Technical Report TR-iSecLab-0911-001

FORECAST – Skimming off the Malware Cream

Matthias Neugschwandtner[1], Paolo Milani Comparetti[1], Gregoire Jacob[2], and Christopher Kruegel[2]

[1]Vienna University of Technology, {mneug,pmilani}@seclab.tuwien.ac.at
[2]University of California, Santa Barbara, {gregoire,chris}@cs.ucsb.edu

***Abstract*—To handle the large number of malware samples appearing in the wild each day, security analysts and vendors employ automated tools to detect, classify and analyze malicious code. Because malware is typically resistant to static analysis, automated dynamic analysis is widely used for this purpose. Executing malicious software in a controlled environment while observing its behavior can provide rich information on a malware's capabilities. However, running each malware sample even for a few minutes is expensive. For this reason, malware analysis efforts need to select a subset of samples for analysis. To date, this selection has been performed either randomly or using techniques focused on avoiding re-analysis of polymorphic malware variants [40], [23].**

In this paper, we present a novel approach to sample selection that attempts to maximize the total value of the information obtained from analysis, according to an application-dependent scoring function. To this end, we leverage previous work on behavioral malware clustering [14] and introduce a machine-learning-based system that uses all statically-available information to predict into which behavioral class a sample will fall, *before* the sample is actually executed. We discuss scoring functions tailored at two practical applications of large-scale dynamic analysis: the compilation of network blacklists of command and control servers and the generation of remediation procedures for malware infections. We implement these techniques in a tool called FORECAST. Large-scale evaluation on over 600,000 malware samples shows that our prototype can increase the amount of potential command and control servers detected by up to 137% over a random selection strategy and 54% over a selection strategy based on sample diversity.

I. INTRODUCTION

Malware is at the root of many security threats on the internet. From spam, to identity theft to distributed denial of service attacks, malicious software running on compromised computers is a key component of internet crime. For this reason, analyzing malware and developing countermeasures against it has become an important aspect of security practice. New malware samples need to be analyzed to understand their capabilities and generate detection signatures, mitigation strategies and remediation procedures. Since tens of thousands of new malware samples are found in the wild each day, security analysts and antivirus vendors have to employ automated analysis techniques for this task.

Malware commonly employs various forms of packing and obfuscation to resist static analysis. Therefore, the most widespread approach to the analysis of malware samples is currently based on executing the malicious code in a controlled environment to observe its behavior. Dynamic analysis tools such as CWSandbox [3], Norman Sandbox and Anubis [13], [2] execute a malware sample in an instrumented sandbox and record its interactions with system and network resources. This information can be distilled into a human-readable report that provides an analyst with a high level view of a sample's behavior, but it can also be fed as input to further automatic analysis tasks. Execution logs and network traces provided by dynamic analysis have been used to classify malware samples [14], [35], to generate remediation procedures for malware infections [30] and to generate signatures for detecting a malware's network traffic [33].

One problem of dynamic analysis of malware is that it is resource-intensive. Panda Labs reported 63,000 new malware samples per day in 2010 with an upward trend [9]. Each of these samples needs to be executed, if only for a few minutes. Furthermore, it is relatively easy for malware authors to aggravate this problem by automatically generating even larger numbers of polymorphic variants. As a result of the limited analysis capacity, only a subset of the daily malware samples can be analyzed. Our own analysis sandbox, despite a large-scale, distributed deployment, has to discard tens of thousands of samples each day. This raises the question of which samples should be selected to best utilize the available resources.

Previous work on selecting samples for dynamic analysis [40], [23] has focused on diversity: that is, the goal is to determine whether a new malware sample, with a never-before-seen message digest, is actually just a minor variant or a polymorphic mutation of a previously analyzed sample. Results obtained with these techniques have shown that discarding polymorphic variants can reduce the amount of samples to be analyzed by a factor of over sixteen. The assumption behind this approach is that analyzing a polymorphic variant of a known sample will not provide any new insight, so the sample should be discarded rather than waste resources on executing it in the sandbox. Depending on the purpose for which samples are being analyzed, however, this assumption may not hold.

One motivation for operating a malware analysis sandbox is that the network behavior of malware can reveal the

command and control (C&C) servers used by bot masters to remotely control infected computers. If the goal is to detect C&C servers, running multiple variants of *some* malware families can prove advantageous. This is a consequence of the constant arms race between bot masters and security professionals: bot masters need to maintain control of their bots while security professionals work to identify and take down their C&C infrastructure. As a result, the C&C servers used by a malware family change over time much faster than its code-base. Furthermore, some malware code-bases are available to multiple independent bot masters, each of which uses a distinct C&C infrastructure [15].

Rather than selecting samples for analysis based only on diversity, we therefore take a different angle and explicitly try to select for analysis those samples that will produce the most valuable analysis results. This requires us to first define how to measure the value of the output of a dynamic analysis run. We argue that this is application-dependent, and that sample selection should take into account the goals for which malware is being analyzed in the first place.

To predict whether and to what extent the execution of a sample will yield useful information, we take advantage of knowledge gleaned from samples that have already been analyzed. We extract all statically-available information on each malware sample, including structural features of the executable, antivirus detection results and the results of static classification using techniques from Wicherski [40] and Jacob et al. [23]. For all dynamically analyzed samples, we further record their behavior in the sandbox and the results of behavioral clustering using techniques from Bayer et al. [14]. Over time, our system thus assembles a knowledge-base of static and behavioral characteristics of malware samples. This knowledge-base can be mined for sample selection. We use the static information on each sample as input to a machine-learning system that aims to predict to which behavioral cluster a sample belongs, *before* we actually run the sample. We then select for execution samples expected to belong to clusters that, based on past performance, are most likely to provide useful information.

We implement the proposed techniques in a tool called FORECAST, and empirically evaluate its performance using a large collection of real-world malware binaries. Our results show that selecting samples for analysis using FORECAST can provide more useful information for a given amount of analysis resources compared not only to naive, random selection, but also to a selection strategy aimed at maximizing diversity. In summary, our contributions are the following:

- We formulate the sample selection problem as the task of choosing samples for dynamic analysis to maximize the aggregate value of the analysis results.
- We introduce novel techniques that allow us to predict the dynamic behavior of a malware sample before executing it. More precisely, they allow us to predict the behavioral cluster [14] to which the sample will belong.
- We introduce scoring functions for measuring the value of information obtained from dynamic analysis that are targeted at two practical applications; Namely the generation of network blacklists of command and control servers and the generation of procedures for the remediation of malware infections on end hosts.
- Based on these techniques, we develop a system for selecting samples for dynamic analysis according to the expected value of the information obtained from a sample's execution.
- We evaluate the proposed techniques on over 600,000 malware samples, and show that they can increase the total value of the information obtained from dynamic analysis by 134% compared to a random selection strategy and by 54% compared to a selection strategy based on sample diversity.

II. SYSTEM GOALS AND APPROACH

The goal of FORECAST is to increase the insight that can be gained from executing malware samples in an analysis sandbox, given a limited amount of computational resources. If insufficient resources are available to analyze all the malware samples that are collected each day, sandbox operators are faced with the choice of which samples to select for analysis.

We call this the *sample selection* problem, and formulate it as follows. Given a set χ of n malware samples, a scoring function v that measures the aggregate value of the analysis results of a set of samples and limited resources that allow the dynamic analysis of only $k < n$ samples, we want to select a subset $\alpha \subset \chi$, with $|\alpha| = k$, that maximizes $v(\alpha)$. The set α can be built incrementally: When selecting the next sample for analysis, we can take into account the analysis results for all previously analyzed samples.

Previous work [40], [23] has implicitly attempted to solve this problem by recognizing and discarding minor variants or polymorphic mutations of previously analyzed malware samples. The assumption behind this is that every such variant will exhibit the same behavior, therefore analyzing more than one variant provides no additional valuable information. Because it lacked a measure of the value of analysis results, previous work did not attempt to quantitatively validate this assumption. As we will show in Section IV, discarding minor variants is indeed a good heuristic for selecting samples. However, we will also show that in some cases executing several almost identical samples can provide valuable information, such as the different C&C servers contacted by each sample.

A. Applications

In this paper, we take a different approach and explicitly measure the value of analysis results. For this, we develop scoring functions targeted at two real-world applications.

Identifying C&C servers. Modern malware uses a command and control (C&C) infrastructure that allows the mal-

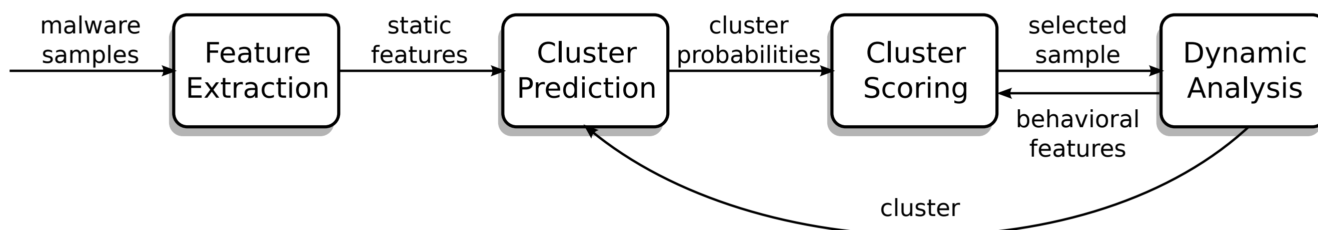

Figure 1. FORECAST overview.

ware operators to remote-control the infected machines (also known as bots). It also lets them update the bots' software to adapt to the changing environment in which they operate and to the changing goals of the botnet owners. While some botnets employ peer-to-peer protocols for C&C, most employ client-server architectures and rely on redundancy and fallback mechanisms to provide robustness. The C&C servers are therefore a weak point of a botnet's operation, making information on their domain names or IP addresses extremely valuable to security practitioners. Recent research has thus focused on identifying C&C communication among the network traffic generated by malware [22]. Such information has been used for coordinated takedowns of a botnet's C&C servers, that in some cases have succeeded in completely shutting down a botnet [29]. In a few cases, C&C server information has even led to the networks of malicious internet service providers being depeered from the internet [27]. Even if the malicious servers cannot be taken down, blacklists of C&C servers such as the one provided by FIRE [38] or Zeus Tracker[8] can be used by network administrators as an additional layer of defense. A C&C blacklist provides two benefits: on the one hand, it prevents infected hosts from receiving commands that would lead them to engage in harmful behavior; On the other hand, it can alert a network administrator to the presence of infected hosts on his network. FIRE builds its C&C blacklist based on the results of large-scale dynamic analysis of malware samples with the Anubis [2] sandbox. Furthermore, the C&C traffic captured during malware execution can be used to automatically generate detection signatures [33], that can be deployed on network intrusion detection solutions. The *network endpoints* scoring function discussed in Section III-D is therefore designed to measure the number of potential C&C servers observed during analysis.

Generating remediation procedures. When malicious code obtains unrestricted execution privileges on an infected host, completely reinstalling the affected machine is typically the only sound way of guaranteeing that the malware is fully eradicated. For a given, known malware, however, it may be possible to generate a reliable remediation procedure that is able to revert the effects of the malicious code on the system and avoid the cost of reinstallation. Remediating malware infections is a task routinely performed by anti-virus software, with varying levels of success. Recent research has proposed techniques for automatically generating such remediation procedures [30]. These techniques are based on dynamic analysis: malware samples are executed in an instrumented environment, and all persistent modifications of the system state are recorded. A remediation procedure essentially consists of a list of affected system resources, that have to be reset to a clean state. While the techniques in [30] include methods for generalizing the observed behavior to some extent, it is clear that behavior that was never observed cannot be remediated. To provide a more complete set of remediation procedures, it is therefore desirable to observe the widest possible variety of system-modifying behavior. The persistent modifications scoring function discussed in Section III-D is therefore designed to measure the amount of distinct system resources affected by malware execution.

B. System overview

Figure 1 shows a high-level overview of FORECAST's architecture. FORECAST works in four phases:

Feature extraction. For each sample that is being considered for analysis, we first extract a number of *static features*. These features represent all the information we can efficiently obtain about a malware sample without executing it. We consider a wide variety of static features. First of all, we extract a number of structural features about the malicious executable. We consider information on the origin of the malware sample, such as the user responsible for its submission to the analysis sandbox. We also include detection results from a number of anti-virus engines. Finally, we leverage previous work on detecting polymorphic malware variants and include a sample's peHash [40], as well as its static cluster and packing level obtained using techniques from Jacob et al. [23].

Cluster prediction. The dynamic analysis phase (discussed below) identifies the behavioral cluster to which each executed sample belongs. Together with the static features, this serves as input to the cluster prediction phase. Here, we attempt to predict to which behavioral cluster a sample

belongs, using a supervised learning approach. For this, we use a confidence-weighted linear classifier that outputs the probabilities that a considered sample belongs to each behavioral cluster. Whenever a sample is dynamically analyzed and assigned to a cluster, our classifier is updated to account for this new information. Note that, while cluster prediction uses supervised learning, the behavioral malware classification techniques we employ are unsupervised [14], and new behavioral clusters are added incrementally as they are discovered.

Cluster scoring. In this phase we measure the *cluster score* for each behavioral cluster C, defined as the average contribution of a sample in that cluster to the scoring function $v(C)$. This step is therefore dependent on the choice of an application-specific scoring function. The scoring function takes as input the *behavioral features* observed during the execution of each sample, and measures their aggregate value. We employ two scoring functions targeted at the applications discussed in Section II-A. For each sample that is considered for analysis, we can then compute its expected contribution to v based on the cluster scores and the cluster probabilities obtained in the previous phase. We call this the *sample score*. The output of the cluster scoring phase is the highest scoring sample, out of a pool of candidates, that is passed to the dynamic analysis phase. Whenever the dynamic analysis results for a sample become available, the cluster scores and sample scores are incrementally updated.

Dynamic analysis. Once a malware sample has been selected, we analyze it by running it in our instrumented sandbox. The host-level and network-level behavior observed during execution is condensed into a set of behavioral features. This set is fed back to the cluster scoring phase. Furthermore, we cluster all analyzed samples based on the set of behavioral features they exhibit, using techniques from Bayer et al. [14]. The behavioral cluster to which each analyzed sample belongs is fed back to the cluster prediction phase.

III. System Description

Figure 2 provides a more detailed view of ForeCast's architecture. ForeCast takes as input a set χ of *candidate* malware samples. The system's goal is to select for analysis a subset $\alpha \subset \chi$. ForeCast works incrementally: when selecting the next samples for analysis, it takes into account the analysis results for all previously analyzed samples. For this, ForeCast maintains an analysis queue where each not-yet-analyzed candidate sample (that is, each sample in $\chi \setminus \alpha$), is associated with a *sample score*. The sample score is a measure of how much valuable information we expect to obtain from the analysis of that sample. Of course, to achieve high throughput, malware analysis sandboxes need to analyze several samples in parallel. Thus, at each iteration ForeCast selects the L top-scoring samples for analysis, where L is the parallelism level of the sandbox, indicating the number of samples that it is able to analyze in parallel.

A. Dynamic analysis and clustering

Each sample selected for dynamic analysis is executed in an instrumented sandbox environment for a fixed amount of time (currently four minutes). The output of dynamic analysis is the set of behavioral features β_s observed during the execution of s. Behavioral features are a representation of program behavior introduced in [14]. Each behavioral feature represents a specific action performed by the analyzed program on a specific operating system or network entity, such as the creation of file `C:\system32\ svcshost.exe`, or an HTTP request to `www.example.com`. In the behavioral clustering phase, β_s is fed to the scalable behavioral clustering techniques from Bayer et al. [14]. As a result, we identify the behavioral cluster C_s to which s is assigned.

The cluster prediction phase receives the label C_s and incrementally updates its classifier. Furthermore, β_s and C_s are fed back to the cluster scoring phase, where the *cluster score* for cluster C_s is updated based on the value of the observed behavior β_s with respect to the application-specific scoring function v. As a result, the sample scores for all samples are incrementally updated, and the next sample can be selected for analysis.

B. Feature Extraction

The goal of the feature extraction phase is to collect all the information on malware samples that can be used to classify it. Therefore, in this phase we extract all the characteristics of each malware sample that can be efficiently obtained from static analysis. To distinguish them from the behavioral features resulting from the dynamic analysis of a sample, we refer to these characteristics as *static* features. The output of this phase is, for each candidate sample s, a set of static features ϕ_s. The feature space $\Phi = \cup_{s\in\chi}\phi_s$ grows as new candidate samples are processed and new static features are discovered.

peHash. Unlike previous approaches to sample selection, ForeCast assumes that minor variants of previously analyzed malware samples are worth analyzing. Knowing that a candidate sample is similar to previously analyzed samples can be extremely useful when attempting to predict its behavior. Therefore, we include a sample's peHash [40] in the static features. peHash uses structural information from an executable file's headers and the Kolmogorov complexity of code sections to compute a hash value that should remain constant across polymorphic malware variants.

Static cluster. Similarly, we take advantage of static malware clustering techniques from [23]. These techniques compute the distances between the code signals – essentially a bigram distribution over a code section – of binaries to group them into clusters. The authors also introduce

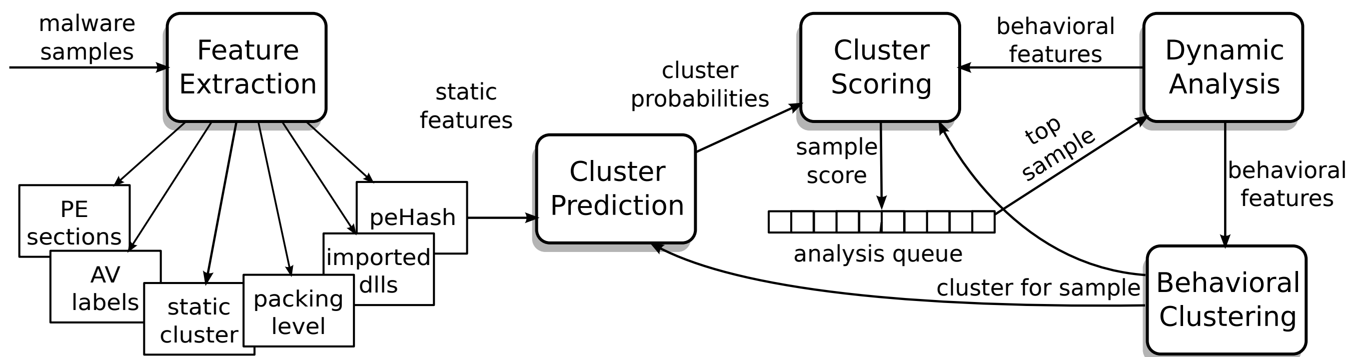

Figure 2. FORECAST architecture.

techniques to statically detect the level of packing used by a sample. We call these clusters *static* clusters. Both of these characteristics – *packing level* and *static cluster* membership – are mapped to individual static features.

PE Header. A variety of information from a binary's Portable Executable (PE) header is already processed by peHash and [23], and is thus already represented in the feature space. We extract two additional groups of features from the PE header. These are the names of the *imported DLLs* and the *PE section names*.

Antivirus Labels. We obtain anti-virus results for all candidate samples from the VirusTotal service [7]. VirusTotal scans each sample using 39 AV engines, and for each detection provides the name of the engine that detected the sample and the label it assigned to it. To avoid an excessively large number of AV label features we discard the most specific part of the AV labels (indicating the malware variant) and normalize the labels to a canonical form – e.g. `Trojan/Downloader.Carberp.n` becomes `carberp`.

Submitter Information. For each sample that is submitted to the sandbox, the hostname of the machine it has been submitted from is logged. Depending on how a malware analyst collects samples and on the types of malware she is interested in, the samples she submits may be skewed towards specific malware classes. We therefore map each distinct hostname from which a sample was submitted to a static feature.

C. Cluster Prediction

The goal of the cluster prediction phase is to predict to which behavioral cluster a candidate sample belongs, before executing it. Cluster prediction therefore aims to establish a mapping between a sample's *static* features ϕ_s and the behavioral cluster C_s to which it will be assigned based on its *behavioral* features. For this task, we take a supervised learning approach and train a classifier by providing it with a labeled dataset consisting of (ϕ_s, C_s) for each sample s that has already been subject to dynamic analysis and clustering. For this, we require an appropriate classification algorithm, considering our requirements and the properties of our datasets:

- High-dimensional feature space: $|\Phi|$ can be very large.
- Sparse data: Every sample only exhibits a small subset of the possible features ($|\phi_s| << |\Phi|$). Individual features may occur infrequently.
- Incremental operation: The results of dynamic analysis and clustering should have an immediate effect on following predictions. Furthermore, the size of the feature space and the number of clusters may change over time.
- Fast prediction: We need prediction results on all candidate samples before we can select the best ones for analysis.

For FORECAST we use linear classification in combination with the confidence-weighted (CW) learning scheme of Dredze, Crammer and Pereira [18].

Linear classification. A set of static features is represented as a binary feature vector $\vec{x}$, where every possible feature is either present (one) or absent (zero) for a specific sample. A linear classifier determines a margin y for a given feature vector $\vec{x}$ by computing the scalar product with a weight vector $\vec{w}$: $y = \sum_i x_i w_i$. The linear classification process can be visualized as splitting the feature space into two sections with a hyperplane that is determined by the weight vector $\vec{w}$, where the sign of the margin y tells us on which side of the hyperplane an input vector is, thus distinguishing two classes. The absolute value of the margin $|y|$ can be interpreted as the confidence in this classification.

Linear classifiers can handle high dimensional feature spaces and perform especially fast on sparse binary feature spaces. This is because for binary spaces, the computational cost of computing the scalar product $\vec{w} \cdot \vec{x}$ is proportional to $|\phi|$, rather than to the total number of features in the feature space $|\Phi|$.

Confidence-weighted learning. The effectiveness of a linear classifier depends on the algorithms used to train it. In online learning, the training instances are supplied one after the other. Training algorithms for linear classifiers use update rules that adjust the weights at each iteration t based on the current weights and a function g of the training instance features and label: $\vec{w}_{t+1} = \vec{w}_t + g(\vec{x}_t, y_t)$. If the feature space grows, weights can simply be added to the classifier.

Sparse data may pose a problem to such an algorithm. The reason is that typically weights are updated only if the corresponding feature is set in $\vec{x}$. Therefore, the weights for features that occur seldom are based on much less information than weights for frequently-occurring features. CW learning addresses this problem by maintaining confidence information for each feature depending on how often it occurred in training. Confidence is modeled with a Gaussian distribution $\mathcal{N}(w_i, \Sigma_i)$ for each feature i. The larger Σ_i is, the smaller is confidence in w_i and the more aggressively the distribution will be updated by g. For a full discussion of this classifier and the update function g we refer the reader to [18].

Multiclass prediction. The linear classifiers we have discussed so far can distinguish two labels. FORECAST however needs to predict to which of several behavioral clusters a sample belongs. Such a multi-class problem can be decomposed into multiple binary problems. These are then solved by a network of n binary classifiers. As a last step, a single multi-class label has to be derived from the n binary results. For this, we can use Error Correcting Output Codes (ECOC) [10]. Considering the results of the n classifiers as an n-length binary codeword, the distances between the codeword from the classification and the codewords of each label are calculated. The prediction result is the label for which the distance is minimized.

One way of performing multi-class classification using ECOC is to use a binary classifier for each pair of labels, where each classifier is trained to distinguish between these two labels. This approach is not applicable for FORECAST, because it would require us to train $|C|^2/2$ classifiers. With over a thousand clusters, this is clearly problematic. This approach is also referred to as pairwise coupling [20] or one-versus-one (OVO) classification. Instead we use a one-versus-all (OVA) approach, which requires only a single classifier per label. Each classifier is trained to distinguish between its assigned label and the "rest", i.e. all other labels. For each training instance the binary classifier of the correct class is trained with $+1$ and all other classifiers with -1.

FORECAST's cluster prediction uses one-versus-all CW learning. An advantage of this choice is that, whenever a new behavioral cluster C emerges from the analysis results, we can add a new classifier and train it to recognize C. For this, we train the new classifier with all past samples. The classifiers for all other samples, however, do not need to be modified. The weights used for classification are stored in a matrix w, where $w_{C,i}$ is the weight of feature i for cluster (classifier) C.

Probability estimates. Just picking the top predicted cluster and proceeding with the cluster scoring would mean to discard important information – the confidence in the prediction, given by the margins of each classifier. Based on that output it is possible to calculate probability estimates for each label. For ECOC models, a generic approach is given in [21]. Here we use a simple approach from the LIBLINEAR project [19] for OVA classification. The probability estimate for label C is computed as an exponential function of the margin y_C of the corresponding classifier $\frac{1}{1+e^{-y_C}}$.

Cluster size threshold. As we will show, the clusters in our behavioral malware clustering vary a lot in terms of size, with a few very large clusters and a large number of clusters containing only a few samples. Clearly, we do not want to train tens of thousands of OVA classifiers to recognize clusters that contain only one sample. First of all, this would significantly slow down FORECAST. Furthermore, a classifier that was provided during training an extremely small number of positive training instances would be unlikely to provide good results. Therefore, we select a minimum cluster size threshold θ and group all samples belonging to clusters with $|C| < \theta$ in a single "other" cluster O. We do not, however, train a classifier for O. The reason is that samples in O have nothing in common and thus their static features will vary wildly. As a probability estimate for O we therefore simply use the ratio $|O|/|\alpha|$ of the number of samples in O to the total number of analyzed samples.

As a final step all probability estimates are normalized so that they add up to one. The output of the cluster prediction phase is the cluster probability matrix p, where $p_{s,C}$ represents the probability that sample s belongs to behavioral cluster C.

D. Cluster Scoring

Given a measure of the value of the analysis results produced by the sandbox, we can evaluate which behavioral clusters produce more valuable results, and try to select analysis samples that will likely fall into those clusters. For this, we calculate a *cluster score* for each behavioral cluster.

As discussed in our formulation of the sample selection problem in Section II, the value of analysis results is measured by an application-dependent scoring function $v(\alpha)$, which computes the aggregate value of the analysis results for the set of analyzed samples α. As discussed in Section III-A, the results of dynamic analysis of a sample s is the set β_s of behavioral features observed during the sample's execution. The scoring function is therefore calculated as $v(\alpha) = v(\{\beta_s : s \in \alpha\})$. Depending on the target application, only a subset of the observed behavioral features may be of interest. For instance, if the goal of

analysis is to identify malware C&C servers, only features related to network traffic are relevant, while features representing interaction with the local system can be ignored. The scoring functions we considered in this work simply measure the total number of relevant behavioral features observed. For this, each scoring function has an associated filter $f(b) \rightarrow \{0, 1\}$ that returns 1 if feature b is relevant. We then compute $v = |\{b \in B : f(b) = 1\}|$, with $B = \bigcup_{s \in \alpha} \beta_s$. Note that assigning different weights to different behavioral features, to express the fact that some features may be more valuable than others, would be a straightforward generalization of this approach.

In this work, we use the following two scoring functions, targeted at the applications discussed in Section II-A.

Identifying C&C servers To assist in the task of identifying and blacklisting C&C servers, we introduce the *network endpoints* scoring function. This scoring function aims to measure the number of potential C&C servers contacted during dynamic analysis. The features relevant for C&C server detection are those indicating network communication with an IP address, and those indicating a DNS request for resolving a domain. A simple scoring function could therefore count the number of distinct network endpoints (IP addresses and DNS names) contacted by the analyzed samples. However, not all network traffic observed is related to C&C, or to malware's auto-update functionality (which can be seen as a type of C&C where commands are delivered in the form of executable code). Therefore, we employ a number of additional filters to make the network endpoints scoring function a more reliable measure of the amount of C&C servers contacted.

- *fast-flux*: C&C infrastructure is sometimes hosted on fast-flux networks, where the same domain name resolves to a rapidly-changing set of IPs. These IPs typically belong to infected hosts that temporarily serve as C&C servers. In this case, the information on the contacted IP addresses is of little value. Therefore, we consider IP addresses for the network endpoints score only if the malware did not obtain them from a DNS query.
- *portscan*: Many malware samples include self-propagation components that scan the internet for targets before attempting to compromise them. Clearly, we do not want to include these hosts in the network endpoints score. For this, we first discard connections to a small number of ports that are known to be typical exploit targets, such as TCP ports 139 and 445, used by Windows file and directory sharing services. Furthermore, we use the Bro IDS [31] to detect port and address scans, and discard connections that are part of detected scans.
- *liveness*: C&C endpoints that are not actually available are of little interest. We therefore filter out endpoints that could not be successfully contacted by the malware. To decide on whether a C&C endpoint is live, we rely on the known semantics of a few protocols commonly used for C&C, and fall back to a default heuristic for other kinds of traffic:
  - HTTP: we only consider servers that responded with a successful status code (that is, 200-299) to at least one request.
  - IRC: we only consider a server if the malware sent or received a private message or if it successfully joined a channel and sent or received a message on that channel.
  - FTP: we only consider a server if the malware successfully logged in.
  - Other: we only consider endpoints where a connection was successfully established (for TCP) and where the server sent back actual payload.
- *clickbots*: Clickbots are malware samples that include functionality to automatically visit advertisement links on target websites. Their goal is to fraudulently generate advertisement revenue for these websites. Due to the dynamic and tiered nature of advertisement networks, this can lead to a significant number of network endpoints being contacted during analysis. Since these endpoints are not related to command and control, we filter most of them out by using an existing list of ad-related domains that is manually maintained for the purpose of blocking advertisement [1].

Generating remediation procedures. To assist in the task of generating remediation procedures for malware infections, we introduce the *persistent changes* scoring function. This scoring function measures the number of distinct system resources affected by malware execution. As in [30], we take into account modifications to the file system and the windows registry. Furthermore, we also consider the processes and services started by the malware, because remediation procedures, if they need to be applied to a running system, will need to make sure that the malicious code is not running. Each system resource is identified by its name. The persistent changes score therefore counts the total number of distinct names of file and registry keys that are created or modified, as well as the processes and services started. Furthermore, our dynamic analysis phase uses techniques from Bayer et al. [14] (that are based on dynamic taint analysis) to detect randomly generated file names, and replace them with a special token. Likewise, we detect and replace names that are obtained by enumerating directories and registry keys. The idea is that a malware that generates a different random or temporary file in each execution, or one that crawls the entire file-system, infecting all the executables it encounters, should not be assigned a high persistent changes score because of this.

Once a scoring function is selected, we can measure the total value $v(C)$ of the information provided by the analysis of samples in a cluster C. We then calculate $cluster_score_C = v(C)/|C|$. The cluster score is thus a value to cost ratio: The amount of valuable information

provided by the cluster, divided by the analysis resources that have been spent to obtain it.

To calculate the sample scores for a sample s, we proceed as follows. Rather than simply consider the most likely cluster, we take into account the entire cluster probability vector p_s. The sample score is therefore the expectation of the cluster score, calculated as the scalar product $sample_score_s = p_s \cdot cluster_score$. At each iteration of FORECAST, the L candidate samples with the highest sample score are passed to the dynamic analysis phase.

E. Online Operation

As a result of the analysis of a selected sample s, the set of observed behavioral features β_s becomes available. These in turn are passed on to the behavioral clustering phase, which determines the cluster C_s. This newly obtained information then needs to be incorporated into our knowledge-base. Thus, we update the cluster score for cluster C_s. Furthermore, the cluster label C_s for the newly analyzed sample is fed back to the cluster prediction phase for learning. Therefore we update the classifier's weight matrix w.

At the next iteration of FORECAST, before selecting the next L samples, we recompute the probability matrix p for all remaining candidate samples, and recompute their sample scores. For this, we need to repeat the prediction step for each sample in $\chi \setminus \alpha$. The actual size of χ depends on how FORECAST is deployed in practice. Our simulation results in Section IV-C are obtained using one day of malware samples from a large-scale sandbox deployment as candidate set. In this scenario, χ can contain tens of thousands of samples. Thus, this step is one of the more computationally expensive in FORECAST's operation. However, we only need to perform this step once every L samples. As we will show in Section IV-D, for realistic levels of parallelism this leads to more than acceptable performance.

IV. EVALUATION

To develop FORECAST, we used a dataset of malware samples analyzed by an analysis sandbox in the months of October, November and December 2008. The resulting dataset consists of 100,408 samples. To evaluate FORECAST we use a larger, more recent dataset, that includes all the samples analyzed in the months of July, August and September of 2010. This 2010 dataset consists of 643,212 samples.

Table I shows an overview of the distribution of static features among feature groups for the 2010 dataset. We can see that the feature-space is large, with over 700 thousand distinct features. However, the mean number of features for each sample is only 21. For several of the feature groups, such as the peHash or static clusters groups, each sample is in fact assigned exactly one feature.

Table I
DISTRIBUTION OF FEATURES IN THE 2010 DATASET.

Feature group	Total	Average
PE section names	42389	4
Imported DLLs	12386	5
peHash	218380	1
Static clusters	203078	1
Packing Level	4	1
AV labels	250852	8
Submitter information	1909	1
Total	729007	21

We determined FORECAST's parameters by testing our system on the 2008 dataset. For the behavioral clustering, we use the same parameters and distance threshold employed in [14]. Recall that FORECAST trains a OVA classifier for each cluster C such that $|C| \geq \theta$. The samples in smaller clusters are instead assigned to the other cluster O. For our experiments, we selected $\theta = 20$. Decreasing θ beyond this point leads to a slow decrease of $|O|$, while causing a sharp increase in the number of behavioral clusters.

A. Cluster Prediction

Table II
CONTRIBUTION OF FEATURE GROUPS TO CLUSTER PREDICTION ACCURACY FOR THE 2010 DATASET

Feature group	Prediction Accuracy	
	Using all but FG	Using only FG
None	68%	-
PE section names	67%	45%
Imported DLLs	66%	40%
peHash	66%	45%
Static clusters	66%	45%
Packing level	68%	-
AV labels	65%	52%
Submitter information	68%	-

To assess the accuracy of FORECAST's cluster prediction, we train FORECAST's classifier on the samples from the first month and measure its accuracy on those of the last two months. Since FORECAST is incremental and adapts with every sample that is analyzed, a dedicated training set is not strictly necessary. However, it is still desirable to bootstrap the system on some initial data, so that predictions can be based on reasonable knowledge. For the 2010 dataset, the training set of July 2010 consists of 193,726 samples while the testing set of August and September 2010 includes 449,486 samples. Samples are processed for prediction and training in chronological order.

The 2010 dataset includes a total of 1303 behavioral clusters, including the other cluster O. The first line of Table II shows the cluster prediction accuracy when using all static features. For 68% of the 449,486 samples, our classifier assigned the highest probability to the correct behavioral cluster (out of the 1303 behavioral clusters). The following lines show the classifier's accuracy if it is

trained *without* features from one of the groups listed in Table I (left column), and if it is trained using *exclusively* features from a single group (right column). Table II thus provides some insight into the contribution of each feature group to the classifier's prediction accuracy. We can see that removing any single feature group does not cause large drops in accuracy. The reason is that features from the different groups are highly correlated. For instance, peHash and the static clustering from [23] lead to similar classifications of malware binaries. Therefore, removing the peHash features causes only a modest decrease in accuracy, because the static features provide similar information. Likewise, samples in each static cluster are typically assigned only a handful of different AV labels. Nonetheless, the right column shows that none of the feature groups, on their own, are sufficient to obtain comparable classification accuracy.

B. Cluster Scoring

The scoring functions proposed for FORECAST are designed to assist the selection of samples for specific analysis goals, by measuring the value of the information provided by an analysis run. It is important to verify that the proposed scoring functions indeed encourage the selection of samples that are relevant to the analysis goals. For this, for each scoring function, we rank the 1,303 clusters in the 2010 dataset by their cluster score $v(C)/|C|$, and manually assess some of the highest- and lowest-scoring clusters, as well as some of the largest clusters in the dataset.

The network endpoints scoring function is designed to encourage the analysis of samples that are likely to reveal new C&C servers. Therefore, we would expect the highest-ranked clusters for this score to belong to bots and other remote controlled malware, and especially to malware families that use a highly redundant or dynamic C&C infrastructure. Table III shows significant clusters of the dataset ranked by the network endpoints score. Here, the feature count is the total number of potential C&C servers detected in this cluster.

Indeed, almost all top-ranked clusters belong to remote controlled bots or trojans. Furthermore, several of these are associated with malware families that are known to use highly dynamic C&C infrastructure. One example is the Pushdo/Cutwail botnet (discussed extensively in [17]), found at rank three. Pushdo binaries contain a frequently-updated list of IP addresses. The malware contacts these addresses over HTTP to download a binary payload: typically an instance of the Cutwail spam engine. Cutwail then proceeds to obtain templates and instructions for spamming from other C&C servers over a custom binary protocol. Likewise, the Koobface botnet, at rank 28, is known to use compromised web pages as part of its C&C infrastructure. Bredolab, at rank 15, is a downloader similar to Pushdo [37].

Vundo/Zlob at rank eight, is a Fake-AV downloader. These samples download binaries from a number of different servers. This is a representative example of cases where a diversity-based selection strategy would cause the analysis sandbox to miss relevant behavior. The reason is that the 26 samples in this cluster have only two distinct peHash values. Discarding the remaining 24 samples would cause most of the 19 C&C servers to remain undiscovered.

In a few cases, however, we assign a high network endpoints score to samples that we are not necessarily interested in. The top non-relevant cluster is the Adrotator cluster at rank six. The samples in this cluster are clickbots: They visit advertisement links to fraudulently generate advertisement revenue. In this case, some of the advertisement servers in questions are not in our adblock blacklist, therefore they contribute to the network endpoints score.

A total of 171.533 samples belong to clusters with score zero. Among these an Allaple cluster can be found, which performs network scans that are filtered by the scoring function as well as another cluster performing no network activity at all.

Table III
SELECTED CLUSTERS RANKED BY NETWORK ENDPOINT SCORE

Rank	Size	Feature Count	Malware Family
1	36	84	Unknown Bot
2	20	20	Harebot
3	29	26	Cutwail
...			
6	31	24	Adware Adrotator
8	26	19	Vundo / Zlob
15	67	32	Bredolab
28	141	51	Koobface
36	82	27	Swizzor
67	173	33	zBot
199	121272	4027	"other" Cluster
273	189285	2596	Unknown Downloader
...			
Last	16370	0	Allaple/Rahack
Last	28179	0	No activity

With the scoring function for persistent changes we aim to gather information that is useful for the creation of remediation procedures. Table IV shows a selection of significant clusters ranked by this scoring function. The feature count here tells us how many persistent changes were discovered on samples in this cluster.

Several small clusters at the very top are variants of Sality, which is a polymorphic file virus that has evolved since 2003 and is now used as a basis for bots, adware and other sorts of malware. The high feature count, in this case, was caused by the creation of a large number of registry keys with random names. Our system currently supports detecting the random generation of file names, but not the less common case of registry keys. For the same reason, these samples with otherwise similar behavior were split across several smaller clusters.

Apart from the Sality clusters, this scoring function worked well and the remaining highly-ranked clusters contain malware for which remediation procedures can be created. Zegost, Nebuler and Swisyn all use different persistent modifications to hide, persist on the infected system and survive reboot. Each of them varies the infected resources to some extent across the different samples. Therefore, multiple analysis results are needed before comprehensive remediation procedures can be generated.

Samples from the Zegost cluster first unpack themselves into a new file in the root directory and launch it. The new process deletes the initial file and creates a DLL in a program file folder. Then various existing system services are hijacked by replacing the registry entries for the service handler with the previously created DLL. Afterwards these services are launched. All of these persistent modifications could be used to create a remediation procedure that can effectively remove this malware from an infected system.

Table IV
SELECTED CLUSTERS RANKED BY PERSISTENT CHANGES SCORE

Rank	Size	Feature Count	Malware Family
1	28	1876	Sality
2	25	1300	Sality
3	36	1785	Sality
...			
8	20	222	Zegost
20	48	362	Swisyn
88	205	783	Nebuler
...			
269	121272	219909	“other” Cluster
906	189285	32873	Unknown Downloader
1168	28179	145	No activity
Last	32	0	Bjlog / Swizzor

C. Simulation

To assess the real-world impact of performing sample selection using FORECAST, we perform a trace-based simulation. For this, we consider all of the samples that our sandbox was able to analyze during August and September 2010, and simulate a sandbox deployment with a smaller amount of resources, that is therefore able to analyze only a specified percentage of these samples. This allows us to compare FORECAST with other sample selection strategies, and measure the effect of each strategy on the value of the analysis results. The fact that our existing sandbox is considered, for the purpose of this simulation, to have 100% capacity does not mean that it is in reality able to process all available samples. However, we clearly cannot include in our evaluation samples for which we do not have dynamic analysis results. To simulate a FORECAST deployment, we split up the last two months of the 2010 dataset according to the day on which each analysis result was produced. For each day, we then perform sample selection using five different strategies, taking into account information obtained from sample analysis during the previous days.

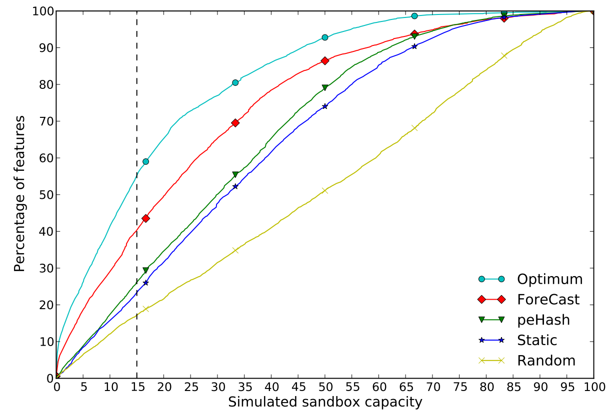

Figure 3. Daily simulation results for network features, $L = 100$

- *Random*: Randomly select the samples for analysis. This simple selection strategy provides a baseline against which other strategies can be evaluated.
- *PeHash*: This selection strategy uses a sample's peHash [40] to attempt to maximize the diversity of the analyzed samples. For this, we randomly select a sample for analysis for each distinct peHash. Once all these samples have been analyzed and if more analysis resources are available, we proceed to select a second sample for each peHash, and so on.
- *Static*: This selection strategy is similar to the previous one, but uses a sample's static cluster, identified with techniques from Jacob et al.[23], instead of its peHash. Together with the peHash selection strategy, this represents the state of the art in sample selection.
- *ForeCast*: We select samples for analysis using FORECAST. We test FORECAST with values of L (the level of parallelism) ranging between 50 and 1600.
- *Optimum*: Here we perform sample selection based on FORECAST's cluster scoring technique, but assuming that cluster prediction is 100% accurate; That is, that we know to which behavioral cluster each sample belongs, *before* executing it. Clearly, such a selection strategy is not possible in practice, but it serves as an upper bound on the benefits a sample selection strategy can bring.

Figure 3 shows the simulation results for the network endpoint scores and $L = 100$. On the X-axis, we have the capacity of the simulated sandbox relative to the real sandbox. This is the percentage of the samples from each day that the simulated sandbox is able to handle. On the Y-axis, we have the percentage of relevant features observed over the entire 61 day period. Thus, the Y-axis represents the percentage of C&C endpoints discovered by the simulated sandbox.

Table V
SIMULATION RESULTS AT 15% SANDBOX CAPACITY.

	Number of features	Percentage of features	Impr. over random
Random	1645	17%	0%
Static	2242	23%	36%
peHash	2504	26%	52%
FORECAST, $L = 50$	3900	41%	137%
FORECAST, $L = 100$	3857	40%	134%
FORECAST, $L = 200$	3899	41%	137%
FORECAST, $L = 400$	3821	40%	132%
FORECAST, $L = 800$	3825	40%	133%
FORECAST, $L = 1600$	3732	39%	127%
Optimum	5271	55%	220%

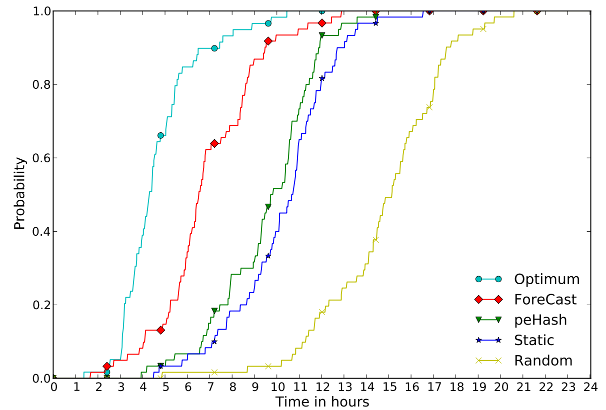

Figure 4. CDF of necessary samples to get 60% of network features, $L = 100$

We can see that FORECAST clearly outperforms the random selection strategy. To provide concrete numbers, we have picked 15% of the simulated sandbox capacity to compare the approaches, because the more limited the resources are, the more important it is which samples are selected for analysis. The results are shown in Table V. With 15% sandbox capacity and $L = 100$, FORECAST allows us to observe 134% more potential C&C servers compared to a random selection strategy. Furthermore, both peHash and static perform significantly better than random selection, by 52% and 36% respectively. This confirms the intuition from previous work [40], [23] that diversity is a good heuristic for sample selection. Nonetheless, FORECAST provides noticeable benefits compared to both strategies, outperforming peHash by 54%, and static by 72%. This demonstrates that explicitly optimizing sample selection for an analysis goal can improve the overall value of the analysis results. It is also worth noting that the optimum selection strategy outperforms FORECAST by 37%. This shows that there is some margin for improving FORECAST's performance if the accuracy of its cluster prediction component can be increased, for instance by considering additional static features or by improving the classifier. Table V also shows that higher levels of parallelism have modest negative effects on performance. With $L = 1600$, FORECAST reveals only 4% less features than with $L = 50$. Note that a parallelism level of 1600 is over an order of magnitude larger than our current deployment.

Figure 4 shows the cumulative distribution function, over the 61 days considered, of the time needed by a simulated sandbox with 100% capacity to observe 60% of relevant features. We can see that on the median day, a sandbox using FORECAST could observe 60% of features after having analyzed only about one third of the daily samples in 7.2 hours. A sandbox using the random selection strategy, on the other hand, would need 15 hours to provide the same number of potential C&C servers, while with the diversity-based sample selection strategies, about 10 hours would be needed. A secondary benefit of FORECAST is therefore the faster response to new C&C servers.

Figure 5 shows the simulation results for the persistent modification scores. The X-axis indicates the capacity of the simulated sandbox relative to the real sandbox and the Y-axis the percentage of persistent modifications observed over the entire 61 day period.

The results are similar to the ones achieved with the network scoring function. With 15% sandbox capacity, FORECAST allows us to observe 130% more persistent modifications compared to a random selection strategy. peHash and static perform significantly better than random selection, by 69% and 48% respectively. Again FORECAST provides noticeable benefits compared to both strategies, outperforming peHash by 36% and static by 55%. The optimum selection strategy outperforms FORECAST by 68%, stressing the importance of correct classification.

Figure 6 shows the cumulative distribution function for persistent features in the same way as Figure 4.

D. Performance

Table VI
FORECAST RUN-TIME ON 2010 DATASET, $L = 100$

	Total (hours)	Per Sample (s)
Feature Extraction	48	0.39
Cluster Prediction	13	0.11
Cluster Scoring	0.34	<0.01
Clustering	17	0.13
Total	78.34	0.64

Since the aim of a sample selection strategy is to more efficiently use the computing resources available for dynamic analysis, it cannot itself require excessive resources. Clearly, deciding if a sample should be analyzed must be much faster than actually running dynamic analysis on it.

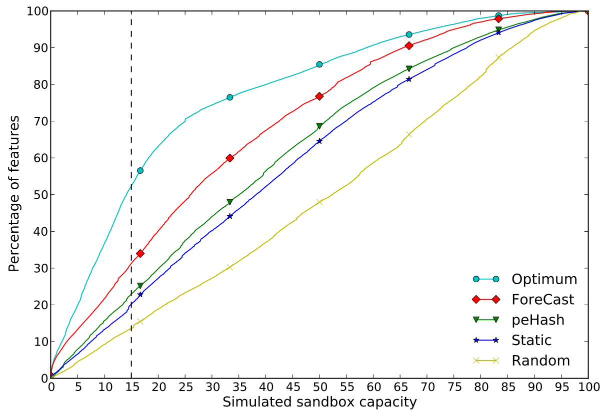

Figure 5. Daily simulation results for persistent features, $L = 100$

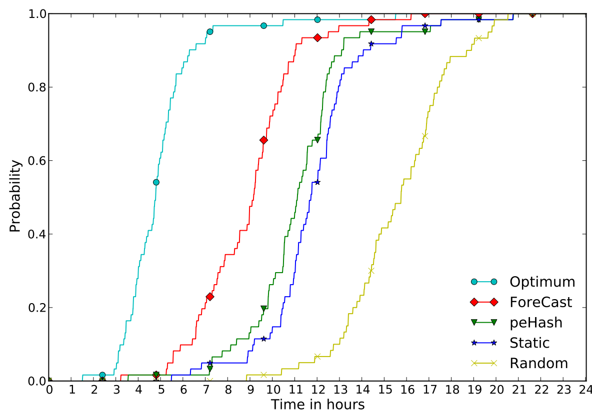

Figure 6. CDF of necessary samples to get 60% of persistent features, $L = 100$

Table VI shows FORECAST's run-time for the simulation described in the previous section on the 2010 dataset, running on a single server. As we can see, the total time per sample is under one second. This is negligible compared to the four minutes our sandbox spends executing each sample. Note that the cost of running AV engines on the samples is not included in this figure, because we obtain AV results from the VirusTotal service [7]. Performing AV-scanning with all engines supported by VirusTotal would require an additional five seconds per sample using a single machine [16].

E. Evasion

Our experiments have shown that FORECAST is effective in selecting for dynamic analysis samples that will provide useful information. However, malware authors could attempt to avoid analysis by tricking our system into *not* selecting their binaries. For this, they could attack our cluster prediction component, which ultimately relies on the static features discussed in Section III-B. A first approach would be to mutate malware samples so that our system cannot statically recognize variants as similar. For this, malware authors could develop techniques for polymorphic mutation designed to evade peHash [40] as well as static clustering [23]. Furthermore, they could try to confuse the AV companies' (proprietary) techniques for assigning names to malware variants. If such mutation techniques were successful and widespread, FORECAST would become at best useless. However, an individual malware author has little incentive to deploy such a technique, because it would not succeed in evading analysis for his samples. The reason is that our cluster prediction component would most likely assign such novel-looking samples to the other cluster O. As we can see from Table III, because of its diversity, the other cluster is ranked quite high by our cluster scoring algorithm.

Alternatively, a malware author could attempt to perform a mimicry attack, tricking FORECAST into assigning his malware to a cluster that has very little interesting behavior (such as the Allaple cluster shown in Table III). Indeed it is relatively straightforward to develop a sample that resembles a variant of the Allaple worm. However, such a sample would hardly be successful, as it would be immediately detected by most AV engines. Evading detection by AV engines while performing mimicry against FORECAST's cluster prediction (which includes AV labels among its features) would seem to be challenging.

V. RELATED WORK

There exists a large body of related work on malware detection and analysis. Currently, the most popular approach for malware analysis relies on sandboxes [13], [3], [4], [5], [6]. A sandbox is an instrumented execution environment that runs an unknown program, recording its interactions with the operating system (via system calls) or other hosts (via the network). Often, this execution environment is realized as a system emulator or a virtual machine. For each malware program that is analyzed, a sandbox will produce a report that details the host-based actions of the sample and the network traffic that it produces.

Based on the reports that capture the dynamic activity of malware programs, it is possible to find clusters of samples that perform similar actions [11], [14], or to perform supervised malware classification to detect samples from known malware families [35].

In addition to approaches that perform malware clustering and classification on the output of sandboxes, there are a number of static techniques that share the same goal but operate directly on the malware executable [26], [34], [25], [39]. Unfortunately, these tools all assume that the malicious code is first unpacked and disassembled. However, existing generic unpackers rely on the dynamic instrumentation of executables [36], [28], [24]. That is, these systems need to

execute the sample. This is a problem in our context, because we aim to avoid the overhead associated with dynamic analysis and need to pick a sample without executing the malware first.

A few tools can process packed malware samples but do not require a previous, dynamic unpacking step. Some of these tools [32] do not attempt to classify (or cluster) malware programs directly. Instead, they use static analysis only to distinguish between packed and unpacked executables. Packed executables are then forwarded to a dynamic unpacker. We are only aware of two systems that can detect duplicate malware samples using a fully static approach [40], [23]. Our system uses both of these tools to produce input that we then leverage for the cluster prediction step. As our experiments demonstrate, FORECAST significantly outperforms these systems.

Finally, [12] takes a dynamic approach to duplicate sample detection: All samples are executed in the sandbox, but after a short time-out (1 minute) the behavior so far is compared with the behavior of previously analyzed samples. If the sample is detected as a duplicate, execution is immediately terminated, otherwise analysis continues until a longer analysis time-out. The authors do not evaluate their approach with respect to an objective function. In any case, their system requires at least one minute to discard "uninteresting" samples, while FORECAST spends only 0.64 seconds on each sample, as shown in Table VI.

VI. CONCLUSION

Given the flood of tens of thousands of malware samples that are discovered every day, time is a valuable resource for a dynamic malware analysis system. Of course, the available time should be spent on analyzing samples that are most relevant, where relevance depends on the goals of the analyst and the application domain. For example, for botnet C&C analysis, one would prefer to pick samples that produce network traffic and reveal the locations of C&C servers.

In this paper, we presented FORECAST, a system that can select the malware sample that is most likely to yield relevant information, given a domain-specific scoring function. The key requirement is that this selection process has to be performed efficiently, on a possibly large pool of candidates, and without actually running a sample. To realize our approach, we use a large number of input features that are statically extracted from malware executables. These features are then fed into a predictor, which estimates the expected information gain when executing a sample. This predictor uses machine learning techniques and leverages a knowledge base built from previously-analyzed samples. Our experiments demonstrate that FORECAST is effective in selecting interesting samples. On a test set of more than 600 thousand malware samples, our system showed a high accuracy and significantly outperformed a selection strategy that simply avoids picking similar (or duplicate) samples.

VII. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec), from the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs (project i-Code), and from the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

REFERENCES

[1] Adblock List. Retrieved September 2010 from https://secure.fanboy.co.nz/fanboy-adblock.txt.

[2] Anubis. http://anubis.iseclab.org.

[3] CWSandbox. http://www.cwsandbox.org.

[4] Joebox: A Secure Sandbox Application for Windows. http://www.joebox.org/.

[5] Norman Sandbox. http://sandbox.norman.com.

[6] ThreatExpert. http://www.threatexpert.com.

[7] Virustotal. http://www.virustotal.com.

[8] Zeus Tracker. https://zeustracker.abuse.ch.

[9] Panda Labs Annual Report, 2010.

[10] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. In *International Conference on Machine Learning*, 2000.

[11] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2007.

[12] U. Bayer, E. Kirda, and C. Kruegel. Improving the Efficiency of Dynamic Malware Analysis. In *Symposium On Applied Computing (SAC)*, 2010.

[13] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Annual Conf. of the European Institute for Computer Antivirus Research (EICAR)*, 2006.

[14] U. Bayer, P. Milani Comparetti, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security (NDSS)*, 2009.

[15] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *International Conference on Privacy, Security and Trust*, 2010.

[16] J. Canto. VirusTotal Timing Information. Hispasec Systemas. Private communication, 2011.

[17] A. Decker, D. Sancho, L. Kharouni, M. Goncharov, and R. McArdle. A study of the Pushdo / Cutwail Botnet. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/Study_of_pushdo.pdf, 2009.

[18] M. Dredze and K. Crammer. Confidence-Weighted Linear Classification. In *International conference on Machine learning*, 2008.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9, 2008.

[20] T. Hastie and R. Tibshirani. Classification by Pairwise Coupling. In *Advances in neural information processing systems*, 1997.

[21] T.-K. Huang, R. C. Weng, and C.-J. Lin. Generalized Bradley-Terry Models and Multi-Class Probability Estimates. *Journal of Machine Learning Research*, 7, 2006.

[22] G. Jacob, R. Hund, T. Holz, and C. Kruegel. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *USENIX Security Symposium*, 2011.

[23] G. Jacob, M. Neugschwandtner, P. Milani Comparetti, C. Kruegel, and G. Vigna. A static, packer-agnostic filter to detect similar malware samples. Technical Report 2010-26, UC Santa Barbara, 2010.

[24] M. G. Kang, P. Poosankam, and H. Yin. Renovo: A Hidden Code Extractor for Packed Executables. In *ACM Workshop on Recurring malcode (WORM)*, 2007.

[25] A. Karnik, S. Goswami, and R. Guha. Detecting Obfuscated Viruses Using Cosine Similarity Analysis. In *Asia International Conference on Modelling & Simulation*, 2007.

[26] J. Kolter and M. Maloof. Learning to Detect Malicious Executables in the Wild. *Journal of Machine Learning Research*, 7, 2006.

[27] B. Krebs. Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. *McAfee Security Journal*, (6), 2010.

[28] L. Martignoni, M. Christodorescu, and S. Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Annual Computer Security Applications Conf. (ACSAC)*, 2007.

[29] A. Mushtaq. Smashing the Mega-d/Ozdok botnet in 24 hours. http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html, 2009.

[30] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. Giffin, and S. Jha. Automatic Generation of Remediation Procedures for Malware Infections. In *USENIX Security Symposium*, 2010.

[31] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *USENIX Security Symposium*, 1998.

[32] R. Perdisci, A. Lanzi, and W. Lee. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables. In *Annual Computer Security Applications Conference (ACSAC)*, 2008.

[33] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX conference on Networked Systems Design and Implementation*, 2010.

[34] K. Reddy, S. Dash, and A. Pujari. New Malicious Code Detection Using Variable Length n-grams. In *Information Systems Security Conference*, 2006.

[35] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and Classification of Malware Behavior. In *Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2008.

[36] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *22nd Annual Computer Security Applications Conf. (ACSAC)*, 2006.

[37] D. Sancho. You Scratch My Back... Bredolab's Sudden Rise in Prominence. http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/bredolab_final.pdf, 2009.

[38] B. Stone-Gross, A. Moser, C. Kruegel, K. Almaroth, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[39] S. Tabish, M. Shafiq, and M. Farooq. Malware Detection Using Statistical Analysis of Byte-Level File Content. In *ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 2009.

[40] G. Wicherski. peHash: A Novel Approach to Fast Malware Clustering. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.