Back to the Future: Malware Detection with Temporally Consistent Labels

Brad Miller[1], Alex Kantchelian[1], Michael Carl Tschantz[2], Sadia Afroz[1]
Rekha Bachwani[3], Riyaz Faizullabhoy[1], Ling Huang[4], Vaishaal Shankar[1]
Tony Wu[1], George Yiu[1], Anthony D. Joseph[1], J. D. Tygar[1]
[1]*UC Berkeley;* [2]*International Computer Science Institute;* [3]*Netflix*;* [4]*DataVisor*

Abstract

The malware detection arms race involves constant change: malware changes to evade detection and labels change as detection mechanisms react. Recognizing that malware changes over time, prior work has enforced *temporally consistent samples* by requiring that training binaries predate evaluation binaries. We present *temporally consistent labels*, requiring that training labels also predate evaluation binaries since training labels collected after evaluation binaries constitute label knowledge from the future. Using a dataset containing 1.1 million binaries from over 2.5 years, we show that enforcing temporal label consistency decreases detection from 91% to 72% at a 0.5% false positive rate compared to temporal samples alone.

The impact of temporal labeling demonstrates the potential of improved labels to increase detection results. Hence, we present a detector capable of selecting binaries for submission to an expert labeler for review. At a 0.5% false positive rate, our detector achieves a 72% true positive rate without an expert, which increases to 77% and 89% with 10 and 80 expert queries daily, respectively. Additionally, we detect 42% of malicious binaries initially undetected by all 32 antivirus vendors from VirusTotal used in our evaluation. For evaluation at scale, we simulate the human expert labeler and show that our approach is robust against expert labeling errors. Our novel contributions include a scalable malware detector integrating manual review with machine learning and the examination of temporal label consistency.

1. Introduction

Desktop malware constitutes an enormous arms race in which attackers evolve to evade detection and detection mechanisms react. A recent study found that only 66% of malware was detected within 24 hours, 72% within one week, and 93% within one month [10]. To evade detection attackers produce a large number of different malware binaries, with McAfee receiving over 300,000 binaries daily [19].

Machine learning offers hope for timely detection at scale, but the setting of malware detection differs from common applications of machine learning. Unlike applications such as speech and text recognition where pronunciations and character shapes remain relatively constant over time, malware evolves as adversaries attempt to fool detectors. In effect, malware detection becomes an online process in which vendors must continually update detectors in response to new threats. Malware detection also has unique labeling challenges. Whereas reading and writing are the only skills necessary to label text, reliably labeling malware requires expert analysis.

In response to the malware detection challenges, we contribute to the design and evaluation methodology of malware detectors. We present *temporally consistent labels*, requiring that training labels predate evaluation binaries since labels collected after evaluation binaries constitute knowledge from the future. Improving over random cross-validation (which makes no effort to respect time), prior work has introduced *temporally consistent samples*, requiring that training binaries predate evaluation binaries. Our analysis shows that temporally consistent labeling significantly impacts evaluation accuracy: temporally consistent samples with temporally inconsistent labels from the future are almost as misleading as cross-validation.

To realize the potential of improved training labels, we integrate a human expert labeler into the ongoing maintenance of the detector. The detector views the human as a limited resource, and selectively queries him to label difficult samples and integrates responses to boost overall system performance. In our evaluation, we use temporally consistent labels and simulate the involvement of the human expert labeler by appealing to the future labels of a sample. Our results demonstrate that by selectively querying the expert labeler, we get back to the level of performance suggested by future based labels.

Our work offers the following key contributions:

- We demonstrate the previously unstudied impact of temporally consistent labels on evaluation results. At a 0.5% false positive rate, strict temporal consistency yields a 72% detection rate, as compared with 91% and 92% detection rates according to temporally inconsistent labels and cross-validation, respectively.
- Our detector integrates a human expert labeler that increases baseline detection from 72% at 0.5% false positive rate to 77% and 89% detection with 10 and 80 oracle queries daily on average. Additionally, our system detects 42% of malicious binaries initially undetected by all other vendors in our evaluation.
- We examine our open-source learning and evaluation techniques using over 1.1 million samples appearing between January 2012 and June 2014. Our code, 3% of data, and all binary hashes will be available online.

*This author did most of her work on this paper while at Intel Labs.

Work	Domain	Malware #	Benign #	Malware Diversity	Benign Diversity	Temporal Labels	Human Integration
Schultz et al. 2001 [25]	exec	3,265	1,001	◑	◔	○	○
Henchiri et al. 2006 [13]	vir	1,512	1,488	?	◔	○	○
Kolter et al. 2006 [17]	vir+worm	1,651 / 291	1,971	?	?	○	○
Perdisci et al. 2008 [23]	exec	5,586	2,258	◔	◔	○	○
Gavrilut et al. 2009 [12]	exec	27,475	273,133	◔	◔	○	○
Ye et al. 2009 [30]	exec	31,518	8,320	◑	?	○	○
Alazab et al. 2011 [7]	exec	51,223	15,480	◑	◔	○	○
Canali et al. 2012 [8]	exec	7,200	?	◑	●	○	○
Shabtai et al. 2012 [27]	exec+dll	7,688	22,735	◔	◑	○	○
Schwenk et al. 2012 [26]	javascript	8,282	3,400,000	◑	◑	◑ †	○
Šrndic et al. 2013 [28]	pdf	82,142	577,071	●	●	○	○
Nissim et al. 2014 [21]	pdf	45,763	5,145	◑	◑	○	◑ ‡
Nissim et al. 2014 [22]	exec+dll	7,688	22,735	◔	◑	○	◑ ‡
Markel et al. 2014 [18]	exec	122,799	42,003	◔	◔	○	○
this work	**exec**	**938,954**	**113,322**	●	●	●	●

†Schwenk et al. focus on a machine learning system which retrains on predictions produced by the system itself and ignores temporally consistent labels from external sources.

‡Nissim et al. seek to maximize the number of malicious files given to the human, resulting in detection comparable to [22] or worse than [21] uncertainty sampling, a human integration technique from other application domains.

Legend

Malware Diversity	
?	No dataset characterization
○	Cleaned dataset (e.g., removed packed files) or otherwise constrained
◔	Minimal diversity characterization or single source (e.g., VX Heaven)
◑	Diverse sources
●	From consumer-facing service
Benign Diversity	
?	No dataset characterization
○	Windows OS files from fresh install
◔	Windows OS files + "popular" software
◑	Diverse sources
●	From consumer-facing service
Temporal Labels	
○	Randomized sample order or temporally inconsistent labels.
◑	Retraining on previous predictions
●	Integration of temporal label knowledge
Human Integration	
○	No human expert interaction
◑	Interaction for purpose other than improving ML malware detection
●	Interaction to improve ML malware detection

Table 1: Prior work in file-based malware detection. Domain: vir: x86 viruses, exec: x86 Portable Executable file format, dll: Windows OS library file, pdf: portable document file, worm: network worms.

Dataset. To conduct our experiments, we have amassed a dataset constructed from over one million binaries submitted to VirusTotal, a malware repository [29]. When VirusTotal receives a submission (including a resubmission) of a binary, it runs a suite of static and dynamic analyses as well as static malware detectors on it. Our corpus includes the results of each of the submissions of each of binary in our dataset to track binaries over time and to weight them by the interest in them (approximated by their submission count). In particular, we have timestamps for all submissions and corresponding detection results enabling the use of the temporal consistency of labels in our evaluation.

Evaluation. We use our evaluation method and dataset to understand the accuracy of our detector under various conditions. Our evaluation examines the impact of variations in retraining interval, detection latency and expert labeling budget on detector accuracy. Although our design includes both static and dynamic features, since VirusTotal detectors must operate statically we also compare our performance against VirusTotal using static features alone. Note that the restriction to static features actually disadvantages our approach, as VirusTotal detectors may operate against the arbitrary file and we restrict ourselves to static attributes available through VirusTotal. Our performance is slightly impacted, producing 84% detection at 0.5% false positive rate with 80 queries daily and still surpassing detectors on VirusTotal. We also explore the impact of inaccurate human labelers on the system's accuracy by adding random noise to the simulated expert labels. We find that our design is robust in the presence of imperfect labelers. Given an oracle with a 80% true positive rate and a 5% false positive rate our system still achieves 85% detection at a 1% false positive rate, as compared to 90% detection using a perfect oracle.

In Section 2, we review prior work. Section 3 presents the design of our system, including feature extraction, machine learning and integration of the labeling expert. Section 4 covers our implementation. Section 5 examines our dataset. Section 6 presents our method of evaluation while Section 7 exemines its results. Lastly, Section 8 concludes.

2. Prior Work

In this section we discuss our relationship to prior work on machine learning based malware detection. We present the first integration of a human reviewer with the purpose of increasing performance of machine learning based malware detection. We also present the first examination of integrating temporally consistent label knowledge, as contrasted with training labels obtained after evaluation or retraining a system on the system's previous predictions. Since there has been considerable prior work on malware detection, we focus our discussion below on the prior work most relevant to our primary contributions. Table 1 presents a comparison of our work with the prior work in file-based malware detection. Note that we only consider detection mechanisms based on file contents; meta-data based approaches [24] and malware family detection approaches are out-of-scope.

Temporally Consistent Labels. Schwenk et al. present the most relevant prior work on labeling by examining the performance of a malicious JavaScript detector over a five month period. Schwenk et al. compare detection performance when iteratively retraining using the system's own predictions as training labels with detection performance when training on labels generated after all data collection. By focusing on a system which retrains purely on the system's own predictions, there is no means of incorporating or analyzing the effect of temporally available knowledge in the anti-malware community. Separate from Schwenk et al., several other works have identified the problem of time lag in sample ground truth [10, 24]. In contrast with prior work, our work is the first to explore *strict* and *natural* temporal consistency in labels, training on unsimulated and strictly historical knowledge of samples and labels in order to make predictions on current and future samples.

Separate from temporal labeling concerns, few works have explored temporal sample consistency. Moskovitch et al. [20], and Kolter et al. [17] present evaluations where malicious instances are temporally sorted, so that the system is trained on historical malicious samples and tested on yet unseen malware. We note that only the malicious samples were temporally ordered, while the benign samples were not. Šrndic et al. [28] used temporally consistent malicious and benign PDF instances to evaluate their system. Moving towards temporal sample con-

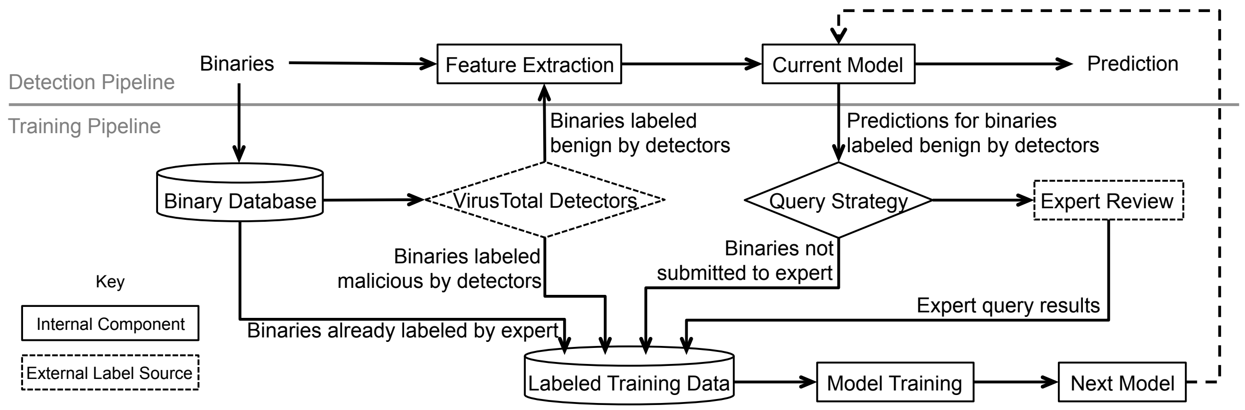


Figure 1: Detector design overview. The detection pipeline employs the current model to detect malware, and the training pipeline produces the next model for use in the detection pipeline. During each retraining period, the training pipeline reviews all available training data and selects binaries for submission to the human labeling expert. Binaries labeled by the expert are combined with training data labeled using the current model and anti-virus scan results to serve as the training data for the next model.

sistency, Henchiri et al. [13] performed family-aware cross-validation, where training and testing samples belong to disjoint sets of families, simulating apparition of novel malware at testing time.

Most prior research uses cross-validation or other equivalent blind dataset randomization schemes for evaluating their machine learning framework. Some works, e.g., Nissim et al. [22], evaluate their system on a synthetic stream obtained from randomly partitioning a small set of malware and benign instances, and refer to each partition as one "day," even though that does not represent real time. While this is a standard evaluation technique in the machine learning literature, it is only appropriate when the underlying data does not "drift" between training and testing time. Unfortunately, this requirement is violated for malware detection, where attackers try to evade the antivirus systems [9, 32] or simply upgrade malware in functionality. Indeed, prior work suggests that cross-validation leads to an unrealistically large detection accuracy at system evaluation time that does not translate to real world performance once the system is deployed [14]. This apparent performance degradation occurs because in reality, the system can only rely on past data to predict present and future samples.

Human Integration. Although some prior work has suggested integration of a human expert into machine learning systems for malware detection, our work differs significantly in goal, effectiveness and quality of evaluation. Nissim et al. conduct prior work exploring the integration of a human expert with the goal of maximizing the number of malicious binaries given to the expert [21,22]. In contrast, our goal is to give the binaries to the expert which will improve detection performance. Although Nissim et al. demonstrate that their query technique selects more malicious samples than uncertainty (or margin) sampling, detector true and false positive rates are either worse than [21] or comparable to [22] uncertainty sampling. In contrast, our evaluation demonstrates at a 15% point improvement over uncertainty sampling. Nissim et al. additionally lack timestamps for both binaries and labels in their evaluation and randomize the order of samples, removing any temporal effects from evaluation results.

Dataset Diversity. Most prior work was evaluated on limited number of malicious and benign instances from a single repository source (e.g., VX Heaven) and/or with low or unquantified diversity in terms of malware families [15, 16, 25, 30]. The largest dataset of binary executables can be found in Markel et al. [18] and consists of 123k and 42k malicious and benign instances respectively, with little diversity characterization. In contrast, our dataset has approximately 1.1 million instances collected from Virus Total, a consumer-facing service, with 113k benign samples and 939k malware comprising from 3k to 406k distinct families depending on which AV vendor is used for naming. We characterize in greater detail our dataset in section 5, and in particular show how new families emerge over the 2.5 year time span of the data.

3. Detector Design

In this section we present our detector design, including feature extraction, machine learning and integration of the human labeling expert. Figure 1 presents an overview of our approach. When a binary arrives, the detection pipeline extracts the features, applies the current model to classify the binary as malicious or benign, and the training pipeline stores the binary in a database along with all other binaries seen to-date. During each retraining period, binaries not detected by scanners on VirusTotal are considered for submission to the human labeling expert. Binaries confidently detected by the current model are included in training data with a malicious label, and the remaining purportedly benign binaries are submitted to the human labeling expert as the query budget allows. The remaining un-submitted binaries are included in the training data as benign. At the end of the retraining period, the next model produced in the training pipeline replaces the current model and the process repeats.

We begin by examining the general techniques used for feature vectorization in Section 3.1, and then present the application of feature vectorization techniques to static and dynamic attributes of binaries in Sections 3.2 and 3.3 respectively. Section 3.4 presents our approach to labeling training data, and Section 3.5 describes how we integrate machine learning with an expert.

3.1 Approaches to Feature Vectorization

Many machine learning algorithms work best with numeric

	Name	Description	Example	Vectorization
Static	Binary Metadata	Metadata from MAGIC and EXIFTOOL	`PECompact2 compressed`	Categorical, String
	Digital Signing	Certificate chain identity attributes	`Google Inc`; `Somoto Ltd`	Categorical, String
	Heuristic Tools	TRID; Tools from ClamAV, Symantec	`InstallShield setup`; `DirectShow filter`	Categorical
	Packer Detection	Packer or crypter used on binary	`UPX`; `NSIS`; `Armadillo`	Categorical
	PE Properties	Section hashes, entropies; Resource list, types	`image/x-png`; `hash:eb0c7c289436...`	Categorical, Ordinal, String
	Static Imports	Referenced library names and functions	`msvcrt.dll/ldiv`; `certcli.dll`	Categorical
Dynamic	Dynamic Imports	Dynamically loaded libraries	`shell32.dll`; `dnsapi.dll`	Categorical
	File Operations	Number of operations; File paths accessed	`C:\WINDOWS\system32\mshtml.tlb`	Categorical, Ordinal
	Mutex Operations	Each created or opened mutex	`ShimCacheMutex`; `RasPbFile`	Categorical, String
	Network Operations	IPs accessed; HTTP requests; DNS requests	`66.150.14.*`; `b.liteflames.com`	Categorical, Ordinal
	Processes	Created, injected or terminated process names	`python.exe`; `cmd.exe`	Categorical
	Registry Operations	Registry key set or delete operations	`SET: ...\WindowsUpdate\AU\NoAutoUpdate`	Categorical
	Windows API Calls	*n*-grams of Windows API calls	`DeviceIoControl | IsDebuggerPresent`	Categorical, Sequential

Table 2: Feature vectors reflect static and dynamic attributes of binaries. We discuss vectorization techniques in Section 3.1 and the application of vectorization techniques to attributes of binaries in Sections 3.2 and 3.3.

features, but not all attributes of binaries come in that format. We present four general techniques to convert raw (static and dynamic) attributes of binaries into numerical feature vectors. The methods we can apply to each attribute depends upon its type.

Categorical. We apply categorical vectorization to all attributes. The categorical mapping associates one dimension with each possible attribute value. For example, the `DeviceIoControl` API call may correspond to index i in feature vector x, where $x_i = 1$ if and only if the binary issues the `DeviceIOControl` API call. We extend this mapping as new binaries present new attribute values. Since the absence of an attribute reveals information about a binary, we include a special *null index* to indicate that the value of the attribute is missing. For example, the file may not generate any network traffic, or may not be signed.

Ordinal. Ordinal attributes assume a specific value in an ordered range of possibilities, such as the size of a binary. We describe these attributes using a binning scheme that works as follows: for a given attribute value, we return the index of the bin which the value falls into, and set the corresponding dimension to 1. For attributes that vary widely, we use a non-linear scheme to prevent large values from overwhelming small values during training. For example, the number of written files v is discretized to a value i such that $3^i \leq v < 3^{i+1}$, where the exponential bins to account for the large dynamic range of this quantity.

Free-form String. Many of the important attributes appear as unbounded strings, such as the comments field of the signature check. Representing these attributes as categorical features could allow an attacker to evade detection by altering a single character in the attribute, causing the attribute to map into a different dimension. Therefore, we capture 3-grams of these strings, where each contiguous sequence of 3 characters represents a distinct 3-gram, and consider each of the 3-grams as a separate dimension. However, this approach is still sensitive to variations that alter 3-grams. In addition to taking original 3-grams, we perform string simplification to reduce sensitivity to the 3-gram variations.

We define classes of equivalence between characters and replace each character by its canonical representative. For instance, the string `3PUe5f` could be canonicalized to `0BAa0b`, where upper and lowercase vowels are mapped to '`A`' and '`a`' respectively, upper and lowercase consonants are mapped to '`B`' and '`b`', and numerical characters to '`0`'. Likewise, the string `7SEi2d` would also canonicalize to `0BAa0b`. Occasionally, we sort the characters of the trigrams to further control for variation and better capture the morphology of the string. For instance, these string simplification techniques are used for mapping the portable executable resource names, which sometimes exhibit long random-looking bytes sequences.

Sequential. The value of some attributes is a sequence of tokens where each token assumes a finite range of values. These sequential attributes are strongly related to free-form string attributes, although the individual tokens are not restricted to being individual characters. We use sequential feature extraction to capture API call information since there is a finite set of API calls and the calls occur in a specific order. As with free-form string features, we use an *n*-gram approach where each sequence of *n* adjacent tokens comprises an individual feature.

3.2 Static Attributes of Binaries

In this section, we describe our use of static attributes of binaries. Static attributes are derived from analysis available through VirusTotal. Table 2 provides an overview of static attributes, dynamic attributes and associated vectorization techniques.

Binary Metadata. We use attributes of binaries gathered from two tools designed to extract metadata: MAGIC and EXIFTOOL. The MAGIC tool interprets the binary's magic bytes and returns a literal, such as `MS-DOS executable PE for MS Windows (console) Intel 80386 32-bit`. The EXIFTOOL tools returns a list of key-value pairs. Common keys are `TimeStamp`, `OriginalFilename`, `FileDescription`. We vectorize the MAGIC and EXIFTOOL attributes using a combination of categorical and string vectorization techniques.

Digital Signing. From a check of the signature, we collect the status of the verification, the identities of every entity in the certificate chain, and additional fields that include comments, product name, description, copyright, internal name, and publisher. We vectorize the verification status and specific identities as categorical features, and treat any additional information as string features.

Heuristic Tools. This group of attributes contain decisions of black-box heuristic tools which are inexpensive to run against a given executable. We use three such tools. CLAMAV PUA checks for Potentially Unwanted Applications (PUA) that are not malicious by themselves, but can be used in malicious applications [1]. SYMANTEC SUSPICIOUS INSIGHT returns a binary result based on its reputation among other users [4]. TRID returns a scored list of the most likely file types (e.g., `Win32`

`Executable MS Visual C++`) [6]. We vectorize the results of these tools as categorical features.

Packer Detection. We use three packer/cryptor detectors: COMMAND UNPACKER, F-PROT UNPACKER and PEID [2]. Each tool returns a list of names of packers detected (e.g., UPX, NSIS, Armadillo). We vectorize them as categorical features.

Portable Executable Format. We use several attributes of binaries derived from the portable executable file format [3], including resource languages (e.g., `ENGLISH US`, `NEUTRAL`, `RUSSIAN`), section attributes (e.g. hashes, names, entropies, and lengths) and resource attributes (e.g. hashes, names and types). We vectorize these attributes using a combination of categorical, ordinal and string vectorization techniques.

Static Imports. Static imports are referenced library functions. Functions can be referenced by their actual name (e.g., `msvcrt.dll/ldiv`) or by an ordinal reference (for example, `user32.dll/Ord(490)`). Static import attributes of binaries are vectorized using categorical techniques.

3.3 Dynamic Attributes of Binaries

VirusTotal uses the Cuckoo sandbox to obtain dynamic attributes of binaries the first time each binary is submitted to VirusTotal [5]. We derive all dynamic features from the Cuckoo sandbox execution trace.

Dynamic Imports, Mutexes and Processes. We generate features from each: a) created or opened mutex name; b) created, injected or terminated process name; and c) dynamically loaded library name. These are all vectorized as categorical attributes.

Filesystem Operations. We generate features from the full paths, types and number of operations made on the file system. Operations include reading from, writing to, deleting, copying, and renaming files. We parse each path to extract individual tokens and prefixes, and then vectorize each as a categorical attribute.

Network Operations. We derive features from all IP addresses accessed via both TCP and UDP, DNS queries and responses, and HTTP requests generated by each binary. To increase generality, we also include subnets derived from IP addresses as features. Likewise, we derive more general features from HTTP requests by extracting query parameters and individual tokens from the request path. All IP address, DNS and HTTP request attributes are vectorized using categorical techniques. Additional attributes related to request lengths and total number of requests are vectorized using ordinal techniques.

Registry Operations. We collect all operations that result in an update to a registry key along with the full key path. We vectorize registry operations using categorical techniques.

Windows API Calls Sequence. The raw output of the Cuckoo sandbox contains the sequence of Windows API calls issued by the binary, as well as the arguments, argument values and return value of the system call. We use the sequential technique to vectorize API call sequences with n-grams for n in $\{2,3\}$, over the sequence of windows API call names. We also apply categorical vectorization to the individual system calls in the sequence, including the names (but not the values) of the supplied arguments.

3.4 Labeling the Training Set

To periodically update the detection model, the detector must construct a labeled training set of binaries. To do so, at each period, the detector collects all the binaries seen up to that point and uses the information currently available to assign labels for training. In particular, the detector uses four sources of information: the community consensus on each binary as provided by VirusTotal, the current learned model, the results, if any, of prior expert review and finally, additional fresh expert reviews for a small number of binaries that it selects with a *query strategy*.

To understand how to make use of antivirus vendors labels, we studied our VirusTotal dataset. We found that antivirus vendors tend to prefer false negatives over false positives, possibly due to detection latency and the potential inconvenience of false positives to users. We quantify the detector bias in Section 5, which shows that the number of detections a binary receives is more likely to increase than to decrease over time. This tendency is so strong that we elected to treat a community consensus that the binary is malicious as a sure thing and assign a malicious label to the binary. We call this heuristic the *undetected* filter. Furthermore, since we trust our current detection model to a certain extent, we assign a malicious label to any binary which detection score exceeds a confidence threshold M. We call this heuristic *auto-relabeling*. If both of these heuristics fails to produce a (malicious) label, and if no known expert label is yet available for the binary, we submit the binary to the query strategy.

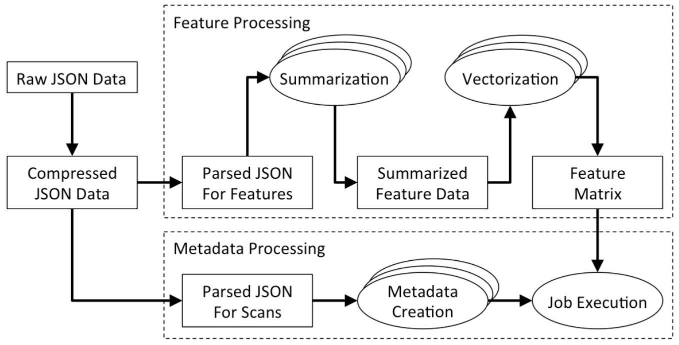
Query Strategy. The query strategy selects which binaries among the previously prefiltered ones to submit to the expert labeler. The stock query strategy of machine learning is the *uncertainty* strategy: submit to the expert the binary about which the current model is most uncertain (i.e., those that lie closest to the boundary).

In general, a query strategy is most effective when the resulting machine learning model incurs a large update as a result of the gained knowledge. For a linear classification model such as logistic regression, the most impactful labels are those of the instances which lie the furthest away from the decision boundary. That is, flipping the labels of the largest scoring binaries (in absolute value) will produce the most different model. Since we know that antivirus providers' false negatives are far more common than false positives, we expect the labels of the highest scoring (most malicious) binaries to have a fair chance of changing from benign to malicious. Hence, it is those a priori benign but high scoring binaries that we submit to the human oracle. We call this strategy *maliciousness*.

More formally, the query strategy has a submission budget B, where B is determined as a fixed percentage of the total number of new training binaries during the retraining period. The detector then submits the B remaining binaries with the greatest maliciousness scores to the expert. The binaries in excess of B, i.e. those that are not submitted to the human labeler are conservatively labeled as benign.

3.5 Model Training

We base our detection on the logistic regression and begin by examining its mechanics and advantages. As a linear classification technique, logistic regression assigns a weight to each feature and issues predictions as a linear function of the feature vector. Formally, if $\mathbf{x} \in \mathbb{R}^d$ is the feature vector representing an instance and $\mathbf{w} \in \mathbb{R}^d$ are the model weights, then the algorithm



(a)

Item	Format	Size
Raw JSON Data	ASCII Text	778.2GB
Compressed JSON Data	gzip	32.3GB
Parsed JSON For Features	Parquet	25.9GB
Summarized Feature Data	Parquet	3.9GB
Feature Matrix	Numpy & Scipy Compressed Arrays	1.7GB
Parsed JSON For Scans	Parquet	0.9GB

(b)

Figure 2: Implementation Overview. Figure 2a presents a block diagram of our implementation, where rectangles correspond to distributed storage and stacked ovals correspond to parallel computation. Figure 2b presents the size of each item in distributed storage. The size of items decreases as a result of compression and processing. Job execution is inherently serial due to dependence on the previous model for expert queries.

scores the instance as:

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{d} \mathbf{x}_i \mathbf{w}_i$$

where without loss of generality, we have omitted the constant additive bias term. Linear classification scales well in prediction as the size of the model is a function of the dimensionality of the data, and not with the size of the training data as happens with a kernelized SVM. We adjust the threshold at which a binary is considered malicious by scoring each binary as a real valued quantity $f_{\mathbf{w}}(\mathbf{x})$. This enables us to create a tradeoff between true and false positive rates. Considering malicious as the positive class, higher thresholds will result in lower false positive rates while missing more malware, and lower thresholds will result in higher false positive rates while detecting more malware. Additionally, the clear relationship between binary features and the classification outcome allows supervisors to clearly understand what the detector is doing and why. Also, logistic regression scales well in training with many available implementations capable of accommodating high dimensional feature spaces and large amounts of training data.

In logistic regression, the model is defined by a weight vector $\mathbf{w}$ learned from a given labeled training set $\{(\mathbf{x}^1, y^1), \ldots, (\mathbf{x}^n, y^n)\}$ where $y^i \in \{-1, +1\}$ represents the label. Logistic regression finds the model $\mathbf{w}$ which minimizes the following risk function:

$$C_- * \sum_{i:y^i=-1} \ell(-f_{\mathbf{w}}(\mathbf{x}^i)) \quad + \quad C_+ * \sum_{i:y^i=1} \ell(f_{\mathbf{w}}(\mathbf{x}^i)) \quad + \quad \frac{1}{2}\|\mathbf{w}\|^2$$

where $C_- > 0$ and $C_+ > 0$ are distinct hyper-parameters controlling for both regularization and class importance weighting and $\ell(x) = \log(1 + \exp(-x))$ is the logistic loss function. The first and second terms correspond to the misclassification losses for negative and positive instances, respectively. The last term is a regularization term that discourages models with many large non-zero weights. We use L_2-regularization, resulting in a dense weight vector $\mathbf{w}$. Finally, while online learning techniques are in theory well suited for frequent model retraining, we fit the weight vector using the exact batch-optimization tool LIBLINEAR [11]: batch training times remain reasonable even for our million instances dataset, and we observe inferior detection accuracies for models learned online.

Since the expert will only review a fraction of the binaries labeled benign, many false negatives will persist. To counteract the false negatives, any binary labeled benign by the expert is given a higher weight W during training.

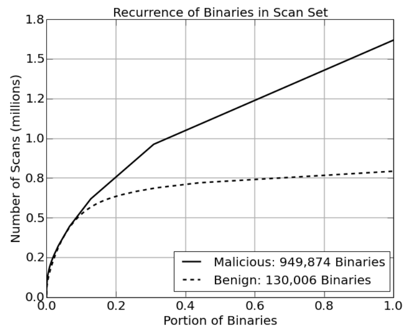
The detector design we present has expert labeler submission budget B, auto-relabeling confidence threshold M, C_-, C_+, W, and the retraining period as parameters. Section 7 presents the effects of varying the submission budget B and retraining period, with increased submissions and more frequent retraining resulting in higher performance. The others are fixed to values yielding good performance on a set of binaries excluded from our evaluation: $M = 1.25$, $C_- = 0.16$, $C_+ = .0048$ and $W = 10$.

4. Detector Implementation

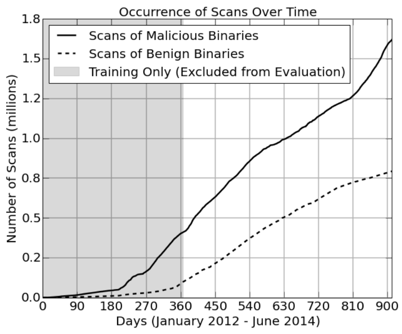
In this section we present our detector implementation. Since anti-virus vendors can receive in excess of 300,000 binaries daily [19], we design our detector with a focus on scalability. Our detector can conduct feature vectorization to prepare binaries for classification for all 1 million binaries included in our evaluation in less than 6 hours. Thus, as VirusTotal receives and processes as many as 1 million new files daily, the end-to-end combination of our detector with VirusTotal analysis infrastructure could perform detection on a realistic workload [29].

We implement our detector on a VMware ESX cluster running Apache Spark [31]. Since Apache Spark holds data in memory, computational tasks making repeated use of the same data elements operate faster, enabling greater scalability. Within the cluster we designate 10 nodes as worker nodes, each allocated 3 cores, 48GB of memory, 16GB disk swap space and 32GB Spark spill space. We designate a single master node with 8 cores, 120GB of memory, 16GB disk swap space and 32GB Spark spill space. We implemented our design in approximately 4,600 lines of Python.

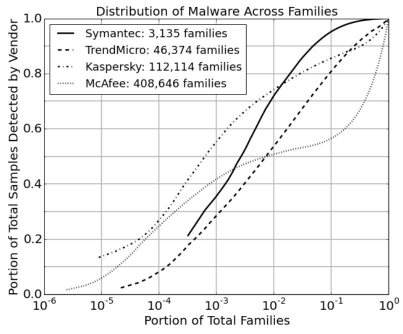
Figure 2a presents an overview of our design, divided into feature processing and metadata processing. Feature processing is responsible for the transformation of raw data into a feature matrix, with one row corresponding to each binary. Once a feature matrix is produced, metadata processing produces tuples specifying all rows and labels necessary for both training and evaluation during each retraining period. Once the features



(a)



(b)



(c)

Figure 3: Data Overview. Figures 3a and 3b demonstrate that scans are well distributed across binaries and our evaluation period, respectively. Note that relative scarcity of scans in the first 200 days reflects availability of necessary attributes in VirusTotal data, not underlying phenomenon in submission behavior. Figure 3c presents the distribution of malware across families. Each vendor regards 50% or more of malware as belonging to 10% of the families identified by the vendor, with the number of families varying from 3,129 (Symantec) to 406,250 (McAfee).

and metadata are produced for each time period, all training, prediction and expert queries occur in job execution.

Since feature vectorization is the most computationally intensive aspect of our detector, we utilize a combination of compression, caching and parallelism to reduce computational burden and maintain scalability. Figure 2b presents the reduction in data size afforded by the feature vectorization sequence. Raw data is first compressed using `gzip` to store the original records unaltered in HDFS, and compressed a second time as the data is transferred into the Parquet format. Parquet enables increased compression over `gzip` by maintaining a schema for the data, effectively removing the need to store the keys associated with each element of a dictionary.

After raw data has been transferred to Parquet, feature vectorization proceeds to an intermediate caching step known as *summarization* allowing result reuse across experiments. During the summarization process we extract the necessary data for subsequent vectorization of the feature, often reducing the total size of the data. For example, a feature based on the total number of URL requests need only store the total number of requests made, not the actual requests themselves.

After summarization, a vectorization process transforms the summarized data into feature vectors. If the vectorization process were constructed such that each binary can be vectorized independent of other binaries, then summarization data can be discarded after vectorization. However, vectorization depends on all binaries, so the summarized data must be re-visited as new binaries are added to the training data.

5. Data and Label Overview

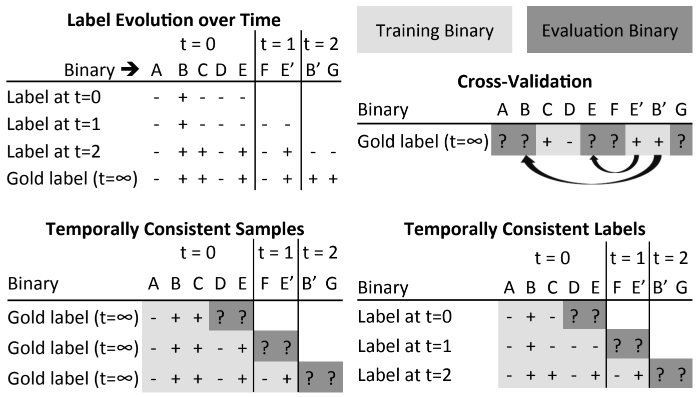
In this section we examine the dataset we use for our evaluation, consisting of over 1 million distinct binaries submitted to VirusTotal between January 2012 and June 2014. We begin with an overview examining the distribution of the data over time and diversity across malware families. Then, we discuss changes in detection results over time and our approach to obtaining accurate labels.

In addition to the static and dynamic analysis and virus detection conducted when VirusTotal first receives a file, VirusTotal rescans the file with up-to-date virus detectors on any subsequent submission of the file. Figure 3a depicts the impact of resubmissions on the dataset. We include these rescan events in our analysis since rescans offer updated, timestamped labels for the file, providing for more timely labeling during evaluation. Additionally, inclusion of re-submissions ensures that the distribution of our evaluation data mirrors the distribution of actual data submitted to VirusTotal by incorporating the prevalence of each individual file, effectively balancing any effects of polymorphism in the dataset.

Although our evaluation includes data from January 2012 to June 2014, we reserve the first year of data for training purposes only and use data from January 2013 to June 2014 to assess the performance of our detector. Figure 3b presents the occurrence of scans over time, indicating that scans consistently occur throughout the period assessing our detector performance. Notice that scans do not occur evenly during the training period, with the first approximately 200 days containing fewer scans. The difference in available data occurs because fewer binaries have dynamic attributes available; the difference does not reflect an underlying phenomenon in submissions.

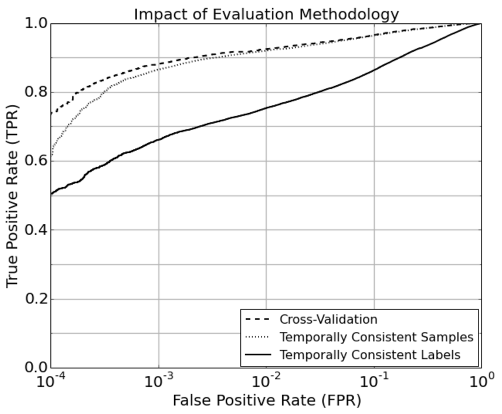
We also demonstrate that binaries reflect a broad range of malware families. Figure 3c presents the distribution of binaries across families from four leading vendors. Since our goal is to reflect the true diversity of the binaries, rather than the initial perceptions of scanners, we associate binaries with the family given in the final scan of the binary to obtain the most accurate family label. Since each vendor uses a unique naming scheme and identifies a different set of binaries as malware we plot the portion of total binaries detected by each vendor compared to the portion of total families identified by the vendor, with families ordered from largest to smallest. Each vendor agrees that the distribution across families is non-uniform, with more than 50% of binaries belonging to the most common 10% of families issued by each vendor. As the number of families identified by vendors ranges from 3,129 to 406,250, the majority of malicious binaries are drawn from hundreds if not thousands of families.

We now discuss our approach to labeling binaries. Although



Label Evolution over Time
t = 0
t = 1
t = 2
Binary ➔ A B C D E F E′ B′ G
Label at t=0 - + - - -
Label at t=1 - + - - - - -
Label at t=2 - + + - + - + - -
Gold label (t=∞) - + + - + - + + +
Training Binary
Evaluation Binary
Cross-Validation
Binary A B C D E F E′ B′ G
Gold label (t=∞) ? ? + - ? ? + + ?
Temporally Consistent Samples
Binary A B C D E F E′ B′ G
Gold label (t=∞) - + + ? ?
Gold label (t=∞) - + + - + ? ?
Gold label (t=∞) - + + - + - + ? ?
Temporally Consistent Labels
Binary A B C D E F E′ B′ G
Label at t=0 - + - ? ?
Label at t=1 - + - - - ? ?
Label at t=2 - + + - + - + ? ?

(a)



(b)

Figure 4: Temporally consistent labels are critical for accurate evaluation results. Figure 4a illustrates three types of evaluations. The upper left shows the evolution of the data set including two instances (B and E) receiving a change in label. The three remaining subfigures each shows how a type of evaluation would handle the shown data set. Rows correspond to successive retraining periods with specified training and evaluation data, binaries appear chronologically from left to right, and + and − denote malicious and benign labels, respectively. Notice that cross-validation disregards time for binaries and labels, while temporally consistent samples orders binaries chronologically but uses gold labels regardless of time. Figure 4b presents the effects of evaluation methodology on accuracy results.

we observe scan results from 80 different vendor's detectors,[1] some of these detectors are only sporadically present in the data. Since we label binaries using detection results from multiple vendors, we restrict our work to the 32 vendors present in at least 97% of scan results to increase consistency in the set of vendors applied to each binary.[2]

Although the contents of a binary may not change over time, the communal understanding of whether the binary is malicious can change. Due to the potential inconvenience for users of false positive detection results, label changes are overwhelmingly from benign to malicious. Examination of binaries with multiple scans in our dataset reveals that while 29.6% of binaries increase in number of detections by at least 5 vendors from their first to last scan, only 0.25% of binaries decrease by 5 or more detections. Given vendors' demonstrated aversion to false positives, we set a detection threshold of four vendor detections to label a binary as malicious, and rescan any binary which received fewer than 10 detections at the most recent scan. We conduct rescans in February and March 2015, 8 months after the end of our data collection period, to allow time for vendor signature updates. We avoid rescanning binaries with 10 or more detections since decreases large enough to cross the four vendor detection threshold are unlikely. After rescanning, we assign a *gold label* to each binary in our dataset representing the best available understanding of whether the binary is malicious.

[1]Since vendors are able to customize their products for VirusTotal results may differ from retail or enterprise products.

[2]In particular, we include the following vendors: AVG, Antiy-AVL, Avast, BitDefender, CAT-QuickHeal, ClamAV, Comodo, ESET-NOD32, Emsisoft, F-Prot, Fortinet, GData, Ikarus, Jiangmin, K7AntiVirus, Kaspersky, McAfee, McAfee-GW-Edition, Microsoft, Norman, Panda, SUPERAntiSpyware, Sophos, Symantec, TheHacker, TotalDefense, TrendMicro, TrendMicro-HouseCall, VBA32, VIPRE, ViRobot and nProtect.

6. Evaluation Method

We build evaluation method around faithfully modeling the actual setting of malware detection at a reasonable cost. To ensure that our evaluation is faithful to the actual setting of malware detection, we maintain the temporal consistency of both samples and labels. To keep costs reasonable, we simulate rather than employ labeling experts.

In practice, knowledge of both binaries and labels changes over time as new binaries appear and malware detectors respond appropriately with updated labels. Evaluations that fail to recognize the emergence of binaries and knowledge over time effectively utilize knowledge from the future, inflating the measured accuracy of the approach. For example, consider malware that evades detection but can be easily detected once the first instance is identified. Accuracy inflation occurs because inserting correctly labeled binaries into training data circumvents the difficult task of identify the first instance of the malware.

Figure 4a presents three approaches to evaluation of malware detectors which recognize the emergence of binaries and labels to varying degrees. Cross-validation is a common approach for machine learning evaluations in situations where binaries are independent and identically distributed (i.i.d.). In the malware detection context the i.i.d. assumption does not hold since malware changes over time to evade detection. Cross-validation evaluations completely disregard time, dividing binaries randomly and applying evaluation quality labels to all binaries. Evaluations maintaining temporally consistent samples recognize the ordering of binaries in time but not the emergence of labels over time, instead applying gold labels from future scan results to all binaries. Use of gold quality labels during training effectively assumes that accurate detection occurs instantly. Evaluations maintaining temporally consistent labels fully respect the progression of knowledge, ordering binaries in time and restricting the training process to binaries and labels available at the time of training. To support the temporal consistency of labels, our dataset stores the label available for each binary at each time it is submitted to VirusTotal.

We also provide a method of simulating a labeling expert. Since the evaluation data spans 2.5 years, involvement of actual humans to reproduce the equivalent effort on a much shorter timescale is not economically feasible. Rather, we model the involvement of a human with an oracle that can see the gold label of a binary from the future. For specific experiments in our evaluation, we consider an imperfect oracle functioning with a specified true positive rate and false positive rate, where the likelihood of the oracle supplying the correct label depends on the gold label of the sample.

In the next section, we present not just the results of evaluating our detector with temporally consistent labels, but also with cross-validation, temporally consistent samples, and various oracle error rates. These examinations illustrate the important of our methodological considerations.

7. Results

In this section we evaluate our malware detector. We begin by examining the impact of temporal consistency of labels on accuracy measurements, showing its importance to evaluating detectors realistically. Next, we examine the performance improvement from the oracle and impact of variations in oracle query budget and accuracy, query strategy, timeliness of training data labels, and length of retraining period. Without the use of the oracle, our detection approach performs comparably to detectors on VirusTotal. With modest support from the oracle, we achieve performance improvements over the vendor labels supplied on VirusTotal. We conclude with an examination of which features of binaries were the most helpful for detection.

Impact of Temporal Consistency of Labels. We begin by comparing several approaches to evaluating malware detectors. Since the oracle effectively reduces the impact of temporally consistent labels by revealing future labels, we conduct these evaluations without any oracle queries. Figure 4b presents the results of our analysis. Notice that cross-validation and temporally consistent samples perform similarly, inflating accuracy results 20 and 19 percentage points respectively over temporally consistent evaluation at a 0.5% false positive rate.

Impact of Labeling Oracle. Given the temporal factors we consider, the vendor detection results on VirusTotal provide the best performance comparison for our work. Based on the false positive rates of vendors, we tune our detector to maximize detection for false positive rates greater than 0.1% and less than 1%. Figure 5 compares our performance to vendor detectors provided on VirusTotal. Without involvement from the labeling oracle our detector achieves 72% detection at a 0.5% false positive rate, performing comparably to the best vendor detectors. With support from the labeling oracle, we increase detection to 89% at a 0.5% false positive rate using 80 queries daily on average. Since we train a separate model during each retraining period, the performance curve results from varying the same detection threshold across the results of each individual model.

VirusTotal invokes vendor detectors from the command line rather than in an execution environment, allowing detectors to arbitrarily examine the file but preventing observation of dynamic behavior. Since our analysis includes dynamic attributes, we also observe our performance when restricted to static at-

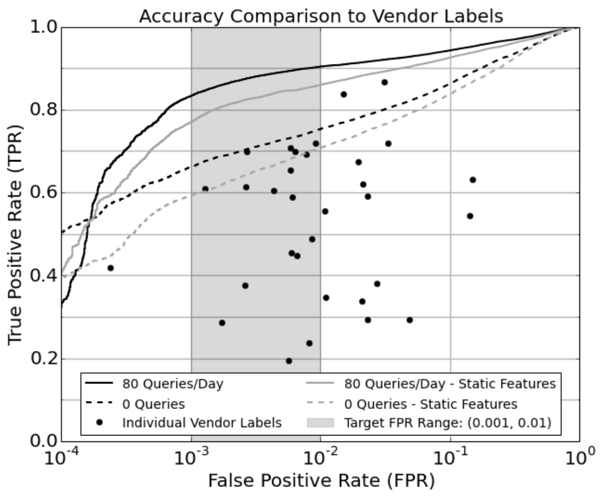


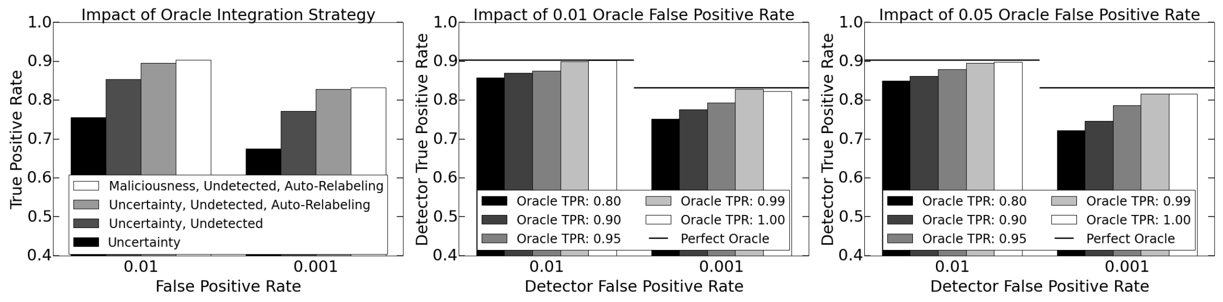
Figure 5: Without the labeling oracle, detector performance competes with VirusTotal detectors. With the labeling oracle, detection improves beyond vendors on VirusTotal. We tune the oracle integration to maximize detection in the (0.1%, 1%) false positive region, consequently decreasing detection at lower false positive rates.

tributes provided by VirusTotal. Note that this restriction places our detector at a strict disadvantage to vendors, who may access the binary itself and apply signatures derived from dynamic analysis. Figure 5 demonstrates that our performance decreases when restricted to static features but, with support from the labeling oracle, continues to surpass vendors, achieving 84% detection at a 0.5% false positive rate.

Performance comparison must also consider the process of deriving gold labels, which introduces a circularity that artificially inflates vendor performance. Consider the case of a false positive: once a vendor has marked a binary as positive, the binary is more likely to receive a positive gold label, effectively decreasing the false positive rate of the vendor. An alternate approach would be to withhold a vendor's labels when evaluating that vendor, effectively creating a separate ground truth for each vendor. Although this approach more closely mirrors the evaluation of our own detector (which does not contribute to gold labels), in the interest of consistency we elect to use the same ground truth throughout the entire evaluation since efforts to correct any labeling bias only increase our performance differential.

In addition to offering higher levels of detection across all data than vendor labels, our approach also experiences greater success detecting novel malware that is missed by detectors on VirusTotal. Of the 1.1 million samples included in our analysis, there are 6,873 samples which have a malicious gold label but are undetected by all vendors the first time the sample appears. Using 80 oracle queries daily, our approach is able to detect 44% and 32% of these novel samples at 1% and .1% false positive rates, respectively. The ability of our approach to detect novel malware illustrates the value of machine learning for detecting successively evolving generations of malware.

To provide a corresponding analysis of false positives, we measure our performance on the 61,213 samples which have a benign gold label and are not detected as malware by any ven-



(a) (b) (c)

Figure 6: Figure 6a presents the impact of each component in our customized oracle query strategy. We improve detection over the *uncertainty* sampling approach from prior work. Figures 6b and 6c present the performance of our detector for imperfect oracles with the specified true and false positive rates. For example, given an oracle with a 5% false positive rate and 80% true positive rate, our detector's true positive rate only decreases by 5% at a 1% false positive rate. Expert budget is set to $B = 80$ in all figures.

dor the first time the sample appears. Of these 61,213 benign samples, our detector labels 2.0% and 0.2% as malicious when operating at 1% and .1% false positive rates over all data, respectively. The increased false positive rate on initial scans of benign samples is expected since the sample has not yet been included as training data.

Oracle Integration Strategies. Our oracle integration strategy represents numerous advances over generic ones for problems outside of computer security. Figure 6a presents the impact of each of the three improvements we introduce and discussed in Section 3.4. For a fixed labeling budget $B = 80$, the stock uncertainty sampling results in 17 percentage points lower detection rate than the combination of our techniques at 0.1% false positive rate.

Oracle Accuracy. Our system also demonstrates strong results in the presence of an imperfect oracle. Although our work models the presence of a human labeler as an oracle, actual human labelers may make mistakes in identifying malware from time to time. Malware creators may explicitly design malware to appear benign, but benign software is less likely to appear malicious. Accordingly, we model the false positive and true positive rates of oracles separately, reflecting an oracle which is more likely to mistake malware for benign software than benign software for malware. Figures 6b and 6c present detection rates for oracles with 1% and 5% false positive rates respectively and a range of true positive rates. For example, given an oracle with a 5% false positive rate and 80% true positive rate, our detector's true positive rate only decreases by 5% at a 1% false positive rate.

Resource Parameterization. Beyond classifier parameters (discussed in Section 3.5), operator resources determine several additional parameters including oracle query budget, frequency of retraining and frequency of rescanning training data. We explore each of these parameters individually below.

As the allowed budget for queries to the oracle increases, the detector accuracy increases since more accurate labels are available. Figure 7a presents the increase in accuracy from increased oracle queries, with the benefit of 80 queries per day on average approaching the benefit of training on gold labels for all binaries. The benefit of oracle queries is non-linear, with the initial queries providing the greatest benefit, allowing operators to experience disproportionate benefit from a limited oracle budget.

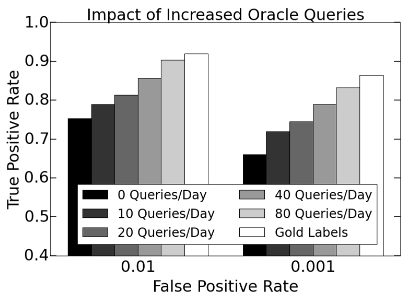
Although our evaluation is large relative to academic work, an actual deployment would offer an even larger pool of possible training data. Since the utility of oracle queries will vary with the size of the training data, increasing the amount of training data may increase oracle queries required to reach full benefit. Fortunately, the training process may elect to use only a subset of the available training data. We demonstrate that 1 million binaries selected randomly from VirusTotal submissions is sufficient training data to outperform vendor labels for our evaluation data.

Separate from querying the oracle, rescanning binaries with updated detectors would also improve label quality. To maintain temporal consistency our evaluation uses the most recent detection results occurring before model training. Unfortunately there is no guarantee that binaries have been submitted regularly, so detection results may be outdated. While our approach maintains a conservative estimation of accuracy, timely labels for training data may improve performance.

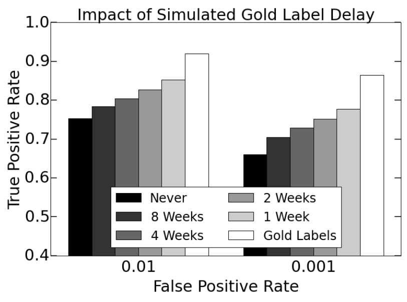
To simulate the benefit of regular rescanning, we reveal the gold label for training purposes once a specified amount of time has elapsed since the binary's first submission. Figure 7b presents the results of our analysis, which we conduct while retraining every 7 days with 0 oracle queries to observe the effects of timely training labels in isolation. Notice that even when gold binaries are released after 1 week, detection remains approximately 9 percentage points lower than when training on all gold labels. This phenomenon illustrates the necessity of the labeling oracle as well as the importance of maintaining timely labels for training data.

Lastly, we examine variations in the length of the re-training period governing how often models are updated. We conduct these experiments with 80 oracle queries on average per day. Figure 7c presents the effect of variations in the retraining period. Notice that the benefit of frequent retraining begins to diminish around 2 weeks. A comparison with Figure 7b reinforces the importance of oracle queries for improving training data labels: the benefit of oracle queries exceeds the benefit of including recent binaries in training data with the wrong label.

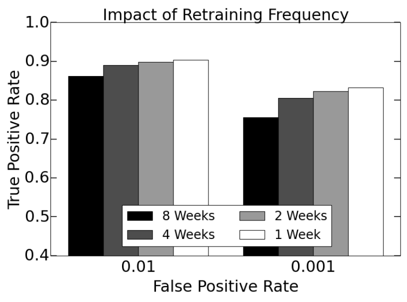
Detection Mechanics. Having analyzed detection accuracy and evaluation methodology, we now examine the features that our detector uses for classification. In the interest of understanding the dataset as a whole, we train a model over all data



(a)



(b)



(c)

Figure 7: Figure 7a presents performance for different oracle query budgets, with significant return on minimal efforts and diminishing returns occurring around 80 queries/day. Figure 7b demonstrates that regular rescans of training data may boost accuracy as training labels are more accurate. Figure 7c demonstrates that retraining more quickly improves detector performance.

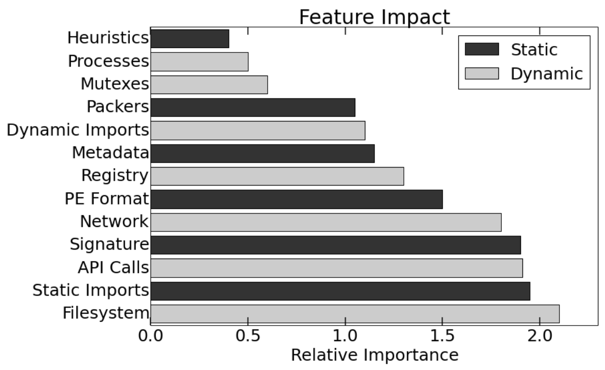


Figure 8: Feature categories ranked by importance.

from all dates. Because we learn a linear model **w** and each feature is 0 or 1, we can readily present the model as a list of these features and their associated weights. However, inspecting the weights vector alone is not enough to understand features importance: a feature can be associated with a large weight but be essentially constant across the dataset. Intuitively, such features have low discrimination power. Furthermore, we are interested in grouping low-level features together into high level concepts that reflect the original measurements.

Thus, we use the following ranking method for sets of features. For a given weight vector **w** and a given set of instances $\{\mathbf{x}^i\}_i$, we can compute the importance of a group $S \subset \{1,\dots,d\}$ of features by quantifying the amount of score variation I_S they induce. Specifically, we use the following formula for ranking:

$$I_S = \sqrt{\operatorname*{Var}_{\mathbf{x}}\left[\sum_{k\in S}\mathbf{x}_k\mathbf{w}_k\right]}$$

Using this ranking method, Figure 8 shows the global ranking of the features when grouped by their original measurements. The most important measurements are thus the file system operations, static imports, API call sequence and digital signature, while the least useful measurement is the heuristic tools. Table 3 shows the most positively and negatively weighted dimensions of **w**.

8. Conclusion

w	dimension semantic
3.9	one section has specific hash `78752d...`
3.4	one resource has specific hash `049eb4...`
2.7	signed by `Conduit Ltd.`
2.7	opens a mutex with pattern `Babababbab`
2.5	one resource has specific hash `932c6e...`
2.5	makes http request with url `/img/beginogo...`
2.4	dynamically loads `lz32` library
2.4	empty TRID measurement
2.3	call sequence contains subsequence `LoadLibraryA, OpenMutexW, SetWindowsHookExA`
2.2	signed by `PC Utilities Software Limited`
...	...
-1.8	one section has specific hash `4d3932...`
-1.8	one resource has specific hash `b7f5c1...`
-1.8	one section has specific hash `2a70e9...`
-1.9	DNS query for `VBOXSVR.ovh.net`
-1.9	one resource has specific hash `2b9c54...`
-1.9	DNS query for `vboxsvr.ovh.net`
-1.9	signed by `Google Inc`
-2.1	one resource has specific hash `a8d9db...`
-2.2	one resource has specific hash `69897c...`
-3.9	no packer found by COMMANDUNPACKER

Table 3: Top 20 model dimensions with largest magnitudes. Negative weights are associated to benign instances, positive weights to malicious. The pattern of the mutex name is a result of the string simplification rules presented in Subsection 3.1.

In this paper, we explore the power of putting humans in the loop by integrating a simulated human labeling expert into a scalable malware detection system. We show it capable of handling over 1 million samples using a small cluster in hours while substantially outperforming commercial anti-virus providers both in terms of malware detection and false positive rates (as measured using VirusTotal). We explain why machine learning systems perform very well in research settings and yet fail to perform reasonably in production settings by demonstrating the critical temporal factors of labeling, training, and evaluation that affect evaluation accuracy in real-world settings. We also provide guidelines for the proper temporal use of labeling during training and evaluation and show that the use of statistical machine learning in malware detection is not just promising, but can produce high quality, competitive results in a setting that more closely reflects realistic conditions.

In future, we plan to estimate the cost of an oracle in terms of time and expertise. Discussions with labeling experts suggest that binaries can typically be labeled as malicious or benign by experts in approximately one hour. Our system can significantly expedite the decision process by strategically choosing tricky binaries and ranking their attributes. A user study with malware analysts can help measure the effectiveness of our approach in reducing human effort. We also plan to investigate adversarial attacks against our system. An adversary aware of our query strategy can manipulate binaries either to bypass or to overwhelm the oracle.

To encourage further exploration of the integration of humans into systems using machine learning for security, we have released our machine learning pipeline as open-source and published a dataset containing the hashes we used for evaluation and training along with 3% of all data.

9. References

[1] ClamAV PUA. `http://www.clamav.net/doc/pua.html`. Accessed: 2014-11-14.

[2] PEiD. `http://woodmann.com/BobSoft/Pages/Programs/PEiD`. Accessed: 2014-11-14.

[3] Portable Executable Format Specification. `http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx`. Accessed: 2014-11-14.

[4] Symantec Suspicious Insight. `http://www.symantec.com/security_response/writeup.jsp?docid=2010-021223-0550-99`. Accessed: 2014-11-14.

[5] The Cuckoo Sandbox. `http://www.cuckoosandbox.org`. Accessed: 2014-11-14.

[6] TRID. `http://mark0.net/soft-trid-e.html`. Accessed: 2014-11-14.

[7] ALAZAB, M., VENKATRAMAN, S., WATTERS, P., AND ALAZAB, M. Zero-day malware detection based on supervised learning algorithms of API call signatures. In *Ninth Australasian Data Mining Conference - Volume 121* (2011).

[8] CANALI, D., LANZI, A., BALZAROTTI, D., KRUEGEL, C., CHRISTODORESCU, M., AND KIRDA, E. A quantitative study of accuracy in system call-based malware detection. In *2012 Intl. Symp. on Software Testing and Analysis* (2012), pp. 122–132.

[9] CHRISTODORESCU, M., AND JHA, S. Testing malware detectors. In *ACM SIGSOFT Software Engineering Notes* (2004), ACM.

[10] DAMBALLA. State of Infections Report: Q4 2014. Tech. rep., Damballa, 2015.

[11] FAN, R., CHANG, K., HSIEH, C., WANG, X., AND LIN. LIBLINEAR : A Library for Large Linear Classification. *The J. Machine Learning Research*, 2008 (2008).

[12] GAVRILUT, D., CIMPOESU, M., ANTON, D., AND CIORTUZ, L. Malware detection using perceptrons and support vector machines. In *Computation World* (2009), pp. 283–288.

[13] HENCHIRI, O., AND JAPKOWICZ, N. A feature selection and evaluation scheme for computer virus detection. In *Sixth Intl. Conf. on Data Mining* (2006), pp. 891–895.

[14] KANTCHELIAN, A., AFROZ, S., HUANG, L., ISLAM, A. C., MILLER, B., TSCHANTZ, M. C., GREENSTADT, R., JOSEPH, A. D., AND TYGAR, J. D. Approaches to adversarial drift. In *ACM Wksp. on Artificial Intelligence and Security* (2013).

[15] KIRDA, E., KRUEGEL, C., BANKS, G., VIGNA, G., AND KEMMERER, R. Behavior-based spyware detection. In *Usenix Security* (2006).

[16] KOLTER, J. Z., AND MALOOF, M. A. Learning to detect malicious executables in the wild. In *Intl. Conf. on Knowledge Discovery and Data Mining* (2004).

[17] KOLTER, J. Z., AND MALOOF, M. A. Learning to detect and classify malicious executables in the wild. *J. Machine Learning Research 7* (2006).

[18] MARKEL, Z., AND BILZOR, M. Building a machine learning classifier for malware detection. In *Anti-malware Testing Research (WATeR), 2014 Second Wksp. on* (2014), pp. 1–4.

[19] MCAFEE LABS. McAfee Labs Threats Report, 2014.

[20] MOSKOVITCH, R., FEHER, C., AND ELOVICI, Y. A chronological evaluation of unknown malcode detection. In *Intelligence and Security Informatics*, vol. 5477 of *LICS*. 2009, pp. 112–117.

[21] NISSIM, N., COHEN, A., MOSKOVITCH, R., SHABTAI, A., EDRY, M., BAR-AD, O., AND ELOVICI, Y. Alpd: Active learning framework for enhancing the detection of malicious pdf files. In *IEEE Joint Intelligence and Security Informatics Conf.* (2014), pp. 91–98.

[22] NISSIM, N., MOSKOVITCH, R., ROKACH, L., AND ELOVICI, Y. Novel active learning methods for enhanced PC malware detection in windows OS. *Expert Systems with Applications 41*, 13 (2014), 5843 – 5857.

[23] PERDISCI, R., LANZI, A., AND LEE, W. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual* (2008), pp. 301–310.

[24] RAJAB, M. A., BALLARD, L., LUTZ, N., MAVROMMATIS, P., AND PROVOS, N. CAMP: Content-Agnostic Malware Protection. In *Network and Distributed System Security Symp.* (2013).

[25] SCHULTZ, M. G., ESKIN, E., ZADOK, E., AND STOLFO, S. J. Data mining methods for detection of new malicious executables. In *IEEE Symp. on Security and Privacy* (2001).

[26] SCHWENK, G., BIKADOROV, A., KRUEGER, T., AND RIECK, K. Autonomous learning for detection of javascript attacks: Vision or reality? In *ACM Wksp. on Artificial Intelligence and Security (AISec)* (2012).

[27] SHABTAI, A., MOSKOVITCH, R., FEHER, C., DOLEV, S., AND ELOVICI, Y. Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics 1*, 1 (2012).

[28] ŠRNDIC, N., AND LASKOV, P. Detection of malicious PDF files based on hierarchical document structure. In *Network & Distributed System Security Symp.* (2013).

[29] VIRUSTOTAL. `https://www.virustotal.com/en/statistics/`. Retrieved on July 30, 2014.

[30] YE, Y., CHEN, L., WANG, D., LI, T., JIANG, Q., AND ZHAO, M. Sbmds: An interpretable string based malware detection system using svm ensemble with bagging, 2009.

[31] ZAHARIA, M., CHOWDHURY, M., DAS, T., DAVE, A., MA, J., MCCAULEY, M., FRANKLIN, M. J., SHENKER, S., AND STOICA, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *USENIX Conf. on Networked Systems Design and Implementation* (2012).

[32] ZHOU, Y., AND JIANG, X. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symp. on* (2012), IEEE.