx system, we conduct experiments in a fashion similar to the practical setting.

Our collection of malware that we used for testing Host-Rx includes 2140 distinct malware instances. Out of these instances, we used a certain percentage as training samples, perform clustering on the samples to form infection groups and construct a diagnosis model for each group. We then ran the system on the (behavior profile) of the remaining instances (view them as the profiles from host machines that need to be diagnosed). Clearly, a perfect system should diagnose all the profiles as infected. We also use the behavior profiles of 20 clean machines (7 real and 13 virtual) to test the false positive rate of the system. These 20 machines were operational systems running Windows XP and Windows 2000, with different service pack levels and software configurations that were currently under use for various projects.

We conducted our experiments with four different mixtures of training and test data (10/90%, 20/80%, 60/40% and 90/10%). Our results, plotted in Figure 3, show the detection rate (i.e., the fraction of malware profiles that are diagnosed to be infected) of our system to vary between 80% and 90%. Each data point is the average of 5 experiments under the same setting (but different random selection of the training malware). In each experiment, we also ran the system on profiles obtained from the clean machines and there were no false positive diagnoses by Host-Rx. This experiment validates the diagnosis power of the system.

Our results also indicate that the more sample malware we have, the better diagnosis Host-Rx can achieve. When we have observed 90% of the malware, the detection rate is close to 90% (with no false positive

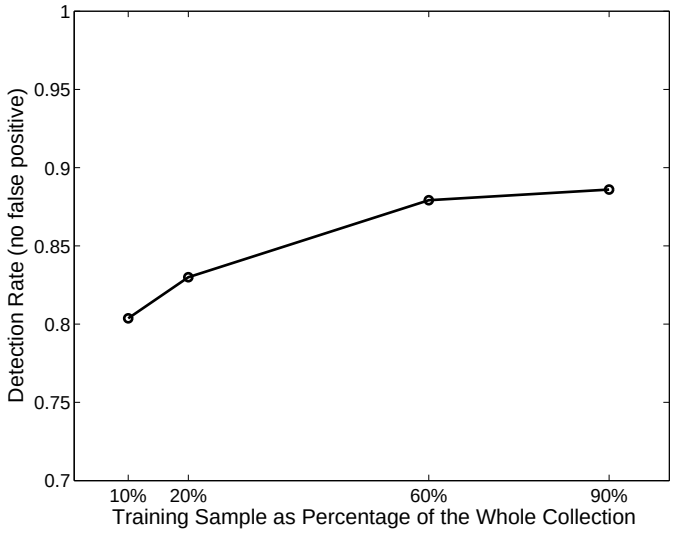

Figure 3: Detection Rate of Host-Rx with Different Training Size

on the 20 clean machine). The malware instances where Host-Rx issues a false negative diagnosis display variant behaviors that are different from the other malware. When the sample malware is only 10% of all the malware, Host-Rx can still achieve a detection rate above 80%.

We also conducted experiments to test the effectiveness of the combination rules. We repeated the above experiments, using models with and without combination rules. Table 1 shows the change of false negative rate when the combination rules are added. Clearly, the combination rules help to improve detection, although the effect is not large for the set of malware used in our experiments.

	10%	20%	60%	90%
with comb. rules	0.196	0.170	0.121	0.114
without comb. rules	0.207	0.172	0.130	0.114

Table 1: False Negative Rate with and without Combination Rules (no false positive in both case)

3.2 Malware Group Diagnosis

Besides determining whether or not a host is infected, Host-Rx also attempts to attribute each infection diagnosis to a malware infection group (MIG). Note that malware infection group may differ from malware families defined by the AV companies. Malware in the same family may form behavior subgroups due to variant behavior in the family. Also, there may be malware instances for which AV systems currently have no label definition. These malware may form behavior clusters and Host-Rx can identify these clusters and construct diagnosis models for them. When diagnosing operational machines, Host-Rx reports the infection group labels along with the diagnosis result. Such group information may also help in the development of recovery rules for system clean up.

Although the group diagnosis capability might seem similar to the task of malware classification, Host-Rx is very different from a malware classification system. Given a collection of malware, a malware classification system groups the malware into families using techniques such as clustering. However, for Host-Rx, the

first task in diagnosis is to decide whether the system is infected or not. It cannot simply run clustering on the forensic profiles from undetermined host machines and associate each cluster to a malware family (as a malware classification system would do), since many undetermined host machines may be uninfected. And there may be infections that behave differently from any of the families.

In the following, we describe results from experiments that test the Host-Rx's capability in infection group diagnosis. This set of experiments are different from experiments described in the previous section. We still divide the malware into two sets: one used for training (clustering and model building) and the other for testing. In the previous experiments, we had apriori knowledge about the true state of the machines that generated the profiles, i.e., whether the profile was from a malware infection or a clean machine. In the experiments to test group diagnosis, we don't have the information about which group the infection should belong to since the testing profiles are left aside and are not included in the clustering step. To illustrate the results from the group diagnosis experiments, we plot the similarity between the testing malware infection profile and the training malware profiles. A good group diagnosis would generate results such that when an infection is diagnosed to be in group k, the infection profile should show high similarity to the training samples in group k and low similarity to the samples in other groups.

There is a technical detail we need to deal with in this experiment. After clustering, we obtain malware infection groups. Some of the groups are very small (less than 10 profiles). The members of the group are often outlier profiles. Because these groups are too small for us to build a diagnosis model, we merge all the small groups together and treat the amalgam as a special group. If Host-Rx indicates that a profile should belong to the amalgam group, we may not be able to identify the type of infection, but we still identify it as a malware infection.

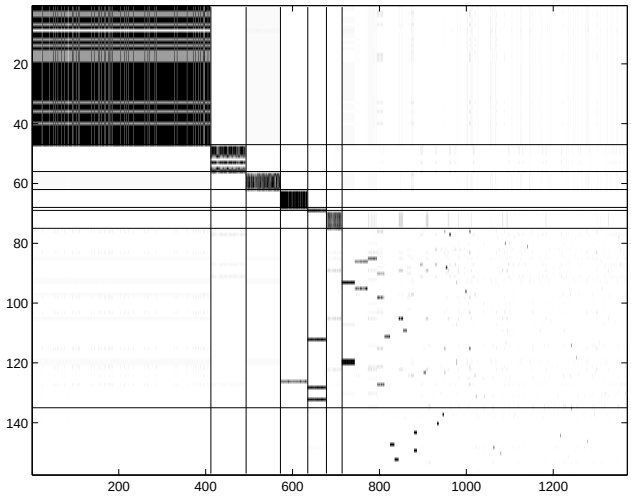

Figure 4: Malware diagnosis similarity matrix comparing testing and training profiles

To illustrate the diagnosis result, we generate a matrix showing the similarity between the testing and the training malware sets. Each row of the matrix corresponds to a test malware and each column a training malware. Entry (i, j) of the matrix represents the similarity between the ith testing malware and the jth training malware. We group the training malware according to their cluster and the testing malware according to the diagnosis result. We plot this similarity matrix in Figure 4 using grayscale to show the similarity (dark–high similarity, white–low similarity) and lines to separate groups. The plot shows that malware infections diagnosed by Host-Rx show strong behavioral similarity with training malware in that cluster. For

example, profiles corresponding to rows 1-48 are diagnosed by our system to belong to MIG 1 (cluster 1). The similarity measure shows that most of the behaviors are indeed similar to that of the members in cluster 1. The rows in the second to the last block (rows 76-135) are diagnosed by Host-Rx to be infected. However, the diagnosis cannot tell which cluster the infection belongs to. (Host-Rx maps them to the amalgam cluster.) This is an example of the case we discussed before. That is, although the type of the infection is not clear, our system is still able to diagnose the profile to be an infection.

The bottom few rows in the plot correspond to malware profiles that are diagnosed by our system to be uninfected, i.e, false negatives. The plot shows that many false negatives have extremely low or zero similarity to any of the cluster groups, including the amalgam group. Those malware can be viewed as variants whose behavior is (almost) completely different from what we experienced in training. In such cases, diagnosis would be extremely difficult. This is a fundamental limitation for any expert system, including Host-Rx.

3.3 Malware Clustering

As discussed in the introduction, we pursue malware clustering with the objective of automating malware diagnosis. The clustering step groups together malware instances with similar behavioral profiles and prioritizes patterns common to the members of a cluster for the knowledge extraction process. Clustering is not the core of the Host-Rx system. However, it is still important that we produce high quality clusters. To validate the similarity measure and the clustering method used in Host-Rx, we select a set of malware instances from our collection that can be reliably labeled by AV programs. We then calculate the similarity measure and perform clustering on these instance. A good clustering system should produce groups corresponding to the labels.

The following process was used to select malware instances with reliable labels. First, we used virustotal to obtain six different AV labels for the set of malware instances used in our experiment. For most malware instances, these programs either do not yield a label or yield inconsistent labels. To obtain reliable labels, we further downselected 4 vendors who showed maximal consistency in labeling. (The labels from the other 2 are quite different.) We attribute a label for a malware instance if at least 2 out of these 4 AV programs attribute the same label to the instance. Out of the 2140 malware instances, we were able to obtain only 450 malware samples with consistent labels. To test clustering, each malware family should have a minimal number of exemplars, as otherwise, the result has minimal significance. Therefore, from our collection of labeled malware, we removed families with less than 20 members and compiled a final collection of 393 malware instances for clustering.

These malware instances belong to six families: Allaple, Mydoom, Virut, Udr, Tibs and Ejik. We calculated the behavior similarity between the profiles of these instances and perform clustering. Except for one cluster, the resulting clusters contained instances from only one family. However, malware from the same family were sometimes split into multiple clusters as there are behavior subgroups in the family and some of the instances display outlier behavior. We summarize the clustering result in Table 2. Note that the row "outlier" does not correspond to one cluster. Malware with outlier behavior, after clustering, fall into clusters of very small size often containing only one instance. We treat clusters of size less than 5 as invalid clusters and view the members of these clusters as outliers. The row "outlier" gives the number of instances from each family that are viewed as outliers.

As shown in the table, except for one cluster, there is no mixing of malware instances from different family in the same cluster. Instances from some family show consistent behaviors and most such instances are grouped into one cluster. The Ejik family of adware agents, on the other hand, displays variant behavior

	Mydoom	Allaple	Virut	Udr	Tibs	Ejik
Cluster1	150					
Cluster2		73	4			
Cluster3			46			
Cluster4				34		
Cluster5					27	
Cluster6						9
Cluster7						7
Outlier	1	8	16		4	14

Table 2: Summary of Clusters

and is divided into multiple clusters and many of its members are outliers. However, this does not mean the malware cannot be detected by Host-Rx. As we demonstrated in the previous section, there are malware whose infection forensics do not form clusters but Host-Rx is still able to detect most of these malware infections. It's just that in this case, Host-Rx cannot reliably attribute it to a specific malware infection group.

4 Limitations

While we believe that Host-Rx is a viable approach for developing models that enable accurate diagnosis of malware infections, there are certain limitations.

1. Malware could thwart model building or diagnosis by intercepting API calls that are used to build the features. As a first step in addressing this limitation Host-Rx does not rely on a single feature, but instead relies on a collection of attributes. To fully address this limitation in the model building step, we plan on moving the attribute collector to a hypervisor. Similarly, the diagnosis component could be moved to the kernel to support environments where running a VM is not a option.

2. Knowledgeable adversaries could design malware that exhibits infinite polymorphism or inject spurious noise, making it resistant to classification. Host-Rx is designed to tolerate polymorphism in a finite set of attributes. However, if all attributes are polymorphic and there is no invariant behavior, it becomes a challenge to any learning system that is used to develop signatures. We plan on exploring game-theory as a means to counter such threats and evaluate the worst case performance of system under such scenarios. Similar adversarial classification techniques have been explored in the context of spam detection [5, 12].

5 Related Work

The related work can be broadly grouped into three categories: malware classification and analysis, automated signature generation systems, and probabilistic modeling techniques. We discuss and differentiate our approach from existing work in each area.

Malware classification and analysis: Malware classification is an important problem that has attracted considerable attention. Of particular relevance are recent efforts that have tried to develop models for describing malware phylogeny. Karim *et al.* examine the problem of developing phylogenetic models for detecting malware that evolves through code permutations [13, 26]. Carrera *et al.* develop a taxonomy

of malware using graph-based representation and comparison techniques for malware [7]. Another approach to malware analysis is behavior-based techniques to classify malware [8]. Several tools exist for sandboxed execution of malware including CWSandbox [27], Norman Sandbox [21], and TTanalyze [25]. While CWSandbox and TTanalyze leverage API hooking, Norman Sandbox implements a simulated Windows environment for analyzing malware. A complementary analysis is proposed in [19], where a layered architecture is proposed for building abstract models based on run-time system call monitoring of programs. Bailey et al. present an automated classification system for classifying malware binaries through offline behavioral analysis [3]. Similarly, Rieck et al. propose a learning approach to behavior-based malware classification. We differ from these in that their objective is not to build a diagnosis system, but to solve the labeling problem. Our diagnosis system can be informed by our own infection clustering results and by theirs. Another related work is a contemporary paper from Kolbitsch et al. [15], where dynamic analysis is used to build models of malware programs based on information flow between system calls. While our objective is the same as theirs, we differ in the attributes that are considered for diagnosis. Furthermore, we use a probabilistic model for diagnosis which tolerates uncertainty and small variations in malware behavior.

Signature generation: The motivation and approach adopted by our system is similar to prior work on automated network-based intrusion signature generation systems such as Honeycomb [16], Autograph [14], Earlybird [24], and Nemean [28]. We are also inspired by efforts to generate vulnerability signatures [6] and other host-based approaches that use host information to detect anomalies and generate signatures such as TaintCheck [20], Vigilante [10], and DACODA [11]. However, unlike the aforementioned detection systems, Host-Rx emphasizes post-infection diagnosis, and its infection models are multi-perspective in considering both network behavior and host forensic changes. Prior work has also studied the problem of attacks against learning-based signature systems and the cost of countermeasures [4, 9]. Data pollution attempts from knowledgeable adversaries pose a problem for our system as well.

Probabilistic diagnosis: Many medical diagnosis systems are based on probabilistic models. Examples include interpretation of electromyographic findings [2], insulin adjustment [1], and infectious disease management [18]. A well-known system is the QMR-dt (quick medical reference - decision theoretic) model [23] that combines expert knowledge and probabilistic reasoning to provide diagnostic support. Bayesian inference is often used in these systems. In the Bayesian model, it is assumed that the symptoms are independent (in probability) given the disease. Malware behavior attributes, however, are often not independent. Also, HostRx's probabilistic knowledge base includes combination rules, which are not independent in most cases. Therefore, instead of Bayesian, we employ a generalized loglinear model for malware behaviors.

6 Conclusion and Future Work

In this paper, we present a new probabilistic methodology for diagnosing malware infections and evaluate a prototype system that implements our technique. Our evaluation demonstrates the feasibility of this approach and the potential of an automated diagnosis system to accurately detect a large class of infections with minimal false positives. It also illustrates some of the challenges in building an automated diagnosis system, such as dealing with polymorphic malware behavior, diverse attribute collection and the need for whitelisting. In the future, we plan to expand our set of forensic attributes, testing on a larger corpus of malware and incorporating profiles of normal behavior into our system. We also plan to solicit beta testers for a free Internet distribution of the Host-Rx prototype that would help evaluate the system for false positives.

References

[1] S. Andreassen, R. Hovorka, J. Benn, K. Olesen, and E. Carson. A model-based approach to insulin adjustment. In *Proceedings of Conference on Artificial Intelligence in Medicine*, pages 239–248, 1991.

[2] S. Andreassen, M. Woldbye, B. Falck, and S. K. Andersen. MUNIN - A causal probabilistic network for interpretation of electromyographic findings. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 366–372, 1987.

[3] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, and F. Jahanian. Automated classification and analysis of internet malware. In *RAID*, 2007.

[4] M. Barreno, B. Nelson, R. Sears, A. Joseph, and J. D. Tygar. Can machine learning be secure? In *ASIACCS*, 2006.

[5] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure. In *ASIACCS*, 2006.

[6] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. Towards automated generation of vulnerability signatures. In *IEEE Symposium on Security and Privacy*, 2006.

[7] E. Carrera and G. Erdelyi. Digital genome mapping and advanced binary malware analysis. In *Proceedings of Virus Bulletin Conference*, 2004.

[8] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-aware malware detection. In *Proceedings of IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 2005.

[9] S. P. Chung and A. K. Mok. Allergy attack against automated signature generation. In *RAID*, 2006.

[10] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *Proceedings of the Symposium on Operating System Principles*, 2005.

[11] J. R. Crandall, Z. Su, and F. Wu. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of ACM CCS*, 2005.

[12] N. N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD*, 2004.

[13] E. Karim, A. Walenstein, and A. Lakhotia. Malware phylogeny generation using permutations of code. *European Research Journal on Computer Virology*, 1(2), November 2005.

[14] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *13^{th} USENIX Security Symposium*, 2004.

[15] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang. Effective and efficient malware detection at the end host. In *Proceedings of Usenix Security*, 2009.

[16] C. Kreibich and J. Crowcroft. Honeycomb–creating intrusion detection signatures using honeypots. In *2^{nd} Workshop on Hot Topics in Networks (Hotnets-II)*, Cambridge, MA, November 2003.

[17] D. C. Liu and J. Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming B*, 45(3):503–528, 1989.

[18] P. J. F. Lucas, N. de Bruijn, K. Schurink, and A. Hoepelman. A probabilistic and decision-theoretic approach to the management of infectious disease at the ICU, 2000.

[19] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. Mitchell. A layered architecture for detecting malicious behaviors. In *Proceedings of RAID*, 2008.

[20] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis and signature generation. In *Proceedings of the 12th Annual Network and Distributed Security Symposium*, 2005.

[21] Norman Sandbox. http://sandbox.norman.no.

[22] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov. Learning and classification of malware behavior. In *Proceedings of DIMVA*, 2008.

[23] M. Shwe and G. Cooper. An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, 24:453–475, 1991.

[24] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *6th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2004.

[25] U.Bayer, C.Kruegel, and E.Kirda. Ttanalyze: A tool for analyzing malware. In *EICAR*, 2006.

[26] A. Walenstein, M. Hayes, and A. Lakhotia. Phylogenetic Comparisons of Malware. Virus Bulletin Conference, 2007.

[27] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy (Vol. 5, No. 2)*, March/April 2007.

[28] V. Yegneswaran, J. Giffin, P. Barford, and S. Jha. An architecture for generating semantic-aware signatures. In *Proceedings of USENIX Security Symposium*, Baltimore, MD, August 2005.