Scalable Platform for Malicious Content Detection Integrating Machine Learning and Manual Review

Bradley Miller

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-194
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-194.html

August 14, 2015

Acknowledgement

I would like to acknowledge the following funding sources: DETER (DHS and NSF Funded Science & Technology Center), State Department Bureau of Democracy, Human Rights and Labor, Freedom2Connect Foundation with funding from the Open Technology Fund administered by Radio Free Asia, Intel Science and Technology Center SCRUB (Secure Computing Research for Users Benefit). The opinions in this thesis are those of its author and do not necessarily reflect the opinion of the US Government, its agencies or any other funding sponsor.

Scalable Platform for Malicious Content Detection Integrating Machine Learning and Manual Review

by

Bradley Austin Miller

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Anthony D. Joseph, Co-chair
Professor J.D. Tygar, Co-chair
Assistant Professor Adityanand Guntuboyina

Summer 2015

Scalable Platform for Malicious Content Detection Integrating Machine Learning and Manual Review

Abstract

Scalable Platform for Malicious Content Detection
Integrating Machine Learning and Manual Review

by

Bradley Austin Miller

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Anthony D. Joseph, Co-chair

Professor J.D. Tygar, Co-chair

This thesis examines the design, implementation and performance of a scalable analysis platform for the detection of malicious content. To reflect the deployment of actual production systems, we design our platform to explicitly model the passage of time and the involvement of human supervisors in the analysis process. This thesis shows how our platform can operate efficiently at a large scale. The thesis presents and evaluates our platform in the context of a case study focused on malware detection.

To model the passage of time while still allowing for batch training methods our platform discretizes time into a series of retraining periods, allowing updated samples and labels to emerge during each period. During each retraining period, our platform combines the presently deployed model with externally available information about newly emerged samples to select samples for submission to a human labeling oracle. To support a large volume of data over successive timeframes, our platform uses advanced techniques to manage the size of data including compression and selective data retention. These operations support efficient feature extraction.

Our platform is implemented in Python, allowing use of both the Python scientific stack (Numpy, Scipy, Scikit-Learn) and IPython for interactive, distributed computation. In the interest of scalability our system uses HDFS and Apache Spark to manage distributed data and computation. This thesis discusses our implementation as well as the hardware and software configuration supporting our system.

This thesis presents an evaluation of our work using a malware dataset containing over 1 million samples collected over a period of 2.5 years. It begins by characterizing our dataset,

including an examination of label shift over time motivating our work. It presents evidence demonstrating that by submitting a small fraction of samples for human review we are able to appreciably increase detection outcomes.

We have released our code along with 3% of our case study data, allowing replication of our results on a single node. Note that detection performance will vary due to the decrease in available training data.[1]

[1]As of the filling of this thesis, our code and data are available online at `http://secml.cs.berkeley.edu/detection_platform/`

To My Father

Contents

List of Figures

List of Tables

Acknowledgments

This thesis would not have been possible without the continuous support of Professors J.D. Tygar and Anthony D. Joseph who co-advised this work. I would also like to thank my collaborators Sadia Afroz, Rekha Bachwani, Riyaz Faizulabhoy, Michael C. Tschantz, Vaishaal Shankar, George Yiu and Tony Wu who also contributed to this work. I would particularly like to thank Alex Kantchelian and Ling Huang for their continuous collaboration, advice and friendship over many projects (including this thesis) and years of graduate school.

Outside of research collaboration, I would also like to thank my family, including my wife Jasmin, parents Rodney and Susan, and sister April for their sustaining support, patience, friendship and inspiration over the years. Jasmin has patiently endured and supported my work over countless nights and weekends spent reading, coding and writing. I owe a particular debt to my father Rodney Miller who succumbed to bile duct cancer during the writing of this thesis but lived to enjoy, discuss and appreciate the production of all key results. Rodney's own personal curiosity and passion for engineering research, holding 33 patents as a Distinguished Inventor and Department Head at the Kodak Research Labs, was without doubt the single greatest factor in shaping my own career.

In addition to individuals, there are several institutions without whom this work would not have been possible. I would like to acknowledge the following funding sources: DETER (DHS and NSF Funded Science & Technology Center), State Department Bureau of Democracy, Human Rights and Labor, Freedom2Connect Foundation with funding from the Open Technology Fund administered by Radio Free Asia, Intel Science and Technology Center SCRUB (Secure Computing Research for Users Benefit). The opinions in this thesis are those of its author and do not necessarily reflect the opinion of the US Government, its agencies or any other funding sponsor.

I would additionally like to acknowledge VirusTotal, a subsidiary of Google, which provided all data used in this thesis. VirusTotal's Emiliano Martinez was outstandingly helpful over a period of years in arranging access to the necessary data. I would also like to thank the Intel vLab environment and Brent Forgeron for computational infrastructure support.

This thesis forms one piece of a broader body of work from Berkeley's SecML group lead by Anthony D. Joseph and J.D. Tygar exploring the intersection of security and machine learning. My previous work in the SecML group has included a preliminary description of the detection platform [67] as well as an attack on HTTPS encryption using machine learning to infer the contents of encrypted web traffic [68]. Other work from the SecML group has explored poisoning attacks [75, 99, 100], evasion attacks [76, 98] and explorations of open problems [5, 6].

Chapter 1

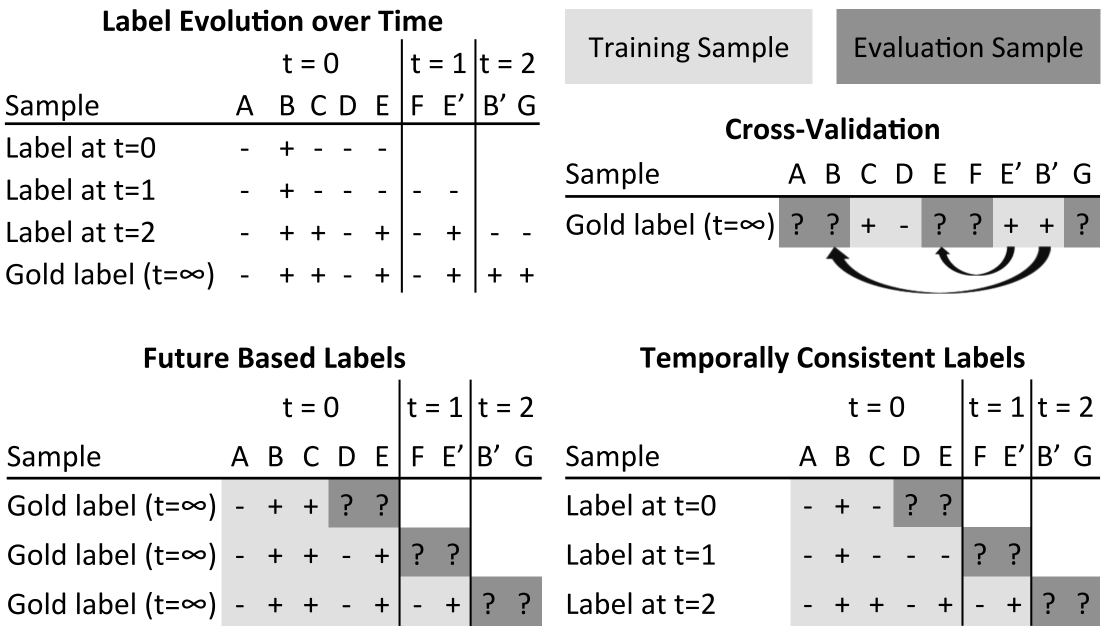
Introduction

This thesis examines the design, implementation and performance of a scalable analysis platform for the detection of malicious content. To reflect the deployment of actual production systems, we design our platform to explicitly model the passage of time and the involvement of human supervisors in the analysis process. This thesis shows how our platform can operate efficiently at a large scale. The thesis presents and evaluates our platform in the context of a case study focused on malware detection.

As the diversity and scope of digital services and technologies has grown, every system component and every aspect of the user experience has been the target of at least some form of malicious content. To cope with the scale, diversity and pace of malicious content, defenders have invoked machine learning to develop models capable of identifying attacks. Researchers have proposed machine learning detection systems for JavaScript [21, 23, 36, 92], PDF [54, 118], mobile [30, 34] and desktop [4, 49, 93, 109] malware detection as well as network intrusion detection [32, 33, 58, 130], spam detection [65, 141] and malicious URL detection [57, 126].

While machine learning has made progress towards timely detection at scale, malicious content presents special challenges for machine learning that do not appear in traditional application domains. Unlike applications such as speech and text recognition where pronunciations and character shapes remain relatively constant over time, malicious content evolves as adversaries attempt to fool detectors. In effect, malicious content detection becomes an online process in which vendors must continually update detectors in response to new threats. This allows a delay during which malicious content will not be detected. For example, a recent study found that only 66% of malware was detected within 24 hours, 72% within one week, and 93% within one month [25]. Malicious content also poses unique labeling challenges. While reading and writing are the only skills necessary to label text, reliably labeling malicious content, such as malware, requires expensive expert analysis.

In this thesis, we present a scalable platform for malicious content detection. Our platform explicitly incorporates the passage of time, allowing experimentation with the impact


Training Sample
Evaluation Sample

Label Evolution over Time

	t = 0					t = 1		t = 2	
Sample	A	B	C	D	E	F	E′	B′	G
Label at t=0	-	+	-	-	-				
Label at t=1	-	+	-	-	-	-	-		
Label at t=2	-	+	+	-	+	-	+	-	-
Gold label (t=∞)	-	+	+	-	+	-	+	+	+

Cross-Validation

Sample	A	B	C	D	E	F	E′	B′	G
Gold label (t=∞)	?	?	+	-	?	?	+	+	?

Future Based Labels

	t = 0					t = 1		t = 2	
Sample	A	B	C	D	E	F	E′	B′	G
Gold label (t=∞)	-	+	+	?	?				
Gold label (t=∞)	-	+	+	-	+	?	?		
Gold label (t=∞)	-	+	+	-	+	-	+	?	?

Temporally Consistent Labels

	t = 0					t = 1		t = 2	
Sample	A	B	C	D	E	F	E′	B′	G
Label at t=0	-	+	-	?	?				
Label at t=1	-	+	-	-	-	?	?		
Label at t=2	-	+	+	-	+	-	+	?	?

Figure 1.1: "Label Evolution over Time" presents the occurrence of samples over three time periods, with samples E and B recurring after their initial appearance. Rows correspond to successive time periods with specified labels, samples appear chronologically from left to right, and + and - denote malicious and benign labels respectively. The remaining subfigures each correspond to different evaluation approaches with varied incorporation of time. Notice that "Cross-Validation" disregards time for binaries and labels, "Future Based Labels" orders binaries chronologically but uses gold labels regardless of time, and "Temporally Consistent Labels" applies labels that are available at the time a binary appears.

on detection performance measurements of different evaluation methodologies. We design our platform to keep pace with evasive data by selecting samples to submit for review and incorporating reviewer labels into training data. We implement our platform using Apache Spark on distributed computational infrastructure, and evaluate our detector in the context of a malware detection case study. The case study demonstrates the ability to process 778GB of data representing 30 months of malware submissions in approximately 12 hours using a 40 core cluster. Applied to our case study dataset, our platform achieves 72% and 89% detection without and with limited support from a labeling oracle at 0.5% false positive rate in an evaluation simulating the emergence of samples and labels over time.

The detection delays caused by adversarially evasive content motivate our examination of different evaluation methodologies. Figure 1.1 presents three different approaches to evaluating detection performance, differing in the extent to which they incorporate the emergence of samples and labels over time. *Cross-Validation*, which selects training samples at random and uses the remaining samples for evaluation, represents the traditional approach to classifier evaluation in settings with independent and identically distributed (i.i.d.) data.

Unfortunately, the i.i.d. assumption does not hold in adversarial contexts since an evasive adversary may change the distribution of the data over time. *Future Based Labels* partitions training and evaluation data according to time, ensuring that training samples predate evaluation samples, but using the best available labels for both training and evaluation. Figure 1.1 refers to the best available label as the *gold label*, collected substantially after the end of the experimentation period. *Temporally Consistent Labels* requires that both training samples and labels predate evaluation samples, effectively modeling the emergence of samples and labels over time as occurs in production settings.

Our comparison of evaluation methodologies reveals that, as expected, evaluations with Future Based Labels produce higher detection results than evaluations with Temporally Consistent Labels, indicating the impact of label quality on detection outcomes. To realize the potential of improved training labels, we integrate a labeling oracle into the detection platform and allow the platform to make limited submissions to the labeling oracle. In practice, the labeling oracle would correspond to a human reviewer capable of manually labeling selected samples. We evaluate the impact of the labeling oracle by evaluating with temporally consistent labels and selectively revealing the gold (i.e. future) label for samples which the oracle labels. Our evaluation reveals that by selectively querying the expert labeler and integrating the query results into a customized training process, we can achieve detection performance comparable to using gold labels for all training samples.

We implement our platform in approximately four thousand lines of Python. Our implementation leverages Apache Spark for distributed computation and HDFS for distributed storage [115, 147]. Spark supports both batch and interactive computation and our platform integrates well with IPython, a browser based user interface for interactive data exploration and experimentation [87]. Implementation in Python also allows us to use the Python scientific stack, including NumPy, SciPy and scikit-learn for numerical operations, sparse matrix representation and machine learning respectively [43, 82, 135].

We evaluate our detection platform using a case studying containing over one million binaries submitted to VirusTotal, a malware repository [133]. VirusTotal applies a suite of static and dynamic analyses, including static malware detectors, to each sample upon submission. Our dataset includes the detection results for each submission of each of binary to track binaries over time and we weight binaries according to number of submissions. Furthermore, since all scan results include a timestamp we are able to conduct evaluations with temporally consistent labels capturing the emergence of label knowledge over time.

The experimental results from our case study demonstrate the following key contributions:

- We demonstrate the previously unstudied impact of temporally consistent labels on evaluation results. At a 0.5% false positive rate, temporally consistent labels yield a 72% detection rate, as compared with 91% and 92% detection rates for future based labels and cross-validation, respectively.

- Our detector integrates a labeling oracle 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, operating at 0.5% false positive rate with 80 oracle queries daily on average, our system detects 42% of malicious binaries initially undetected by all other vendors in our evaluation.

- We demonstrate the scalability of an open-source detection platform using 778GB of data representing over 1.1 million samples appearing between January 2012 and June 2014. Our code, 3% of data, and all binary hashes are available online.

We use our evaluation method and dataset to understand the detection performance of our platform 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, VirusTotal detectors operate statically, so 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 use arbitrary static features derived from the sample and we restrict ourselves to static attributes available through VirusTotal. Additionally, vendors are free to conduct dynamic analysis in the process of static signature generation, causing static signatures to reflect knowledge which can only be obtained through dynamic analysis. Our performance is slightly impacted, yielding a 84% detection rate at a 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 a 85% detection rate at a 1% false positive rate, as compared to a 90% detection rate using a perfect oracle.

The remaining chapters of the thesis examine prior work, platform design and implementation, case study and experimental results. In Chapter 2 we present prior work related to malicious content detection with particular focus on evaluation methodology and integration of expert review. Chapter 3 presents the detection platform design, and Chapter 4 presents the corresponding implementation. In Chapter 5 we present an overview of the case study dataset and design of portions of the detection platform which are application domain specific. Chapter 6 presents the impact of different evaluation methodologies as well as the scalability and detection performance of our platform. In Chapter 7 we discuss open problems and conclude.

Portions of the work reported in this thesis have previously been reported in preliminary form in earlier papers: [45, 46, 67].

Chapter 2

Background and Prior Work

In this chapter we present prior work on malicious content detection. Since our platform is domain independent, we review prior work from diverse application domains including x86 binary, PDF, JavaScript, ActionScript, Android and URL malicious content detection. We discuss prior work incorporating the progression of time or prioritization of expensive resources in greater depth given the increased relevance to our own contributions.

Each section within this chapter focuses on a specific area within prior work. In Section 2.1 we discuss the use of signatures for malicious content detection. While signatures offer low false positive rates and remain prevalent in practice, signatures fail to generalize across evolving threats and thereby enable attackers to evade detection. Section 2.2 presents prior work that does not incorporate the passage of time or prioritize resource consumption. Since our work focuses on the progression of time and integration of a human labeler, we present only a brief summary of each work. In Section 2.3, we closely examine prior work integrating the passage of time into design and evaluation. Our discussion includes any work that models the introduction of new data over time, even if the underlying dataset does not actually include timestamps. Lastly, Section 2.4 discusses prior work that prioritizes consumption of expensive resources, including expert analysis.

2.1 Signature Based Malicious Content Detection

In contrast to our work on machine learning for malicious content detection, many systems use signatures matching specific classes of malicious content [16, 119, 47]. We discuss signatures briefly since signatures have limited relevance to our contributions but motivate the need for machine learning content detection.

Signatures remain prevalent due to practical advantages in deployment settings. The targeted nature of signatures simultaneously offers low false positive rates and reliable detection of content matching the specific profile specified by the signature. Unfortunately, the specificity of signatures corresponds with an inability to generalize across new variants of malicious content. Adversaries can leverage the brittleness of signatures to lower detection rates by introducing new variants of malware designed to evade present signatures and require signature updates. Christodorescu et al. demonstrate the lack of generalization of signatures in an industry malware scanner by introducing code obfuscations that decrease detection to only 5% [18].

In response to evasive activity, prior work has proposed approaches for improving the performance of signature based detectors. Christodorescu et al. demonstrate an approach using behavior templates to incorporate instruction semantics to detect malicious program traits. Behavior templates specify generalized behaviors which a malware instance must engage in, independent of specific instruction or argument values, using sequential manipulations of variables and symbolic constants [18]. Bonfante et al. propose a signature approach utilizing control flow graphs for increased generality across malware variants. Control flow graphs offer a higher level of abstraction, and consequently greater resilience against adversarial manipulation [7]. Venugopal et al. present a signature approach crafted specifically to the needs of mobile devices by reducing memory requirements [132].

Despite proposals for improved signature techniques, actual performance continues to stagnate. Malware creators introduce large volumes of obfuscated malware designed to evade signature based approaches and overwhelm the industry's ability to produce signatures. McAfee alone receives over 300,000 new malicious binaries daily [63]. Damballa examines the performance of malware detectors, finding that only 66% of malware was detected within 24 hours, 72% within one week, 93% within one month and the remainder over the coming months [25]. In response, we present a detection system designed to keep pace with the the introduction of new malware by combining timely expert review of select samples with increased generalization from machine learning detectors.

2.2 Time and Resource Prioritization Agnostic Systems

We begin our discussion of machine learning for malicious content detection with prior work that does not recognize the passage of time or prioritize consumption of expensive system resources. Given the limited relevance to our own contributions, we summarize these works in Table 2.1 and present a more complete summary including the sources for benign and malicious samples in Table A.1.

The passage of time introduces constraints on real-time systems that do not apply to systems operating on data in retrospect. For example, a system that attempts to classify new malware instances must operate using solely information from the past. However, an evaluation attempting to classify the same malware instance one year after the instances has appeared may use information from after the instance appears. We refer to evaluation methodologies that fail to incorporate the passage of time as as *temporally inconsistent evaluations*. Temporally inconsistent evaluations follow two structures: *cross-validation* and *random split*. Cross-validation evaluations randomly partition data into k splits, iteratively

Author	Year	Domain	Structure	# Benign	# Malicious
Schultz et al. [109]	2001	x86	5-fold cross-validation	1,001	3,265
Shih et al. [114]	2005	x86	4-fold cross-validation	335	1,436
Reddy et al. [91]	2006	x86	10-fold cross-validation	250	250
Masud et al. [61]	2007	x86	3-fold cross-validation	1,967	1,920
Stolfo et al. [120]	2007	PDF	random split, 80% training	362	616
Ye et al. [144]	2007	x86	10-fold cross-validation	12K	17K
Masud et al. [62]	2008	x86	3-fold cross-validation	1,967	1,920
Ye et al. [142]	2008	x86	10-fold cross-validation	12K	17K
Menahem et al. [64]	2009	x86	10-fold cross-validation	23K	7,690
Santos et al. [105]	2009	x86	random split, 66% training	1,000	1,000
Schmidt et al. [108]	2009	Android	10-fold cross-validation	100	240
Shafiq et al. [113]	2009	x86	10-fold cross-validation	1,000	16K
Tabish et al. [123]	2009	x86	random split, 50 training samples	1,800	10K
Tian et al. [127]	2009	x86	5-fold cross-validation	161	1,206
Ye et al. [145]	2009	x86	random split, 25% training	8K	32K
Rieck et al. [92]	2010	JavaScript	random split (10x), 75% training	220K	609
Santos et al. [104]	2010	x86	random split, *unspecified*	13K	13K
Alazab et al. [1]	2011	x86	k-fold cross-validation	15K	51K
Chau et al. [16]	2011	x86	random split, 90% training	†	†
Curtsinger et al. [23]	2011	JavaScript	random split (5x), 25% training	8K	919
Laskov et al. [54]	2011	PDF	2-fold cross-validation	40K	26K
Santos et al. [103]	2011	x86	random split, 10-90% training	1,000	1,000
Canali et al. [10]	2012	x86	10-fold cross-validation	‡	7,200
Maiorca et al. [59]	2012	PDF	random split, 50% training	9,989	11K
Overveldt et al. [81]	2012	ActionScript	random split, 5% training	691	1,184
Sahs et al. [101]	2012	Android	k-fold cross-validation	2,081	91
Sanz et al. [107]	2012	Android	10-fold cross-validation	357	249
Tahan et al. [124]	2012	x86	random split (10x), 50% training	2,627	849
Dahl et al. [24]	2013	x86	random split, 90% training	817K	1.84M
Gascon et al. [30]	2013	Android	random split, *unspecified*	136K	12K
Stringhini et al. [121]	2013	URLs	10-fold cross-validation	510	1,854
Markel et al. [60]	2014	x86	random split, *unspecified*	42K	123K

†Evaluation uses approximately 900 million files collected from users between 2007 and 2010, but does not indicate exactly how many are benign or malicious.

‡Evaluation uses 180GB of execution traces from 10 real-world machines, as well as 36 known-benign samples.

Table 2.1: Prior work on malicious content detection that does not incorporate progression of time or resource prioritization. *Structure* refers to the type of evaluation used. For entries of the from "random split (Nx), P% training", N (if present) refers to the number of times the trial is repeated and P refers to the percent of data used in training. "k-fold" indicates the number of folds in the cross-validation was not specified.

training on $k-1$ splits and evaluating on the remaining split until all splits have served as evaluation data. Random split evaluations divide data randomly into training and evaluation data. In some cases random split evaluations repeat multiple times, each time dividing data irrespective of previous partitions. Since neither cross-validation nor random split evaluations incorporate the inherent order in the appearance of samples, both assume that the distribution of malicious and benign content remains static over time. The static distribution assumption does not hold in the adversarial context as the attacker may react and modify content in response to deployed detection mechanisms.

Separate from temporal concerns, assumptions affecting resource prioritization also impact system accuracy in evaluation. For example, the availability of accurate training labels for all data requires unbounded reviewer resources as well as perfectly accurate reviewers. Designs that explicitly account for resource scarcity allow for more accurate evaluation by recognizing resource scarcity.

Due to the large scale of the evaluation, Chau et al. merit discussion beyond Table 2.1. Chau et al. present Polonium, a malware detection system leveraging relationship information in a bipartite graph associating files with end hosts. The evaluation uses over 900 million samples submitted to Symantec by users from 2007 through 2010. As a reputation system, Polonium requires a global view of reputation data unavailable to most detection systems. Additionally, detection performance varies with file prevalence. Treating malicious content as the positive class, detection drops below 50% true positive rate at 1% false positive rate for files observed less than four times.

2.3 Prior Work Incorporating Time

In the previous section, we presented work which randomly partitions training and evaluation data and assumes accurate labels are available during training. Assumptions that accurate labels are readily available and that training data are distributed identically to production data are incongruous with the reality of malicious content detection. If accurate labels were readily available, the detection problem would be solved and there would be no need for research on malicious content detectors. In this section, we discuss prior work that incorporates the effect of time on training samples, training labels and evaluation labels.

We refer to accurate incorporation of the effects of time on training samples and training labels as *temporally consistent evaluation.* Realization of temporally consistent evaluation requires two components. *Temporal sample consistency* requires that training samples predate evaluation samples. Violations of temporal sample consistency during detector evaluation are equivalent to using information from the future when attempting to detect malicious content and lead to misrepresentations of detector performance. Similarly, *temporal label consistency* requires that all training labels predate all evaluation samples. Violations of temporal label consistency effectively use labels from the future.

Separate from the constraints which the progression of time places on training samples and labels, the progression of time also impacts evaluation labels. Just as accurate labels are

Author	Year	Objective	Collection Period	Delay	Stabilization Claim	# Vendors
Bailey et al. [4]	2007	Family Name	7 Months	13 Days	No	5
Rieck et al. [94]	2008	Family Name	5 Months	4 Weeks	No	1
Perdisci et al.† [86]	2010	Detection	6 Months	<1 Month	Yes	1
Rieck et al. [93]	2011	Family Name	7 Days	8 Weeks	No	1
Smutz et al. [116]	2012	Detection	1 Week	10 Days	Yes	5
Rajab et al. [90]	2013	Detection	1 Day	10 Days	No	45
Damballa et al. [25]	2014	Detection	10 Months	6 Months	Yes	4

†Collection occurs from February - July 2009 with detection in August 2009. Perdisci et al. report monthly detection results for the best of three vendors included in the evaluation, demonstrating a decrease in detections in more recent months but not claiming labels are stabilized in any month.

Table 2.2: Prior work recognizing or examining the delay required for accurate vendor labels. *Objective* indicates the type of label required by the work, with *Family Name* corresponding labels distinguishing various types of malicious behavior, and *Detection* corresponding to works distinguishing malicious from benign content. *Stabilization Claim* indicates whether the work claims that the labels ceased to change after the delay period.

not immediately available for training data, the quality of evaluation labels also improves with time. Prior work recognizing the effect of time on evaluation labels inserts a delay between the collection and labeling of evaluation data to ensure opportunity for malware detectors to update signatures and detect all malicious samples.

We begin by discussing prior work involving a labeling delay between collection and labeling of evaluation data, including work that directly measures the delay in vendor labels. Then, we discuss work that moves towards and work that achieves temporal sample consistency. Lastly, we discuss prior work involving temporal label consistency.

Label Availability Delay

We now discuss work which either incorporates or measures the delay in availability of accurate evaluation labels for malicious content. Evaluations may incorporate labeling delay by inserting a waiting period between the collection and labeling of evaluation data. Some prior work allow a fixed delay for labeling evaluation data, while other work periodically updates labels until labels cease to change. Outside of the context of an evaluation, work may directly measure vendor delay in detecting malicious samples. Table 2.2 presents an overview of prior work involving label delay; we discuss each work individually to provide greater insight into current practice and knowledge.

In contrast with our own work on malicious content *detection*, we begin by reviewing work utilizing family labels to *distinguish various types* of malicious content. Bailey et al. present an unsupervised clustering approach for detection of similar malware instances. Although the evaluation includes data from 2004 - 2007, the evaluation primarily uses a dataset collected over seven months and labeled 13 days after the end of collection. The labeling uses five vendors and does not attempt to verify that new detections have stopped emerging in vendor labels [4].

Unlike the unsupervised approach used by Bailey et al., Rieck et al. present a supervised approach to detection of new variants of malware families. Rieck et al. collect malware over a five month period, label the malware using the Avira AntiVir engine, train a model and predict labels for undetected samples. The evaluation demonstrates the ability to predict labels for samples that the Avira AntiVir engine does not initially detect, implicitly recognizing the delay in vendor labels. Four weeks later the Avira AntiVir engine rescans all samples to evaluate model predictions on undetected samples [94].

In further work, Rieck et al. integrate unsupervised methods for the detection of new malware families. Using a similar structure to prior work [94], data collection occurs over a seven day period followed by labeling with the Kaspersky Anti-Virus engine, model training and prediction of unlabeled samples. Eight weeks later, the Kaspersky Anti-Virus engine rescans all samples to evaluate the correctness of model predictions [93].

Separate from work utilizing labels to distinguish families of malware, prior work also demonstrates a delay in binary labels distinguishing benign and malicious content. Smutz et al. study detection of malicious PDF documents and collect evaluation data over a period of one week. Unlike prior approaches, the evaluation scans samples daily using detectors from five vendors to monitor the emergence of new detections. After 10 days, new detections no longer emerge and Smutz et al. consider all labels final.

Rajab et al. present an approach to malware detection using sample metadata and evaluate the approach against labels from a proprietary malware detector. To validate the proprietary malware detector, Rajab et al. select 2,200 binaries from a single day that are not found in VirusTotal and submit the binaries to VirusTotal. After 10 days, VirusTotal rescans the 2,200 binaries to provide an evaluation labeling for the private malware detector. The evaluation makes no claim that all malware detection occurs in the 10 day period, and does not rescan samples later to check for new detections [90]. Note that since the VirusTotal detection signatures operate statically, they may lag behind full anti-virus product signatures that include dynamic sample attributes. Additionally, vendors may tune their detection engines differently in the VirusTotal context relative to corporate products leading to further differences in detection.

Prior work also approaches detection latency from the perspective of evaluating vendors rather than evaluating a research system. As part of a study identifying malware generating HTTP traffic, Perdisci et al. collect a dataset comprised entirely of known malicious samples from February - July 2009 and scan all samples using detectors from three vendors in August 2009. The vendor detecting the most samples detected 93.1% of samples from February but only only 70.2% of samples from July, demonstrating a decrease in detection for more recent samples. While Perdisci et al. do not claim that detections have stabilized six months after the beginning of collection, their analysis does show that detections increase over time and

Author	Year	Domain	Structure	# Benign	# Malicious
Lee et al. [55]	1999	Network	Family Aware Evaluation	n/a	n/a
Henchiri et al. [37]	2006	x86	Family Aware Evaluation	1,488	1,512
Moskovitch et al. [74]	2007	Network	Family Aware Evaluation	n/a	n/a
Bach et al. [3]	2008	x86	Randomized Sample Order	1,971	1,651
Gavrilut et al. [31]	2009	x86	Separate Datasets	280K	39K
Cova et al. [21]	2010	JavaScript	Separate Datasets	11K	823
Heiderich et al.† [36]	2011	JavaScript	Separate Datasets	62K	81
Nissim et al. [79]	2012	Network	Family Aware Evaluation	n/a	n/a
Nissim et al. [78]	2014	PDF	Randomized Sample Order	5,145	1,629
Nissim et al. [80]	2014	x86	Randomized Sample Order	22,735	7,688

†Heiderich et al. use separate datasets for benign data only and divide malicious data randomly for training and testing.

Table 2.3: Prior work on malicious content detection moving towards temporal sample consistency. For work with separate datasets, the benign and malicious sample counts reflect the datasets combined. Since individual instances are less well defined in continuous network data than discrete malware samples, we indicate the numbers of benign and malicious samples as "n/a" for work in the Network domain.

the most dramatic increases occur in the several months after samples appear.

In a similar approach to Perdisci et al., Damballa collects samples from January - October 2014 and regularly scans the samples with four vendor products. Damballa finds that only 66% of malware was detected within 24 hours, 72% within one week, 93% within one month and the remainder over the next six months [25].

Towards Temporal Sample Consistency

We now discuss prior work moving towards temporal sample consistency. Adherence to temporal sample consistency requires that training samples predate evaluation samples. Since accurate timestamps are not always available to order samples, prior work has developed a range of approaches designed to mimic the phenomenon of new samples emerging over time.

Our discussion of prior work approaching temporal sample consistency identifies three common approaches shared by multiple works. Table 2.3 presents an overview of each work and the approach taken to simulate temporal sample consistency, and Table A.2 presents a more complete summary including data sources. The three approaches we identify are as follows:

Family Aware Evaluation

Prior work has simulated adaptive adversaries by introducing families of malicious content in evaluation data that are absent from training data. Some works partition content such that the entirety of each family falls within a single partition, and then conduct a *family aware* cross-validation where families in evaluation data are completely absent from training data. Alternately, other work maintains a fixed set of malicious content that appears in evaluation data only.

Randomized Sample Order
Prior work has randomly ordered samples to simulate the appearance of new samples over time, allowing training data to iteratively expand as samples first appear in evaluation and then transition to training data. Unfortunately, since random ordering does not correlate to the true chronological order, the introduction of new phenomenon over time is limited since phenomenon may appear in multiple samples. In effect, randomized sample order is similar to random selection of training data where the amount of training data varies.

Separate Datasets
As an alternative to randomized ordering or partitioning according to family, some work uses training and evaluation data from separate sources. Although the chronological relationship of the data is unspecified, differences in distribution between the two sources may introduce phenomenon in the evaluation data that are not present in the training data.

The DARPA 1999 network security challenge used family aware evaluation. The challenge solicited network intrusion detection systems and evaluated the systems using synthetic network data. The data contained 38 unique types of attacks spread over nine simulated weeks of network traffic, with the final two weeks serving as unlabeled test data and containing 14 attacks not observed over the first seven weeks. Lee et al. present a detection system designed against the DARPA 1999 data, achieving 80.2% detection of attacks in training data, but only 37.7% detection of new attacks appearing in the test data only.

In the context of malware detection, Henchiri et al. conduct a family aware evaluation by dividing data into five folds according to malware family and conducting a family aware five-fold cross-validation [37]. The isolation of each family to an individual fold ensures that evaluation data will contain novel malicious content not available in training data. Although family labels guide the division of malicious samples, the evaluation splits benign samples randomly between each fold. The evaluation uses 1488 benign samples collected from Windows as well as 1,512 previously labeled viruses released by Schultz et al. [109]. Using the family aware evaluation, Henchiri et al. achieve results varying from 37% detection at 0.16% false positive rate to 93% detection at 5% false positive rate, depending on the learning algorithm. The evaluation does not include a comparison of family aware and traditional cross-validation approaches.

Moskovitch and Nissim et al. evaluate an approach to worm detection using active learning with family aware evaluation [74, 79]. The dataset includes synthetic network traces from five worms as well as clean traffic from an isolated environment including hosts with eight different configurations varying hardware, background applications and user activity. The evaluation divides data according to the presence of specific worms, such that all instances with the same worm appear in either training or evaluation data. The authors vary the number of worms included in training data, and find that training data containing instances from more types of worms produces superior models compared to less diverse training data.

As an alternative to partitioning data based on family labels, some prior work has used a randomized sample order to simulate the appearance of new samples over time. Bach and Maloof present a technique for learning in the presence of concept drift known as *paired learners* in which one model trains over all past data and another model trains over recent data only [3]. They evaluate the paired learning approach on five separate datasets, including the dataset used by Kolter and Maloof [50, 49]. Although the paired learner approach produces positive results on some datasets, the malware dataset yields the same performance as standard learning techniques. Although Bach and Maloof recognize the potential for concept drift in malware data, the data itself fundamentally lacks drift because the samples are merely placed in randomized order.

Similarly to Bach and Maloof, Nissim et al. evaluate approaches to malicious PDF and x86 binary detection by randomly dividing evaluation data to form ten synthetic days [78, 80]. The evaluation proceeds chronologically, successively training a model on days 1 through N and evaluating the model on day $N + 1$. Since the chronological divisions in the data are artificial, the evaluation is equivalent to randomly partitioning training and evaluation data and successively increasing the amount of training data. Accordingly, detection accuracy improves over successive days as the amount of training samples increases. Nissim et al. do not compare the results from their evaluation with a cross-validation evaluation.

In cases where multiple datasets that do not include timestamps are available, training and evaluating on separate datasets can increase the potential for differences between the training and evaluation data. Gavrilut et al. conduct a study of x86 malware detection in which the training and evaluation data are drawn from separate data sources [31]. The training data comes from Virus Heaven and includes 273,133 benign files as well as 27,475 malicious files. The evaluation data comes from the Wildlist collection, including 11,605 malicious files and 6,522 benign files. Although Gavrilut et al. do not make an assertion about the chronological relationship of the datasets, the composition of malware families varies between the datasets with the training data including more backdoors and fewer worms than the testing data. The evaluation uses cross-validation only to fit learning parameters to the training data and draws accuracy conclusions from the evaluation dataset. Consistent with expectations, the detector achieves higher accuracy during cross-validation on the training dataset than during evaluation on a separate dataset.

Similar to Gavrilut et al.'s study of x86 detection, Heiderich et al. present an approach to JavaScript detection which maintains limited distinction between training and evaluation data. Heiderich et al.'s evaluation uses a set of 61,554 benign instances obtained by visiting

Author	Year	Domain	Structure	Duration	Interval
Kolter et al. [49]	2006	x86	Chronological Datasets	n/a	n/a
Moskovitch et al. [72, 71]	2008/9	x86	Periodic Retraining	8 Years	1 Year
Perdesci et al. [86]	2010	Network	Periodic Retraining	6 Months	1 Month
Shabtai et al. [112]	2012	x86	Periodic Retraining	8 Years	1 Year
Smutz et al. [116]	2012	PDF	Chronological Datasets	n/a	n/a
Stokes et al. [119]	2012	x86	Periodic Retraining	13 Months	1 Month
Srndic et al. [118]	2013	PDF	Periodic Retraining	14 Weeks	1 Week

Table 2.4: Prior work on malicious content detection with temporal sample consistency. *Chronological Datasets* indicates separate training and evaluation datasets where training data entirely predates evaluation data. *Periodic Retraining* indicates a single dataset with timestamped samples divided into intervals where model training proceeds through successive intervals with evaluation occurring on the upcoming interval.

Alexa traffic ranking and sanitizing results as well as 81 malicious instances obtained from a blacklist. The top 50 Alexa sites supply benign instances for training, leaving the remaining 61,604 instances for evaluation. By selecting benign training instances based on the Alexa ranking criteria, Heiderich et al. introduce a distinction between the training and evaluation datasets. Although benign instances are not selected randomly, Heiderich et al. do select 30 malicious instances for training at random and leave the remaining 51 for testing.

Anomaly detection presents a natural opportunity for distinction between training and evaluation data since the training process does not include any instances from the malicious class. Cova et al. present a malicious JavaScript detection system based on anomaly detection. The training process only requires a benign dataset and evaluation uses six separate datasets, including four which are malicious and two which are unlabeled. The authors build the benign training data set from live webpages over the two years preceding publication. Each of the remaining datasets are generally collected over a period of several months overlapping with the benign dataset collection. The authors limit use of the training dataset in the evaluation to assessing false-positive performance by doing a cross-fold evaluation. The authors use separate data to conduct additional evaluation of false positives as well as all evaluation of false negatives [21].

Temporal Sample Consistency

Temporal sample consistency requires that training samples predate evaluation samples. Evaluations can maintain temporal sample consistency by using distinct, chronologically separated datasets for training and evaluation. Alternately, evaluations may use a single dataset with timestamped samples and simulate the passage of time by iteratively transitioning samples from evaluation data in the simulated present to training data in the simulated past. Simulated passage of time corresponds to reality more closely since any chronological gap between training and evaluation is minimal and detectors retrain as new threats emerge. We begin by discussing prior work using separate datasets, and then discuss work that conducts periodic retraining.

In a continuation of prior work using a cross-validation evaluation [50], Kolter and Maloof present the first work on x86 malware detection with temporal sample consistency [49]. The temporally consistent training data comes from the original work, including 1,971 benign executables obtained from Windows installations and 1,651 malicious executables obtained from Virus Heavens and the MITRE corporation [50]. All training data collection occurred during summer 2003. For the purposes of chronological evaluation, Kolter and Maloof obtain an additional dataset from Virus Heavens in August 2004 containing 3,082 new executables. According to unspecified data from various websites, 291 of the 3,082 samples in the new dataset appear after July 2003, postdating the training dataset. Since Kolter and Maloof do not present details of the sample dating process, external auditors cannot verify the effectiveness or correctness of the process. An evaluation using the 291 temporally consistent samples finds that the model detects 98% of samples appearing after July 2003 at a 5% false positive rate. As expected, the temporally consistent performance represents a slight decrease from the 99% detection achieved at 5% false positive rate when conducting a cross-validation evaluation using the original dataset.

In the PDF detection domain, Smutz et al. present an evaluation with separate training and evaluation datasets maintaining temporal sample consistency [116]. The training dataset comes from Contagio and contains exactly 5,000 benign and 5,000 malicious samples. The evaluation dataset contains exactly 100,000 samples from monitoring of HTTP and SMTP traffic on a live network over one week. Smutz et al. label the evaluation dataset using five anti-virus scanners and observe that new detections stop appearing 10 days after the end of collection. The authors state that there is no overlap between the samples of the two datasets, and that the training dataset occurs "months" before the testing dataset. The evaluation compares a 10-fold cross-validation on the training data with performance on the evaluation data, and observes approximately 95.3% detection at 0.14% false positive rate on the testing data compared with 99.9% detection at 0.3% false positive rate during cross-fold evaluation.

Although the collection of chronologically separated training and evaluation datasets is sound, techniques that model the passage of time including periodic retraining correspond more closely to the maintenance of production systems. We now examine prior work maintaining temporal sample consistency using a single, timestamped dataset.

Moskovitch and Shabtai et al. present the first work examining temporal sample consistency with periodic retraining for an x86 malware dataset [72, 71, 112]. The dataset contains 7,688 malicious samples from Virus Heavens and 22,735 samples from Windows system files. The file creation dates serve as timestamps to order samples, resulting in a progression of samples spanning from 2000 through 2007. The evaluation proceeds chronologically, iteratively adding one year to training data and evaluating performance on all remaining years.

The results demonstrate the importance of temporal sample consistency, with detection performance generally decreasing as a model ages.

Also in the x86 malware domain, Stokes et al. examine a system designed to detect malware using telemetry data sent back to Microsoft from end hosts [119]. The evaluation initially uses a five-fold cross-validation to tune system parameters, and then measures actual system performance using periodic retraining over 18 months of data. During the temporally consistent portion of the evaluation the classifier trains using the past five months of data and predicts data in the present month, iteratively advancing through 13 months of evaluation data. In the interest of comparison, the authors also conduct a cross-validation evaluation using the five months of training data. The evaluation demonstrates that cross-validation consistently results in lower false negative and false positives than evaluation with temporal sample consistency.

Outside of x86 host-based malware detection, Perdisci et al. present a technique for clustering malware HTTP traffic and automatically extracting signatures from the clusters [86]. The evaluation data includes six months of malicious traffic samples from a malware execution environment divided into one month intervals. The detector retrains each month by clustering all available training data and extracting signatures, and the evaluation applies signatures from each month over all remaining months. Consistent with expectations, the evaluation demonstrates that signature detection rates decrease over time as the signature becomes older. To assess system false positives, the evaluation also applies signatures to benign traffic collected from a 2-day period before the beginning of the study.

In the PDF detection domain, Srndic and Laskov present a detector operating on static features [118]. In addition to cross-validation, the evaluation also conducts periodic retraining maintaining temporal sample consistency over a 14 week period with weekly retraining. However, since the evaluation uses multiple datasets and does not directly compare cross-validation and temporally consistent evaluation on the same data, the exact impact of temporally consistent evaluation remains unclear.

Temporal Label Consistency

Thus far, within the context of temporal awareness, we have focused on temporal sample consistency. We now examine prior work involving temporal label consistency, which requires that both training samples and labels predate evaluation data.

Evaluations of deployed systems operating in real-time must maintain temporal label consistency since knowledge from the future is not available in practice. The Content-Agnostic Malware Protection (CAMP) system presented by Rajab et al. protects Chrome users from malicious downloads using reputation data [90]. The evaluation covers a six month period in which CAMP serviced eight to ten million reputation requests daily. Rajab et al. determine ground truth for the purposes of evaluation using a proprietary dynamic analysis pipeline, and validate the pipeline itself by submitting samples to VirusTotal and observing detection

Author	Year	Domain	Prioritization Strategy	# Benign	# Malicious
Provos et al. [89]	2008	URLs	Weak Detection	1M	25K
Perdisci et al. [85, 84]	2008	x86	Process Optimization	2,900	2,598
Hu et al. [38]	2009	x86	Similarity Detection	0	102,391
Neugschwandtner et al. [77]	2011	x86	Reward Estimation	0	643,212
Canali et al. [11]	2011	URLs	Weak Detection	19M	132K
Jacob et al. [41]	2012	x86	Similarity Detection	0	794,665
Chakradeo et al. [15]	2013	Android	Weak Detection	51,407	923

Table 2.5: Prior work exploring prioritization of computational resources in the context of malicious content detection and analysis.

results 10 days later. While a 10 day delay recognizes some labeling delay, Perdisci et al. and Damballa demonstrate that label updates continue for months after a sample first appears [86, 25]. Consequently, although Rajab et al. maintain temporal label consistency, the ground truth labels for performance evaluation likely contain errors due to insufficient labeling delay. Since the CAMP evaluation maintains temporal label consistency due to practical constraints, the authors do not present or develop the concept of temporal label consistency nor measure the impact of temporal label consistency.

Separate from evaluation of deployed systems, systems that iteratively retrain on their own predicted labels also maintain temporally consistent labels. Schwenk et al. evaluate a malicious JavaScript detector which retrains using previous predictions, and find that the detector performs better when training on labels assigned in retrospect using external knowledge [110]. Notice that while retraining on predicted labels is temporally consistent, excluding the introduction of external knowledge which would be available in practice artificially deflates system performance.

2.4 Resource Prioritization

In our work, we present a strategy for prioritizing access to a human expert labeler. Since prior work examining integration of human reviewers is limited, in this section we discuss work examining *resource prioritization* by modulating access to any expensive system resource including dynamic analysis, dynamic unpacking and human review.

Prioritizing Use of Computational Resources

We now discuss prioritization of computational resources. Table 2.5 presents an overview of prior work on prioritization of computational resources. We categorize the prioritization strategies proposed in prior work as follows:

Weak Detection

In some cases, noisy detection may be possible through inexpensive means such as static analysis. Weak detection uses a noisy classifier to identify samples that are likely benign so that expensive analysis may be reserved for samples that may be malicious.

Process Optimization

While weak detection attempts to directly estimate the maliciousness of a sample with a noisy detector, process optimization trains a detector to identify samples that may circumvent expensive steps in the analysis process. For example, binaries that are not packed do not need to undergo unpacking.

Similarity Detection

Given that malicious content creators may perturb content to evade detection, similarity detection attempts to identify similar instances to avoid unnecessarily repeated analysis.

Reward Estimation

As an extension of similarity detection, reward estimation attempts to estimate the additional benefit of analyzing a sample given the results of previous analysis. For example, execution of malware that regularly changes C&C servers will reveal new C&C server information, but execution of malware with a static C&C server is less likely to produce new knowledge.

The enormous scale of the web combined with the appreciable cost of loading a single web page in an actual browser creates considerable incentive to prioritize URLs for dynamic analysis. Provos et al. present a weak detection approach to prioritize dynamic analysis of URLs for drive-by-download detection [89]. Provos et al. demonstrate that their approach to prioritization offers a 0.1% false positive rate and 90% true positive rate, effectively submitting one million URLs for analysis daily to yield 25 thousand malicious URLs. When evaluating detection the evaluation uses five-fold cross-validation, which may inflate performance results by disregarding the inherently temporal nature of the problem.

Similar to Provos et al., Canali et al. prioritize URL submissions to a dynamic analysis system without an appreciable increase in false negatives [11]. The evaluation uses 10-fold cross-validation to fit the model on training data, then evaluates performance using a separate data set collected over 15 days. Evaluation on 19 million URLs demonstrates that the prioritization technique isolated malicious URLs to 15% of the original dataset by sampling and scanning the remaining 85% of the dataset with few positive detections. Canali et al. determine ground truth for the evaluation using the Wepawet URL scanner [21].

Outside of the URL context, Chakradeo et al. present Mobile Application Security Triage (MAST), a static approach to prioritizing mobile malware for dynamic analysis [15]. MAST operates four times faster than signature detectors and reduces the need for dynamic analysis by isolating 95% of malware included in the evaluation to 13% of samples. The evaluation trains on benign apps from Google play and malicious apps from public datasets, and evaluates against apps from three markets. The evaluation uses separate datasets but does not maintain temporal sample consistency. Chakradeo et al. label evaluation data in February 2012, one week after the end of collection.

Malware creators often apply *packers* or *crypters* to obscure the contents of binaries and thwart static analysis [2]. Separate from efforts focusing dynamic analysis on the samples most likely to be malicious, Perdisci et al. present a process optimization prioritization strategy to avoid waste when unpacking binaries. Although inexpensive unpackers are available, the most robust and generalized approach to unpacking is to execute the actual binary and image the contents of memory [97]. Since dynamic unpacking is expensive, Perdisci et al. modulate access to dynamic unpacking resources by static detection of packed executables [85, 84]. The evaluation demonstrates significant improvements over PEiD, the state-of-the-art in packer detection, with PEiD failing to detect 30.2% of packed executables and Perdisci et al. detecting 98.9% of executables missed by PEiD. Note that the evaluation separates training and evaluation samples according to which packed executables PEiD detects, and demonstrates higher accuracy using cross-validation on training samples than on the evaluation samples that are undetected by PEiD.

For settings where the analysis objective is broader than binary detection, such as malware family classification, similarity detection attempts to prioritize novel instances. Given the prevalence of polymorphism in malware, techniques identifying related pieces of malware offer potential utility when prioritizing samples for expensive forms of analysis since repeated analysis of similar content may be avoided. Techniques for identifying related content can be associated into two categories: *classification* and *clustering*. Classification techniques operate in a supervised manner, training on labeled samples of different classes of malicious content to identify new variants of known classes. Clustering techniques operate in an unsupervised manner, identifying related samples in unlabeled datasets to group samples into families. Although prior work has applied both classification [140, 106, 69, 51, 44] and clustering [4, 42, 34, 33, 53, 138, 131, 48] to malicious content, none of these works have directly examined the effectiveness of these techniques as a form of resource prioritization.

Departing from traditional supervised classification and unsupervised clustering approach, Hu et al. present an approach to avoid redundant analysis of malicious samples by efficiently detecting similarity to known samples [38]. The similarity computation operates over function call graphs since call graphs are less susceptible to instruction level obfuscation. Although the evaluation demonstrates ability to detect similar samples, Hu et al. do not further examine the integration of their similarity detector into an analysis system.

Similarly, Jacob et al. present an approach to similarity detection which operates over packed binaries to detect similar malware samples [41]. A small scale evaluation uses binaries from clean OS installs and source code for several known malware families to assess accuracy using a labeled dataset, and a larger evaluation uses 794,665 malicious binaries submitted to

Anubis. The larger evaluation considers two binaries to be equivalent based on the results of a clustering using both structural and dynamic properties of the binary. For the larger evaluation, similarity detection reduces the dynamic analysis load by a factor of three to five by preventing repeated analysis of equivalent binaries.

Rather that simply detecting similarity between two binaries, Neugschwandtner et al. present a reward estimation approach prioritizing dynamic analysis to maximize the amount of useful information obtained as a result of the dynamic analysis [77]. For the purposes of evaluation, useful information must aid in either blacklisting C&C servers or generation of malware remediation procedures. Neugschwandtner et al. estimate reward by clustering previous samples using dynamic features, determining the utility of executing additional samples from each cluster, and then building a model to predict the dynamic cluster of a new sample using static features. Since the goal of the work is not detection, all samples used in evaluation were malicious. The evaluation demonstrates improvements with respect to the usefulness metrics over both static clustering and random selection. Neugschwandtner et al. incorporate time in the evaluation by using one month as seed data and then incrementally advancing the system using one day intervals over two months, examining the marginal benefit of sample execution relative to previous knowledge.

Prioritizing Use of Human Reviewers

We now discuss prior work on prioritizing samples for human review. We confine our discussion to work which focuses purely on prioritization and does not present and evaluate a full active learning approach by integrating reviewer labels into detector training.

In the JavaScript context, Karanth et al. present an approach for prioritizing expert analysis with the end goal of identifying novel attacks and generating signatures, not increasing machine learning based malicious content detection [47]. The approach includes a novel feature extraction technique leveraging the co-occurrance of features and statistical methods accounting for the class imbalance in training data. The evaluation includes a training dataset of 13.5K samples collected from July - October and a testing dataset of 27K samples collected from August - November; although the collection periods overlap by several months the datasets do not have any samples in common. To demonstrate effective prioritization, Karanth et al. present the top 2% of prioritized evaluation samples to an expert, who successfully identified multiple zero days within the data. Additionally, a random sampling of 100 instances showed that malicious samples were given the highest priorities. Having submitted samples for expert analysis, Karanth et al. do not experiment with the effects of retraining with the updated expert labels.

In unpublished work, Morales presents an approach to prioritizing samples for human review combining clustering and classification [70]. After clustering to prevent repeated analysis of similar samples, Morales applies a classifier to identify clusters with malicious content. A 10-fold cross-validation evaluation measures classifier effectiveness, achieving area under the ROC curve greater than .97. Although Morales suggests that both samples ranked as the most malicious and the most benign by the classifier may be the best candidates for analysis, Morales does not actually evaluate the impact of a query strategy on subsequent detection results. Since the evaluation does not actually examine the impact of the proposed query strategy on detection results, the evaluation does not establish that Morales has actually proposed an effective approach to prioritizing malware for analysis.

Expert Integration and Active Learning

Machine learning systems that select samples for expert review fall within the field of *active learning*, which offers techniques to improve the performance of machine learning systems by allowing the learner to query an oracle [9]. Although active learning offers various approaches, including querying for labels of synthetic instances and querying the significance of individual features, most work focuses on queries asking the oracle to label samples drawn from a pool of unlabeled instances. We have previously discussed prior work that prioritizes use of both computational resources and human reviewers. We now examine prior work that both involves human experts for review and incorporates the expert analysis into subsequent learning to improve model performance. Additionally, we examine an approach using techniques from active learning to improve performance in the absence of an expert reviewer.

We begin by briefly discussing a system that incorporates expert knowledge but does not suggest samples for the expert to review. Ye et al. present a clustering approach that integrates a "human-in-the-loop" who may constrain the clustering such that specific sets of samples *are* or *are not* in the same cluster [143]. The system iteratively retrains, allowing the human to introduce additional constraints during each retraining cycle. The evaluation reports that expert constraints improve clustering results in an experiment which retrains daily for two weeks but lacks details regarding the number of pairs the expert must constrain or how the expert selects samples to analyze.

The first application of active learning to malicious content detection sought to improve worm detection by improving training data labels without the integration of a human expert. Moskovitch and Nissim et al. use synthetic data collected by recording traffic in an isolated network with eight different node configurations varying hardware, background applications and user activity [74, 79]. The authors record traffic in the presence of five different worms and without any worms present. Since the featurization process divides captured traffic into one second intervals and the worms are not always active, some training instances are effectively mislabeled. Rather than using active learning to label samples, Moskovitch and Nissim et al. use the *expected error reduction* query strategy from active learning to filter mislabeled instances out of training data. Expected error reduction selectively adds samples to training data which increase the confidence of the classifier on remaining samples, effectively avoiding mislabeled samples that tend to decrease classifier confidence [96, 9]. The evaluation partitions training and evaluation data by worm to introduce attacks in evaluation data not present in training data, and reports that expected error reduction does improve performance.

In subsequent work, Moskovitch and Nissim et al. explore active learning in the context of malware detection as a means of building a set of training data from samples presumed to be unlabeled [73]. The authors do not propose a novel query strategy and instead focus on *uncertainty sampling* and expected error reduction. Uncertainty sampling selects unlabeled samples that are closest to the decision boundary [56, 9]. Corresponding with a belief that "the proportion of malicious files in real life is about or less than 10%", Moskovitch and Nissim et al. structure the evaluation dataset to contain 2,500 malicious binaries and 22,500 benign binaries. Although the evaluation maintains a training set, evaluation set and set of potential samples for oracle queries, the sizes, composition and overlap of these sets is unclear. The evaluation does not incorporate the progression of time and uses 10-fold cross-validation to conduct all experiments. The authors report all performance as accuracy, which obscures performance given the class imbalance of the dataset. For example, a classifier that labels all samples benign would appear favorable by achieving 90% accuracy, but actually fail to detect any malware. The evaluation reports that the uncertainty sampling approach improves accuracy from 86.63% to 92.13%, while expected error reduction improves accuracy from 85.80% to 89.05%. Notice that the expected error reduction strategy does not improve accuracy beyond a classifier which always labels samples benign. The authors do not explain why the initial classifier accuracies before active learning are different for the two query strategies. Given the methodology, reported statistics and ambiguous details, the evaluation does not establish the effectiveness of the active learning techniques.

Subsequent work by Nissim et al. revisits the application of active learning to x86 malware detection and introduces active learning for malicious PDF detection [78, 80]. In addition to random oracle queries and uncertainty sampling, the evaluation includes the novel *exploitation* and *combination* query strategies. The exploitation query strategy submits samples to the oracle which are furthest from the decision boundary located on the malicious side and uses the kernel farthest-first method to avoid selecting similar samples. The combination query strategy mixes uncertainty sampling and exploitation, querying samples both close to the decision boundary and far from the decision boundary on the malicious side. The structure of the x86 and PDF malware detection evaluations is similar, randomly dividing samples to simulate 10 days and iterating through the samples while training a new model for each day. The x86 evaluation maintains the 10% malware balance with 2,526 malicious samples and 22,734 benign samples; the PDF evaluation uses 1,629 malicious samples and 5,145 benign samples.

Despite the introduction of novel query strategies and reduced ambiguity in evaluation methodology, Nissim et al. do not demonstrate improvements to automated malware detection for several reasons. First, random division of data to simulate 10 days in the evaluation offers minimal improvement over randomized cross-validation since the random division of samples prevents any concept drift over the simulated days. The primary difference between simulated days and cross-validation is the increase in training data over time, and accordingly performance improves for both x86 and PDF malware detection as more training data becomes available. Second, the primary objective of the work is not to increase detection performance, but rather to increase the number of malicious samples given to the expert for review. Although the evaluations demonstrate that exploitation and combination select more malicious samples for expert review, x86 detection results remain similar and PDF detection results are strictly worse, achieving a lower true positive rate and a higher false positive rate than uncertainty sampling. By adding training samples located far from the decision boundary, exploitation and combination tend to select samples which have low impact on the model and do less to improve performance. Third, all query strategies in the evaluation heavily burden the human reviewer. In the context of x86 malware detection on day 10 using uncertainty sampling, Nissim et al. achieve 30% and 78% true positive rate with 50 and 250 queries daily, respectively. Notice that 250 queries for 10 days amounts to 2,500 queries total, nearly equivalent to the 2,526 malicious samples in the x86 evaluation. In the context of PDF malware detection Nissim et al. only present the results of querying 160 samples daily leading to 1,600 queries total, nearly equivalent to the 1,629 malicious PDF samples in the evaluation.

The most complete depiction of a system integrating human reviewers in an adversarial context comes from Sculley et al. and describes the system used by Google to detect adversarial advertisements [111]. The adversarial advertisement detector combines cascading models targeting different, specific classes of adversarial activity as well as multiple levels of human reviewers with varied expertise. As time progresses, the human experts maintain and monitor the detector by labeling new samples, introducing new rules and models, and monitoring the properties of existing models. Sculley et al. present a careful strategy guiding the integration of human experts to balance competing concerns including catching hard adversaries, improving model accuracy, focusing efforts on ads with high impression counts and monitoring system performance.

While Sculley et al. provide a broad overview of their system and identify important concerns in design and implementation, their work presents their design at a high level and lacks critical details in implementation and evaluation.[1] For example, the paper does not specify how many human reviewers are necessary, the added benefit from additional reviewers or the load individual reviewers are able to handle. Likewise, the paper also does not specify the accuracy of human reviewers or the impacts of reviewer errors. The authors also remain silent on the impact of different query strategies, details of the retraining regiment and techniques for monitoring changes in high dimensional models over time. In effect, Sculley et al. provide an overview of a system integrating human experts in an adversarial context which raises as many questions as it answers.

In unpublished work, Tang and Schneider explore prioritization strategies involving both

[1]Details are likely sparse since this paper was published on the industry track of KDD, where it received the best paper award.

classification and anomaly detection [148]. The evaluation uses 356 malicious samples and 658 benign samples and does not maintain temporal consistency. Although the results include relative effectiveness of different query strategies, aspects of the experimental design including the number of oracle queries, baseline performance without an oracle and learning algorithm remain unclear.

Chapter 3

Platform Design

In this chapter we present the design of our malicious content detection platform. Prior work often treats data, detection system and human supervisor as isolated entities, disregarding adversarial introduction of new content, responsive intervention from human supervisors and necessity of human insight into detector behavior. In contrast, we present a platform that reflects the dynamic nature of malicious content detector deployments and interactive relationship between content detector and human supervisor. The following goals motivate and guide our platform design:

- **Modular Design** The platform design should separate concerns of feature extraction, integration of labeling information from external sources, integration of human reviewer and model training. Separating design concerns allows for testing and improvement of each system component in isolation.
- **Integrated Progression of Time** The platform design should directly reflect the emergence of new samples and refined understanding of sample labels over time. Integrating the progression of time allows for accurate evaluation during system design and effective maintenance during deployment.
- **Scalability** In the interests of both detection performance and deployment, the platform should support resource increases to match increased data. The platform design focuses on the scalability of feature extraction and relies on established software for scalable storage and computation.
- **Interaction With Human Supervisor** The human expert should have an interactive relationship with the detector. The human should inform the detector by labeling samples the detector chooses, and the detector should inform the human by exposing the factors impacting detection results.

Each goal influences aspects of our platform design. To maintain modularity, we decouple generation and labeling of feature vectors into separate series of software modules. To

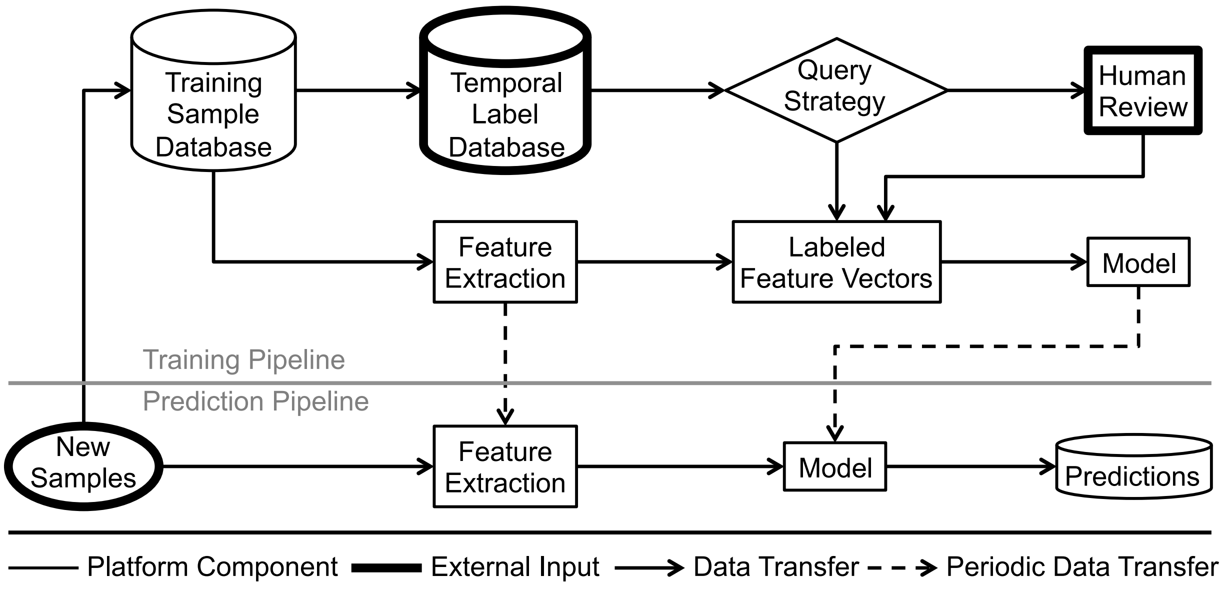


Figure 3.1: The detection platform employs the prediction pipeline to detect malware, and the training pipeline produces the next model and feature extractor for use in the prediction pipeline. During each retraining period, the query strategy reviews all available training data and selects binaries for human review. The results of human review are combined with training data labeled using the current model and temporally consistent labels to serve as the training data for the next model.

incorporate the progression of time we discretize time into epochs, allowing detection models to incorporate new samples in batches rather than individually. The decoupling of feature vectors and labels in modular design also incorporates the progression of time by allowing labeling modules to update sample labels over time. To achieve scalability, we structure our feature extraction process to limit dependencies across samples in support of parallel feature extraction. In addition, our feature extraction integrates caching to expedite computation as new samples arrive during successive epochs. Lastly, we combine externally available knowledge with predictions from the detector itself to guide efforts of the human reviewer, and present a learning algorithm utilizing a linear model in a high dimensional space to achieve accurate and human-interpretable detection results.

In Section 3.1 we present an overview of our modular design, illustrated in Figure 3.1. The remaining sections discuss realization of other design goals in greater detail. Section 3.2 discusses integration of the progression of time and associated requirements for experimental datasets. In Section 3.3 we present details of our feature extraction process for maintaining scalability, and Section 3.4 discusses interaction with the human supervisor.

3.1 Modular Design

We structure the malicious content detection platform to include separate modules for system components to allow testing and improvement of each component in isolation. Figure 3.1 provides an overview of the detection platform and the relationship of each module. Notice that we mark the new sample, temporal label database and human review modules as *external input*. In deployment settings, external input modules receive data from sources outside the detector codebase, such as a human supervisor or user submissions. In development settings, we use data to simulate interactions with external input modules. We discuss the role and relationship of each module below.

We divide the detection platform into a *training pipeline* and a *prediction pipeline*. The prediction pipeline receives new samples, applies the current feature extractor and model, and produces detection results. The training pipeline manages the labeling and model training process, and exports updated feature extractors and models to the prediction pipeline. The dashed lines in Figure 3.1 represent the periodic updates to the feature extractor and model which occur at epoch boundaries. Treatment of the feature extractor and model as dedicated modules facilitates periodic updates incorporating new samples and labels.

The training pipeline begins with the training sample database, which holds all samples previously processed by the prediction pipeline. From the training sample database, samples proceed through both the temporal labeling database and the feature extraction process. During system design, the temporal labeling database assigns labels to samples using historical labeling data. During deployment, the temporal labeling database associates any known labeling information with the sample. After labeling, both the sample and label advance to the query strategy, which selects samples for human review and labels remaining samples using temporal labeling knowledge. During design, we simulate the involvement of a human reviewer by appealing to the most recent labeling data for a sample. Once all samples are vectorized and labeled, the training pipeline produces a model from the labeled feature vectors.

The modular design of our detection platform enables flexibility in design and evaluation of detection systems. Modular design maintains platform generality across application domains by separating feature extraction from both sample storage and learning, allowing for transparent introduction of new feature extraction techniques for data in new application domains. The separation of labeling into multiple modules decoupled from samples maintains flexible labeling practices, allowing variations in human review query policy and accuracy of simulated human reviewers. Lastly, modular design facilitates experimentation with different learning and feature selection implementations, as well as updates to learned models as time progresses.

3.2 Integrated Progression of Time

As motivation and context for our integration of the progression of time, recall that Chapters 1 and 2 place evaluations into three categories according to the incorporation of time. Temporally inconsistent evaluations divide samples regardless of time and label training samples using the best labels available at the time of evaluation. Both cross-validation and random, fixed division of training and evaluation data are examples of temporally inconsistent evaluation. Temporal sample consistency maintains the requirement that training data predate evaluation data, respecting the temporal constraints on the availability of training samples in practice. However, evaluations with temporal sample consistency may use labeling knowledge obtained after samples appear, effectively introducing knowledge from the future into the training process. Lastly, temporal label consistency requires that both training data and labels predate evaluation data. Deployed systems operating in real-time must adhere to temporal label consistency since in practice utilization of samples or labels from the future is not possible.

Our platform discretizes time into epochs to represent the full breadth of temporal evaluation behaviors. Epochs occur sequentially, allowing each epoch to introduce new samples and new labeling knowledge for known samples. As time progresses, the detector trains on samples from the first N epochs to detect malicious content in epoch $N + 1$. Figure 3.1 depicts the release of a new feature extractor and model, which coincides with the beginning of each new epoch. Although sequential epochs are naturally well suited to model the progression of time, the placement of samples within epochs does not need to respect temporal constraints. Epochs may also differ in size to allow variations in the availability of training data. For example, an evaluation that randomly places 80% of samples into the first epoch and 20% of samples into the second implements a random split evaluation with 80% training data.

In addition to flexibly modeling different evaluation methodologies, epochs also facilitate the use of batch learning methods. Batch learning methods require the availability of all training data throughout the training process. Since samples appear sequentially as time progresses, a naive approach retraining after the appearance of each sample would introduce high computational burden. Division of samples into epochs allows for periodic retraining and reduced computational burden.

Online learning algorithms designed for streams of data present an alternative to batch techniques. Online algorithms update the model as each sample appears, removing the need to retain the entirety of training data [8]. In the adversarial context, online algorithms may have a disadvantage since labels available after a sample first appears are less likely to be accurate than labels issued later in time. By processing training samples only once, online algorithms forfeit the ability to benefit from improvements to label knowledge as time progresses.

Having established and motivated the design structures which our platform uses to in-

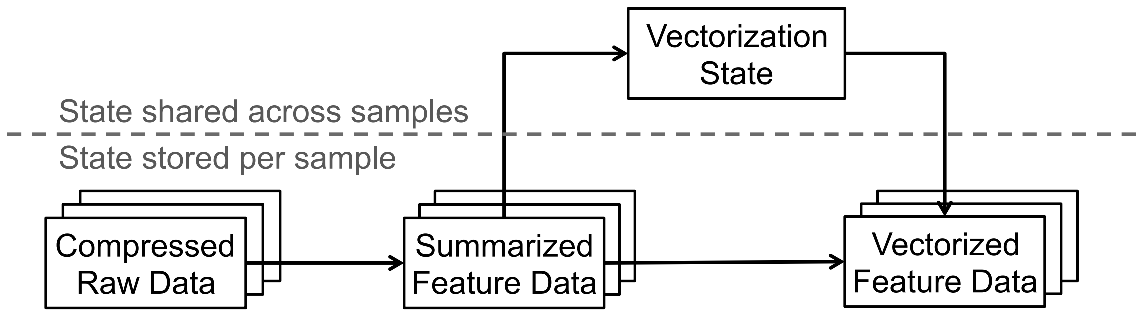


Figure 3.2: Feature extraction leverages parallel computation to improve scalability. Note that stacked rectangles indicate data processed or generated in parallel, and all computation performed on a per-instance basis occurs in parallel. Summarization reduces the size of data by discarding unessential data, and vectorization state guides the process of generating feature vectors from summarized feature data.

tegrate the progression of time, we now discuss the implications of our design for dataset requirements. Datasets traditionally treat samples and labels as pairs, associating a single static label with each sample. Our design requires a more expressive approach to labeling, associating each sample with a series of timestamped labeling events. We refer to the final label for a sample as the *gold label* since the final label represents the most mature label knowledge for a sample. We implement temporal sample consistency by replacing temporally consistent sample labels with the gold labels. We implement temporal label consistency by including current labeling knowledge for all historical samples in each epoch.

Creation of a dataset reflecting accurate label knowledge over time requires disciplined collection from labeling sources at regular, frequent intervals to ensure up-to-date labelings. Alternately, retrospective collection of labels over time may be possible using outdated signatures, but online components of detection systems may prevent retrospective collection. If historical labelings are not available at regular intervals, any available historical labels may *safely approximate* temporally consistent labels back to the time at which the first historical labeling is available. Outdated labels represent a safe approximation of temporally consistent labels since the outdated label was available at the time of training. To compensate for outdated labels, the gold label may replace a historical label once a sample has reached a fixed age.

Separate from labeling concerns, the dataset should also capture the prevalence of samples over time. Chapter 5 demonstrates that sample prevalence can vary greatly, potentially informing efforts to prioritize analysis as well as indicating the impact of potential threats. Datasets which indicate prevalence of samples over time allow reproduction of samples within and across epochs to represent the effects of polymorphism.

3.3 Scalability

We design our platform to scale to large amounts of data to both facilitate deployment and improve detection performance. Given the prior and active work in the area of scalable and distributed learning algorithms [52, 117], we focus on scalability of feature extraction. We achieve scalable feature extraction through a combination of parallel computation, compression and caching, and discuss how each applies to our feature extraction process. Figure 3.2 presents an overview of our feature extraction process.

Parallel processing across samples expedites feature extraction insofar as the vectorization of an individual sample happens *in isolation* from all other samples. Unfortunately, common vectorization workflows introduce dependencies on the set of samples as a whole. For example, the bag-of-words feature extraction technique represents strings as vectors of token frequency, effectively representing the string "`the girl and the ball`" as `{and:1, ball:1, girl:1, the:2}` [102]. Although bag-of-words can determine a token count for each string in isolation, vectorization requires assigning a unique index for each token observed in the dataset as a whole.

We refer to any state which is necessary for vectorization and depends on multiple samples as *vectorization state*. Given the vectorization state each sample can be vectorized in isolation, allowing parallelization of feature extraction. Implemented naively, feature extraction requires processing each sample twice; the first round of processing collects the vectorization state, and the second round generates the actual feature vectors.

To expedite the feature extraction process, we introduce a caching step eliminating the need to process all raw data twice. The caching step, referred to as *summarization*, removes extraneous data from the raw record which is unnecessary for subsequent feature extraction. For example, bag-of-words requires only a dictionary of word counts, which is often more compact than the original text. Similarly, a feature examining the length of URL requests needs only the length of each URL requested, which is much smaller than the original URLs themselves. The benefit of summarization increases with the incremental introduction of each new batch of training data as updates to feature vectors and vectorization state depend only on summarized data rather than the larger raw data.

We present an example illustrating our feature extraction process in Figure 3.3. Each raw sample corresponds to a list of URLs, and the feature vectors represent the domains contained in the list of URLs. Since the path and arguments in a URL are often longer than the domain, the summarization process retains only the domain from each URL. Notice that Epoch 1 neither retains nor requires the raw samples that appeared in Epoch 0; instead, Epoch 0 produces sample summaries which persist into Epoch 1. Once all sample summaries from present and past epochs are available, the vectorization state associates each domain with a feature index. Lastly, the vectorization process operates on the vectorization state and sample summaries to produce the feature vectors.

We design the feature extraction process to support flexible vectorization of diverse raw

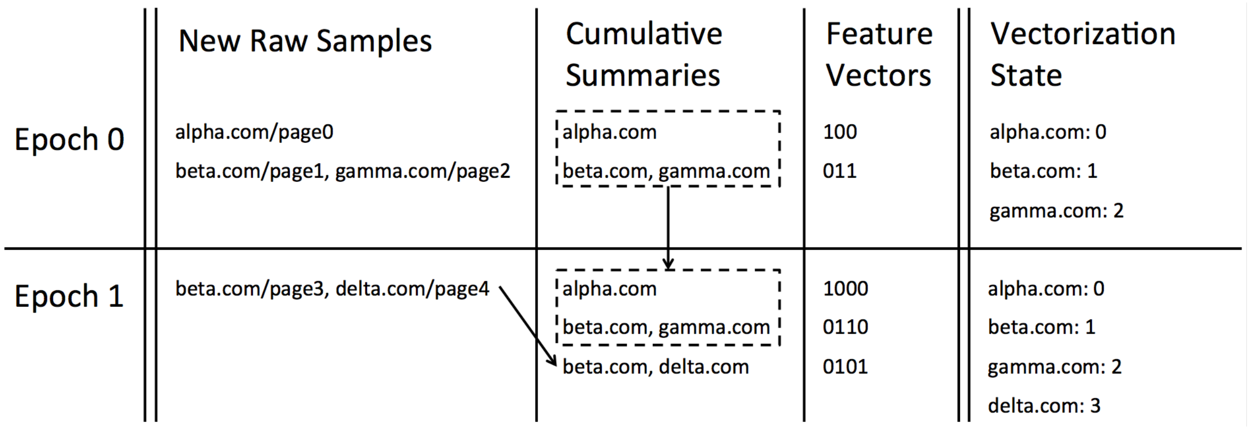


Figure 3.3: As raw samples appear the summarization process extracts the data from each sample necessary for feature extraction. Only summarized data persists across successive epochs, allowing deletion of larger raw data. The vectorization process derives vectorization state from summarized data and subsequently generates a feature vector corresponding to each summarized sample.

data. The vectorization function generating feature vectors given summarized samples and vectorization state is an arbitrary computation, and the vectorization state is similarly unconstrained. As motivation for the flexibility of our design, consider vectorizing a feature describing the length of each list of URL requests. Although the raw data supporting the number of URL requests is the same as in Figure 3.3, the feature extraction process is different. The vectorization process could assign each list length to a unique index in the feature vector, similar to the assignment of domain names in Figure 3.3. Alternately, the vectorization process could use the vectorization state to store discretization boundaries for a binning scheme customized to the training data. Prior work demonstrates advantages to binning schemes with bin boundaries specifically chosen to maximize correlation with data labels [27], which is fully expressible in our feature extraction platform.

Stateless approaches to feature extraction capable of transforming a raw instance directly into a fixed-width feature vector offer an alternative to our feature extraction process. Feature hashing associates strings directly with feature indices using the hash of the string [136]. Similarly, predetermined binning schemes can discretize continuous features independent of other samples. While stateless approaches can simplify design and improve resource utilization, disadvantages discourage adoption. Hashing techniques and predefined binning schemes can adversely effect detection performance. Additionally, hash collisions cause multiple features to map to the same index and consequently reduce the interpretability of the model.

While we design and present the feature extraction process to provide favorable performance in deployment settings which regularly incorporate new data, we can improve

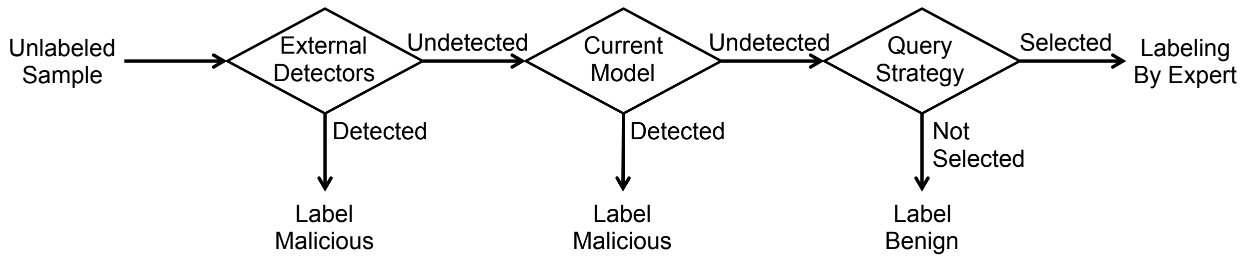


Figure 3.4: External detectors and the current model detect known malware to prevent wasting expert labeling effort. The expert reviews a limited number of samples selected by the query strategy, and any sample not selected for expert review is labeled benign.

performance when analyzing data in retrospect. Subject to constraints, feature extraction may occur once over the entire dataset rather than incrementally as new samples appear. To extract features only once, the feature extraction process must maintain the condition that future samples only impact the association of features to vector indices but never the semantic significance or value of each feature. For example, bag-of-words features correspond to individual word counts, which are constant for each sample independent of other samples in the dataset; the introduction of additional samples impacts only the assignments of word counts to vector indices. Alternately, discretization of continuous values using dynamic bin sizes depends on all samples in the training data, and consequently allows future samples to influence the semantic meaning of feature values. Provided that dependence across samples influences only the association of feature values to vector indices and does not influence feature values themselves or semantic meaning, the future samples will only have the impact of inserting zero-valued indices into feature vectors. For a given sample, any non-zero feature value must correspond to some feature extracted from the sample itself, and hence would have appeared in an incremental feature extraction. In our experiments we constrain feature extraction such that the semantic meaning of the feature vector for a sample exists independent of the other samples.

In addition to the parallel computation and caching enabled by summarization, our design uses lossless compression to improve scalability. We apply lossless compression to all data stored on disk, including raw data, sample summaries and feature vectors. Compression transparently conserves both disk and network resources, and decreases runtime due to the relative latencies of disk reads and decompression computation [17].

3.4 Interaction With Human Supervisor

We now discuss all aspects of our design relating to interaction with the human reviewer. We begin by presenting our process for labeling training data, including the query strategy for selecting samples for human review. We then present our approach to learning and integration of labeling knowledge from the expert reviewer, including learner design to increase interpretability. Although we present specific approaches to labeling and learning, other approaches to labeling and learning are possible within our platform. The approaches we present are application independent and well suited to multiple malicious content domains, but could be replaced by any means of labeling training data and subsequently producing a model.

Since our platform supports training a new model during each epoch, we present a labeling approach which updates labels for all samples during each epoch to include latest label knowledge in model training. Figure 3.4 presents an overview of the labeling process, combining external detectors, the current model and limited expert analysis to label samples. Both the external detectors and the current model function as filters to reduce the burden on the human reviewer by identifying known-malicious samples. Lastly, a query strategy moderates access to the human reviewer using heuristics to select samples whose review will benefit detector performance the most.

We refer to the use of external detectors when determining queries to the human reviewer as the *external detection* heuristic. The objective of external detection is to prevent wasting human review effort on samples that can be reliably labeled using external labeling knowledge, such as public blacklists, other detectors or industry threat exchanges [28, 66]. We label a sample as malicious given consensus among external labeling sources, where the definition of consensus may vary across application domains and external labeling sources. For example, in the malware detection domain we may regard the appearance of a sample in a malware distribution network as sufficient for consensus that the sample is malicious. Alternately, we may not regard detection by a single vendor as sufficient for consensus, and instead require detection by multiple detectors. Since adversaries design malicious content to evade detection and detectors are adverse to mislabeling of benign content across security application domains, we do not definitively label any sample as benign based solely on lack of detection from external sources. Our analysis in Chapter 5 confirms detectors are adverse to falsely identifying samples as malicious.

We refer to the use of the current model when determining queries to the human reviewer as the *internal detection* heuristic. The objective of internal detection is to prevent wasting human review effort on samples that can be reliably labeled using the present detector. We calibrate the model to strongly favor errors that label malicious content as benign over errors that label benign content as malicious. Accordingly, we label as malware any samples the model detects as malicious, but do not label a sample as benign based solely on lack of detection from the current model.

After applying the external detection and internal detection heuristics to identify malicious content, we consider any remaining undetected samples as candidates for expert review. We use a query strategy to select samples for review up to a maximum submission budget B. We determine B dynamically as a fraction of the number of new samples during an epoch to remain responsive as new threats and malicious content campaigns emerge. Our approach labels all samples not selected for review as benign.

The query strategy is responsible for efficient utilization of the human reviewer. Prior work has explored the *uncertainty sampling* query strategy, which selects samples for expert review that are closest to the decision boundary [56, 128, 22]. While we evaluate uncertainty sampling, we also introduce the *maliciousness* query strategy customized for the malicious content detection context. Uncertainty sampling leverages the intuition that samples farthest from the decision boundary represent the most confident classifications, and consequently are likely to receive a label consistent with the present prediction. To increase impact during retraining, uncertainty sampling queries samples closest to the decision boundary. Since we have pre-filtered samples to identify known malicious content, all samples presently in training data are benign by default. In an adaptation of the core intuition of uncertainty sampling, maliciousness therefore queries samples which are ranked as the most malicious since these are the samples where the classifier is least certain of the present label.

While actual domain experts would perform manual labeling in production settings, less expensive alternatives are advantageous in research and experimentation settings. Our platform models the involvement of a labeling expert by appealing to the gold label of a sample, defined as the final label obtained for the sample. Since adversaries may explicitly attempt to mislead domain experts, for example in the phishing context, we do not assume that experts always produce perfect labels. Instead, we assign both a true positive rate and a false positive rate to experts representing the error bias from deceptive samples and conservative labeling practices.

Having discussed our approach to labeling and querying the human reviewer, we now discuss our approach to learning and integration of labels from the human reviewer. Although our platform is compatible with any means of creating a model from labeled training data, we present a specific approach well suited to malicious content detection. We design our approach to maintain model interpretability so that supervisors can understand the impact of features on classification outcomes. We also design our approach to scale well, allow for class imbalance in training data and support trade-offs in classification errors to avoid labeling benign content as malicious.

We begin by examining the mechanics and advantages of logistic regression, which forms the basis of our approach to learning. 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 a feature vector representing an instance and $\mathbf{w} \in \mathbb{R}^d$ is the model, then logistic regression obtains a raw, continuous prediction $f_{\mathbf{w}}(\mathbf{x})$ as:

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

Logistic regression may also apply a bias term to all samples, shifting the predicted value $f_{\mathbf{w}}(\mathbf{x})$ for all $\mathbf{x}$ to bias classification errors. The clear relationship between the feature vector

$\mathbf{x}$ and the raw prediction $f_{\mathbf{w}}(\mathbf{x})$ allows supervisors to clearly understand what the detector is doing and why.

Logistic regression obtains a binary classification result by comparing $f_{\mathbf{w}}(\mathbf{x})$ to a classification threshold T. If $f_{\mathbf{w}}(\mathbf{x}) \geq T$ then logistic regression labels the instance positive, and otherwise labels the instance negative. Adjusting the threshold T creates a tradeoff between true and false positive rates. Considering malicious as the positive class, higher values of T will result in lower false positive rates while missing more malicious content, and lower values of T will result in higher false positive rates while detecting more malicious content. Logistic regression scales well in training with many available implementations capable of accommodating high dimensional feature spaces and large amounts of training data. 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.

Having discussed the structure of the logistic regression classifier and associated transparency and flexibility in balancing labeling errors, we now discuss the training process which we use to produce models. Since the human reviewer only labels a small portion of the training set, we assign a higher weight W during training to reviewer labeled samples. The higher weight W increases the penalty for models which misclassify any points labeled by the reviewer, leading models to favor errors on unreviewed samples over errors on samples that are reviewed by the human. The value chosen for W represents a balance between availability and accuracy of training labels, with high values of W valuing the more correct but limited labels of the human and low values of W valuing the plentiful but noisy labels available through automated means. In practice, values of W which are either too high or too low decrease performance, and the optimal value must be found experimentally from training data.

Chapter 4

Platform Implementation

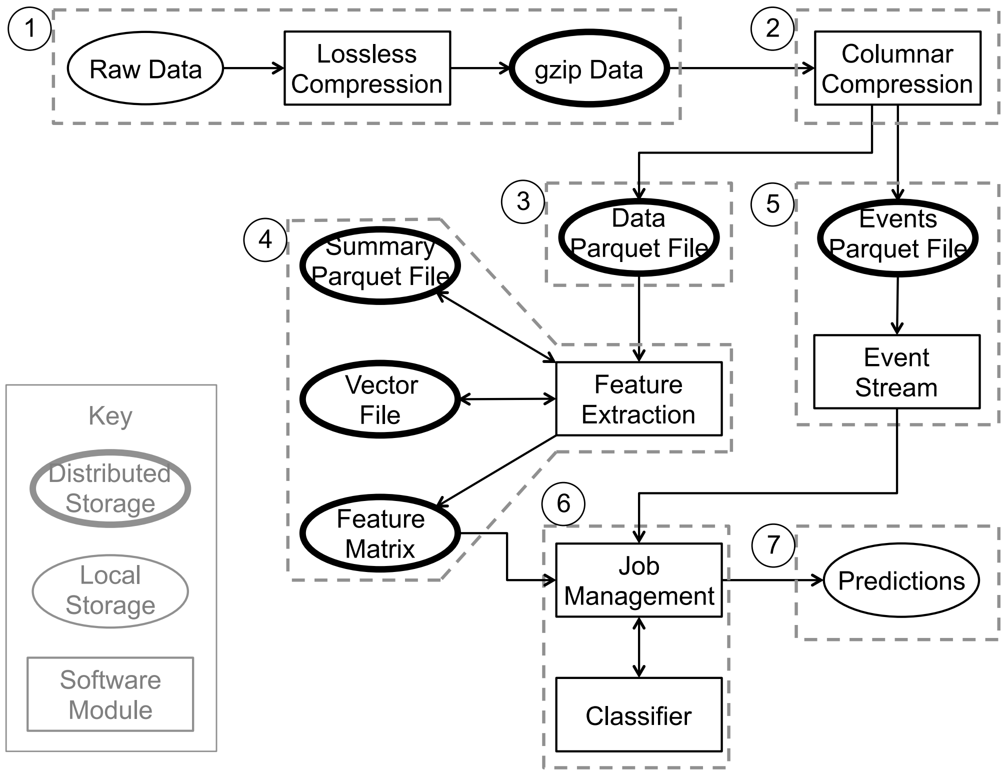
In this chapter we present the implementation of our platform. We implement our platform in approximately four thousand lines of Python,[1] and use NumPy, SciPy and scikit-learn for numerical operations, sparse matrix representation and machine learning respectively [43, 82, 135]. Implementation in Python facilitates integration with IPython, a browser based user interface for interactive data exploration and experimentation [87]. Our implementation leverages Apache Spark for distributed computation and HDFS for distributed storage [115, 147]. The combination of IPython and Spark allows for interactive, large-scale data exploration using distributed computational infrastructure. Interactive data exploration provides critical insight for both system supervisors and researchers to understand data trends and evaluate system performance. We deploy our implementation on a cluster managed by VMWare vSphere [134], and use approximately 44 cores and 600GB of RAM to process 778GB of raw data.

We structure our presentation of the platform implementation as follows: In Section 4.1 we discuss the structure of our implementation source code and associated design choices, Section 4.3 discusses our integration with IPython, and Section 4.4 discusses the configuration of hardware resources within our cluster.

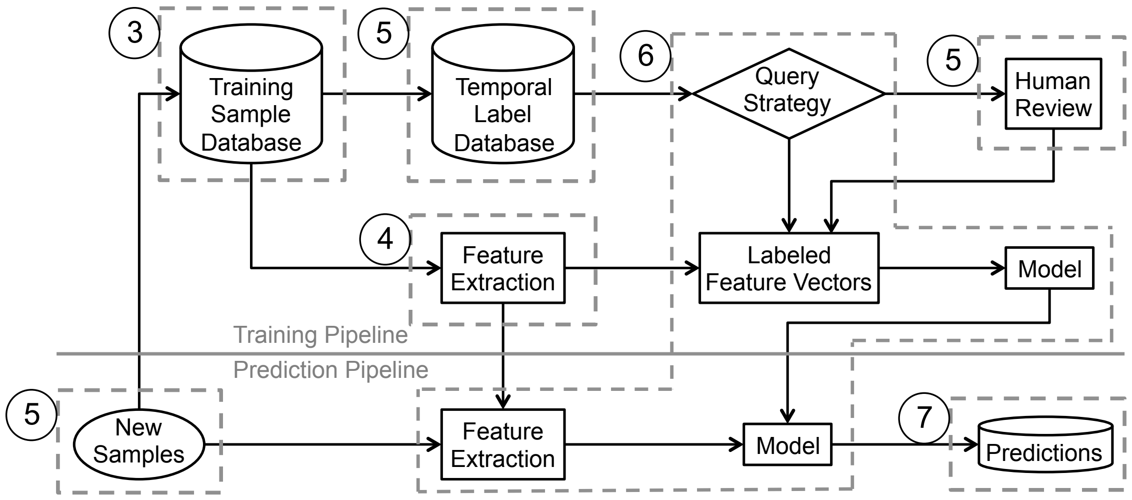
4.1 Implementation Modules

In this section we discuss the structure and details of our platform implementation. We present the division of functionality between modules, design choices within modules, and flow of data between both modules and persistent storage. Figure 4.1a presents an overview of the implementation, and Figure 4.1b presents the correspondence between the platform design and implementation. Since the platform design in Chapter 3 reflects the arrival of data

[1]As of the filling of this thesis, our code and data are available online at `http://secml.cs.berkeley.edu/detection_platform/`



(a) Platform implementation overview.



(b) Correspondence between platform design and implementation.

Figure 4.1: We design our platform implementation to operate on historical data. Numbered circles indicate the correspondence between design and implementation. The simulation of external inputs from historical data drives several modifications to the platform design. For example, the Training Sample Database initially holds all samples, and New Samples indicates which samples from the database appear in each epoch.

in real-time and our implementation operates on historical data, we introduce corresponding design modifications while maintaining an accurate representation of detection results.

We begin with an overview of the interactions between modules. Prior to analysis, the Lossless Compression module compresses all data to reduce storage burden and expedite access. Next, the Columnar Compression module transfers raw data into Apache Parquet, a distributed storage format that uses compressed columnar storage to improve database query performance [29]. The Columnar Compression module generates the Data Parquet File to store sample feature data and the Events Parquet File to store labeling knowledge over time, decoupling sample feature from temporal labeling knowledge. The Feature Extraction module extracts the necessary data for feature vectorization from the Data Parquet File into the Summary Parquet File, and ultimately produces the Feature Matrix which includes feature vectors from all samples. The Data Parquet File, Summary Parquet File, Vector File and Feature Matrix serve as caches to reduce repeated computation throughout the feature extraction process and across repeated experiments.

Separate from the feature extraction process, the Event Stream module interacts with the Events Parquet File to model the sequential appearance of samples over time. Each appearance of a sample constitutes an *event*, and includes a label that is temporally consistent with the position of the sample in the sample sequence, and a *gold label* corresponding to the best label in retrospect. The Event Stream partitions the sequence of events into epochs. The Job Management module uses the epochs from the Event Stream to define a sequence of *jobs*, each specified by a set of training and evaluation samples with gold labels for all samples and temporally consistent labels for training samples only. The Classifier modules execute jobs, including querying the human reviewer to improve temporally consistent labels, training a model and predicting labels for evaluation data. The Job Management module invokes Classifier modules to manage job execution, and the Predictions module stores all detection results.

We now discuss the correspondence between platform design and implementation. The platform design in Chapter 3 cycles through feature extraction, labeling, model training and prediction of labels for new samples, and then repeats with the new samples included in training data. Our implementation realizes the platform design in accordance with operating on historical rather than real-time data. The Data Parquet File implements the Training Sample Database, and initially holds all samples rather than receiving incremental updates as new samples appear. Since all samples are initially available, we extract a single Feature Matrix to avoid the unnecessary computation of repeated feature extraction. The Event Stream module implements the New Samples, Temporal Label Database and Human Review modules by defining ordered epochs consisting of sets of training and evaluation data. Epochs include temporal labels for training samples and gold labels for all samples to both model the human reviewer and evaluate prediction results. Each epoch corresponds to a distinct learning and prediction job.

The Job Management module instantiates Classifier modules to process each job, implementing the actual interaction with the human reviewer, model training, prediction and persistence of state across epochs. The Job Management module selects rows of the feature matrix corresponding to training and evaluation data and associates labels with each row, implementing the Prediction Pipeline Feature Extraction and Labeled Feature Vectors modules. The Job Management module associates both a temporally consistent and gold label with each training vector, allowing the Classifier module to implement and execute the Query Strategy to finalize training labels. The Classifier modules then produce Model objects, predict labels for training data and store the results in the Predictions module for evaluation. The Job Management module supports the Classifier module by persisting Model objects and the results of previous oracle queries across jobs.

We discuss each component of our implementation in greater detail below. Our discussion follows the seven aggregations of components shown in Figure 4.1a.

(1) Lossless Compression

We begin by applying lossless compression exactly preserving the original raw data and subsequently deleting uncompressed raw data to conserve storage resources. Compression can simultaneously improve disk and network utilization while decreasing runtime due to the relative latencies of disk reads and decompression computation [17]. For our implementation, we retain compressed raw data primarily for archival purposes since the Data Parquet File and Events Parquet File will be the primary data sources for other modules.

While any compression technique improves storage utilization, improving read latency requires awareness of decompression parallelization constraints. Many common compression formats, including gzip, LZ4 and Snappy, must decompress serially thereby preventing parallel retrieval of raw data from disk. Compression formats bzip2 and a modified form of LZO support parallel decompression, but can require additional decompression latency or have reduced effectiveness relative to gzip [137]. Since our platform operates on individual samples which may be stored in isolation, we aggregate samples into multiple files which we compress and store with gzip [26]. We select gzip compression for storage of raw data due to effectiveness of compression, broad support and relatively rare need to re-access files [20]. Although each file decompresses serially, we parallelize decompression across individual files to rapidly retrieve data from disk.

(2) Columnar Compression

Although lossless compression of raw data improves read latency, compression across entire samples hinders common database operations. For example, consider samples consisting of multiple fields and a database query selecting all samples where a specific field falls within a specified range. Compression across entire samples forces loading and decompression of all data to access the single relevant field. As an alternative, *columnar storage* formats support retrieval of individual fields without loading the entire dataset.

To improve database query performance we transfer raw data to Apache Parquet, a columnar storage implementation accessible from Spark. Parquet supports nested data, allowing flexible combinations of common data structures such as dictionaries and lists. We use LZO compression with Parquet to maintain efficient storage and access, although alternative compression algorithms are available.

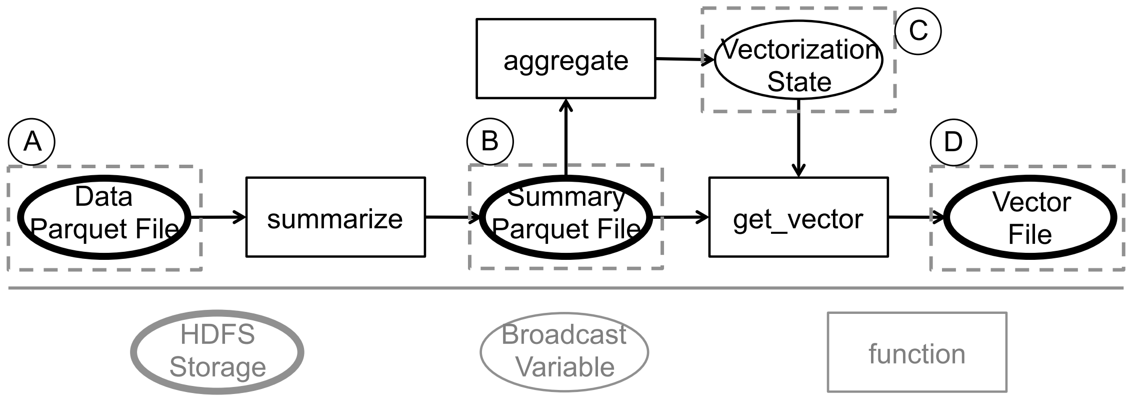
We strive to preserve all information in raw data, but the transfer of raw data into Parquet is not lossless. Since Parquet maintains a schema for data, a naive transformation creates a column for each field in the raw data. Consequently, raw data with a wide range of key values can result in large schemas and poor performance. For example, our case study data includes a dictionary representing library imports where the keys corresponds to a library names and values correspond to functions imported from the library. Since the keys are arbitrary strings, the number of Parquet columns can be arbitrarily large. To limit the size of the Parquet schema, we transform any dictionary with a large number of keys to a list of `(key, value)` pairs, where the `value` is a serialization of the original dictionary value. Transforming the original dictionary to a list retains all information while limiting the size of the Parquet schema.

Additional data modifications are necessary if dictionary keys contain characters which Spark does not support in conjunction with Parquet. To maximize both data retention and explainability, we adopt a two-pass solution. First, we remove any of the following characters that do not cause collisions between different keys: `-_.)(@?,‘><` and any whitespace. Then, we replace any key which still contains non-alphanumeric characters with a hash of the key. Although hashing the key reduces explainability, normalizing the key could cause collisions between semantically different key values (e.g. by mapping all non-printable characters to null).

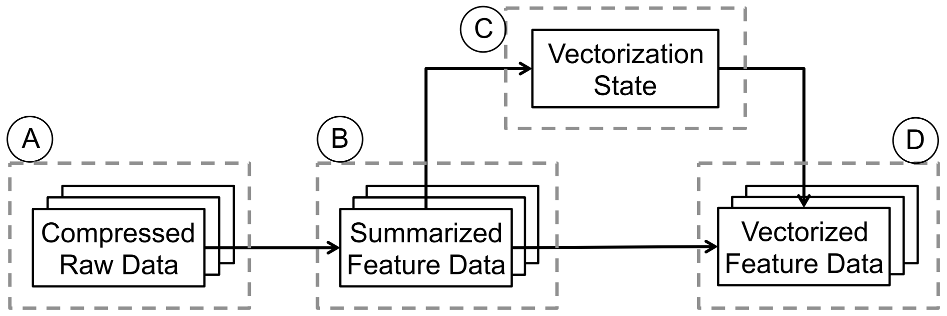
The Columnar Compression module produces both the Data Parquet File and the Events Parquet File. The Data Parquet File contains all raw data related to feature information, and the Events Parquet File contains all raw data related to labeling. Creation of separate files for feature and labeling information improves performance and simplifies implementation since subsequent design steps separate feature extraction and labeling. We retain the full extent of both feature and labeling information to allow modifications to feature extraction and labeling techniques without regeneration of the Parquet files.

③ Data Parquet File

As Figure 4.1b indicates, the Data Parquet File corresponds to the Training Sample Database in the detector platform. If the detection platform were operating in real-time, the Training Sample Database would receive periodic updates to hold all samples seen to-date. Since we implement an evaluation of the platform using historical data, we initially populate the Data Parquet File with data from all samples to avoid the unnecessary overhead of periodic database updates. Spark implements a SQL interface



(a) Feature extraction implementation overview.



(b) Correspondence between feature extraction design and implementation.

Figure 4.2: The feature extraction implementation conforms closely to the feature extraction design, with circled letters indicating points of correspondence. Note that Spark Broadcast Variables offer distributed access, but reside in memory rather than persistent storage.

for data in the Parquet format, allowing the Data Parquet File to present the same interface as the Training Sample Database to other modules.

The Data Parquet File represents the first point of caching in the detection platform implementation. After the Columnar Compression process creates the Data Parquet File and Events Parquet File, subsequent modules never refer back to the original raw data because the data and events parquet files contain information equivalent to the entire raw dataset. Consequently, only a change in the raw dataset can cause an update to the Data Parquet File.

4 Feature Extraction

Although the specifics of feature extraction are application specific, the platform design enforces a general implementation framework for feature extraction. We design the feature extraction framework to maintain scalability, explainability and flexibility across a broad range of feature extraction applications. We adopt an object oriented design and separate features representing distinct data into distinct classes. For example, a malware execution trace may involve both disk and network activity, our platform allows clean separation of feature extraction code for each. Treating each separate type of feature as a class facilitates observation of aggregate properties about the feature class, such as the distribution of feature values.

Our feature extraction implementation follows the feature extraction design from Chapter 3. Figure 4.2 presents an implementation overview as well as correspondence to the feature extraction design. Each feature extraction class must implement the `summarize`, `aggregate` and `get_vector` functions: `summarize` discards unnecessary raw data, `aggregate` processes summarized data to produce any state necessary for vectorization, and `get_vector` produces the actual feature vectors. We discuss each function in greater depth below.

`summarize` The `summarize` function implements summarization, the process of discarding any raw data unnecessary for feature extraction. `summarize` accepts a single sample from the Data Parquet File as an argument and returns arbitrary data for storage in the Summary Parquet File. Since Parquet supports nested structures, the summary data may contain arbitrary combinations of lists, dictionaries and Python primitives. For example, for bag-of-words feature extraction `summarize` would transform the string “`the girl and the ball`” to the dictionary `{and:1, ball:1, girl:1, the:2}`. Since each feature extractor object corresponds to a unique column in the Summary Parquet File, the columnar format of Parquet allows efficient retrieval of `summarize` results for an individual feature class across all samples.

`aggregate` After `summarize` extracts summaries from each sample, `aggregate` operates over all summary data for the feature extraction class to compute any vectorization state necessary for feature vectorization. Since the results of `summarize` are in distributed Parquet storage, `aggregate` can use distributed computation to remain scalable as data increases. The vectorization state output from `aggregate` is an arbitrary Python object, and may contain any data structures helpful for transforming summary data into feature vectors. In the case of bag-of-words feature extraction, `aggregate` may contain a dictionary mapping unique words to indices in the feature vector. Alternately, to discretize a continuous value, `aggregate` may create a dictionary mapping continuous feature value ranges to different indices. After creation of any vectorization state, `aggregate` stores the vectorization state in a Spark Broadcast Variable, allowing for efficient distribution and access across the cluster during feature vectorization.

`get_vector` After `aggregate` computes any vectorization state, `get_vector` generates the actual feature vector from the sample summary data and vectorization state. For example, in the case of bag-of-words feature extraction, `get_vector` generates a sparse vector by setting word count values at the indices indicated by the vectorization state. After each feature extraction object generates a corresponding feature vector for an instance, we concatenate the individual feature vectors to form the entire feature vector for the instance.

In addition to the `summarize`, `aggregate` and `get_vector` functions, each feature extraction class also implements the `get_interpretation` and `get_name` helper functions. The `get_interpretation` function returns a dictionary mapping each feature index to a string interpretation of the meaning of the feature. For example, in bag-of-words feature extraction the string may contain the word corresponding to the index. Forcing each feature extraction class to provide string interpretations of features helps maintain model explainability and allows the learner to inform the human reviewer. The `get_name` function returns a string that uniquely identifies the feature extraction class. The output of `get_name` serves as the column name for the feature extraction class in the Summary Parquet File, allowing efficient retrieval of all sample summaries for the feature extraction class. By default, `get_name` inspects the feature extraction object and returns the name of the instantiated class.

The object oriented design of feature extractors facilitates code reuse across feature extraction tasks. For example, consider binary vectorization indicating which of a set of events occur within raw attribute data. Whether one event constitutes contacting an IP subnet, importing a library or accessing a path on disk, `aggregate` must assign each event to a distinct index and `get_vector` must set the appropriate indices in the feature vector corresponding to the sample summary. Only the `summarize` function needs differ across classes since the process of extracting IP subnets, library imports or disk accesses from raw data is unique for each type of feature.

The detection platform design in Chapter 3 includes repeated feature extraction for real-time systems to incorporate new training samples. However, Chapter 3 also discusses a feature extraction approach for analysis of historical data allowing generation of a single feature matrix inclusive of all samples while maintaining temporal consistency. In summary, generation of a single feature matrix requires that the feature extraction process maintain the property that the introduction of future samples may introduce additional dimensions in the feature vector but may not change feature values. Additionally, for a given sample the feature value of any dimension caused by the appearance of a future sample must be zero. For example, bag-of-words feature extraction satisfies the properties: the non-zero dimensions of the feature vector for a sample are a function of the words in the sample alone, and will not change with the introduction of new samples. Additionally, any introduction of new samples will only result in additional zero-valued dimensions in the feature vector. After `get_vector` vectorizes each sample we store the feature vectors in HDFS as Vector File, creating a distributed feature matrix. To leverage the breadth of learning implementations available for single machines, we also generate and store a single SciPy sparse matrix in HDFS as Feature Matrix.

5 **Event Parquet File and Event Stream**

The Event Parquet File holds all raw data related to sample prevalence and label knowledge over time, and the Event Stream module supplies sample prevalence and label knowledge to the detection platform. The Event Stream module produces a sequence of *event tuples* of the form `(sample, temporally consistent label, gold label)`, where each tuple corresponds to a sample that will pass through the detection platform. Recall that the temporally consistent label represents the best available label at the time the tuple appears in the event tuple sequence, and the gold label represents the best sample label in retrospect. The Event Stream module synthesizes temporally consistent labels and gold labels from raw data in the Event Parquet File to allow flexible modifications to labeling policy. For example, if the raw data contains labels from multiple sources, assignment of labels in the Event Stream allows varying the policy for harmonizing labels from disparate sources.

The Event Stream module partitions the sequence of event tuples into epochs, which are well suited to model a range of flexible phenomenon for evaluation. For example, the same sample may repeat in multiple event tuples to reflect sample prevalence. Sample repetitions may occur within a single epoch or across epochs to represent changes in sample prevalence over time. When a sample repeats, the gold label remains the same across all sample appearances but the temporally consistent label may change, consistent with changes in labeling knowledge over time. In the interest of comparison with alternate evaluation methodologies, epochs can also capture temporally inconsistent evaluations by ordering the sequence of event tuples regardless of actual chronology.

By specifying sample timing and label knowledge, the Event Stream module implements the external input components in the platform design. The partitioning of event tuples into epochs specifies the relative ordering and prevalence of samples, implementing the New Samples module. Similarly, the assignment of a temporally consistent label to each event tuple implements the Temporal Label Database. Lastly, the assignment of a gold label to each even tuple simulates the Human Review by providing the best available label for the sample.

6 **Job Management and Classifier**

The Job Management and Classifier modules are responsible for learning and prediction within the detection platform, including feature selection. The Job Management module interacts with the Event Stream to construct a sequence of jobs, each containing the following elements:

- Rows of the Feature Matrix for use as training data
- Rows of the Feature Matrix for use as evaluation data
- Temporally consistent labels for training data
- Gold labels for training data
- Gold labels for evaluation data

For job n, the samples appearing in the first n epochs serve as training data, and the samples in epoch $n + 1$ serve as evaluation data. Consequently, the number of jobs is one fewer than the number of epochs. We include the gold labels for training data to simulate Human Review in the platform design. The Job Management module instantiates Classifier modules to execute each job, which includes applying the Query Strategy, producing the Model and outputting Predictions for the evaluation data, as described in the platform design.

We design the learning and prediction process to maximally use parallelization. Since we specify each job as a set of training and evaluation row indices and labels, and share the same Feature Matrix across all jobs, the amount of job specific data remains relatively small. Consequently, we are able to efficiently distribute jobs for execution in parallel while distributing the larger Feature Matrix only once. Parallel execution of jobs expedites experimentation with different learning parameters and techniques at large scale. Unfortunately, parallelization across jobs requires defining each job in isolation, preventing any use of previous results during job execution.

Query strategies that use the results of previous models to select samples for human review introduce dependencies across jobs. For example, the uncertainty sampling query strategy selects samples which are close to the decision boundary of the previous model, introducing a dependency between models that inherently serializes job execution. To support dependencies across jobs, the Job Management module allows each Classifier to produce state which will be given to the next Classifier. Although the passage of state decreases the isolation of each Classifier, we maintain each Classifier as an individual object for each epoch to facilitate inspection and comparison of Classifier objects across time.

Having described the role and services of the Job Management module and the structure of each job, we now discuss the Classifier module. The Classifier module presents an interface for learning and prediction on labeled training and evaluation matrices to the remainder of the platform. Abstracting implementation details allows the actual learning implementation to remain flexible. Our Classifier implementation uses the scikit-learn package [82], but could also use a distributed learning implementation such as MLlib to scale beyond the constraints of a single machine [52, 117]. We also include any feature selection within the Classifier module since feature selection technique may depend on the learning algorithm.

In addition to learning and feature selection, the Classifier module also implements the the Query Strategy, which controls submission to the human reviewer. We implement the Query Strategy within the Classifier module to allow for dependence on the specific learning algorithm. The Classifier satisfies queries using the gold labels provided for training data, and adds noise as necessary to simulate errors in human review. Note that implementing the Query Strategy requires support from the Job Management module to maintain the set of both prior queries and labeling errors across jobs.

7 Predictions

The Predictions module stores output from the Classifier module locally on an individual host. We store predictions locally to facilitate access by other processes running on the same host, but could also store predictions in the same distributed storage as all other data.

While the seven aggregations of components that we present in Figure 4.2a form the bulk of our platform implementation, there are several additional modules which we present below:

Driver

The Driver module serves as the execution entry point for the Python interpreter. The Driver accepts, parses and applies arguments from the user, instantiates all other modules and directs the execution of the detection platform. To invoke the detection platform, the user submits the Python source file containing the Driver module to the Spark cluster.

Common

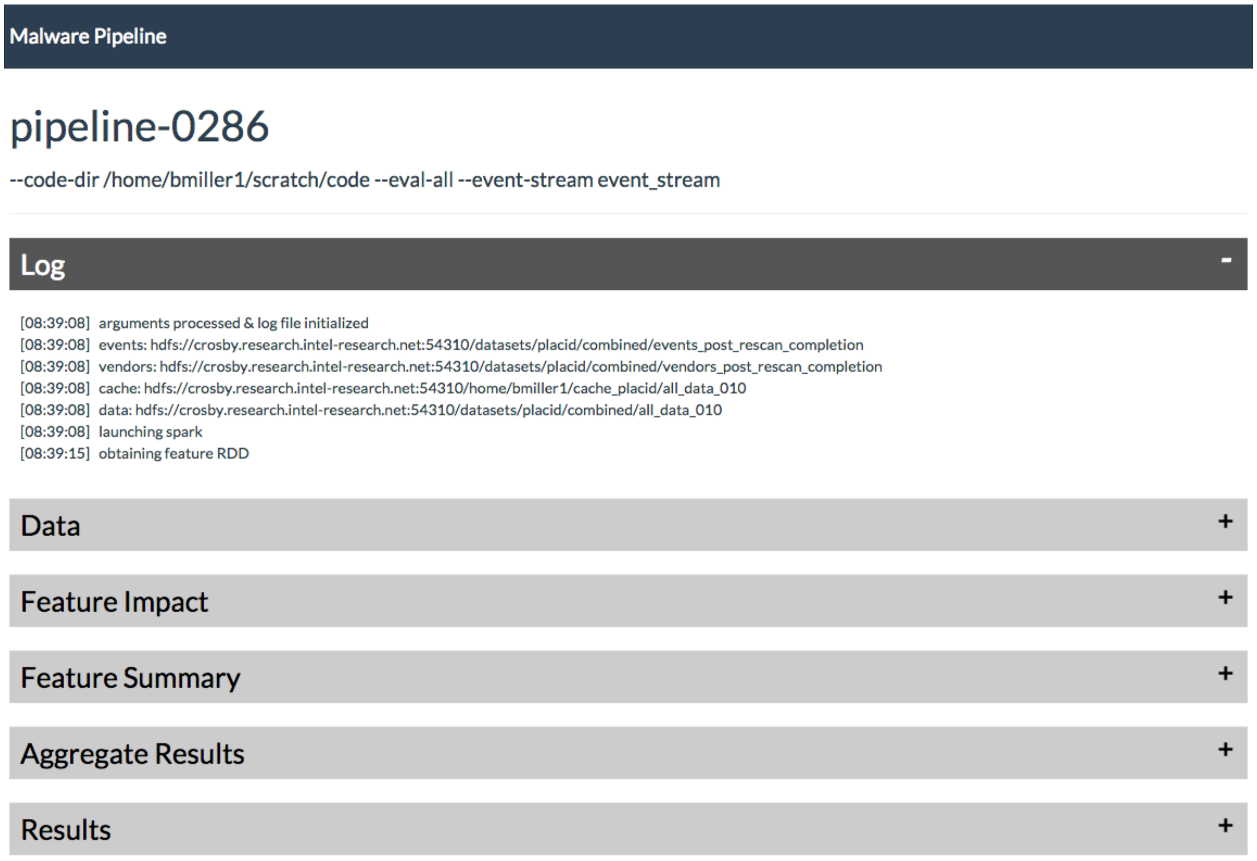
The Common module holds constants and helper functions for use in multiple other modules, such as logging infrastructure and wrappers for file system access.

Resume Driver

Although the majority of platform modules contribute to the production of the Feature Matrix and specification of rows and labels for jobs, the execution of successive learning and prediction jobs consumes the bulk of computational effort. Since the feature matrix and job specifications concisely define the learning jobs which are the bulk of computational effort, we implement functionality to export the Feature Matrix and job specifications from the detection platform for resumed execution on separate hardware. The Driver and Job Management modules implement support for exporting the Feature Matrix and job specifications, and the Resume Driver module resumes execution from the exported state. The Resume Driver module instantiates the Job Management and Classifier modules similarly to the Driver module. Since the specification of the Feature Matrix and job specifications happens before any feature selection, learning or queries for human review, the Resume Driver module is able to support experimentation with these aspects of the learning algorithm as well as human reviewer availability and accuracy.

4.2 Web Interface

The complexity of both the detection platform implementation and analysis results motivates the creation of an exclusive mechanism for monitoring platform execution. We design and



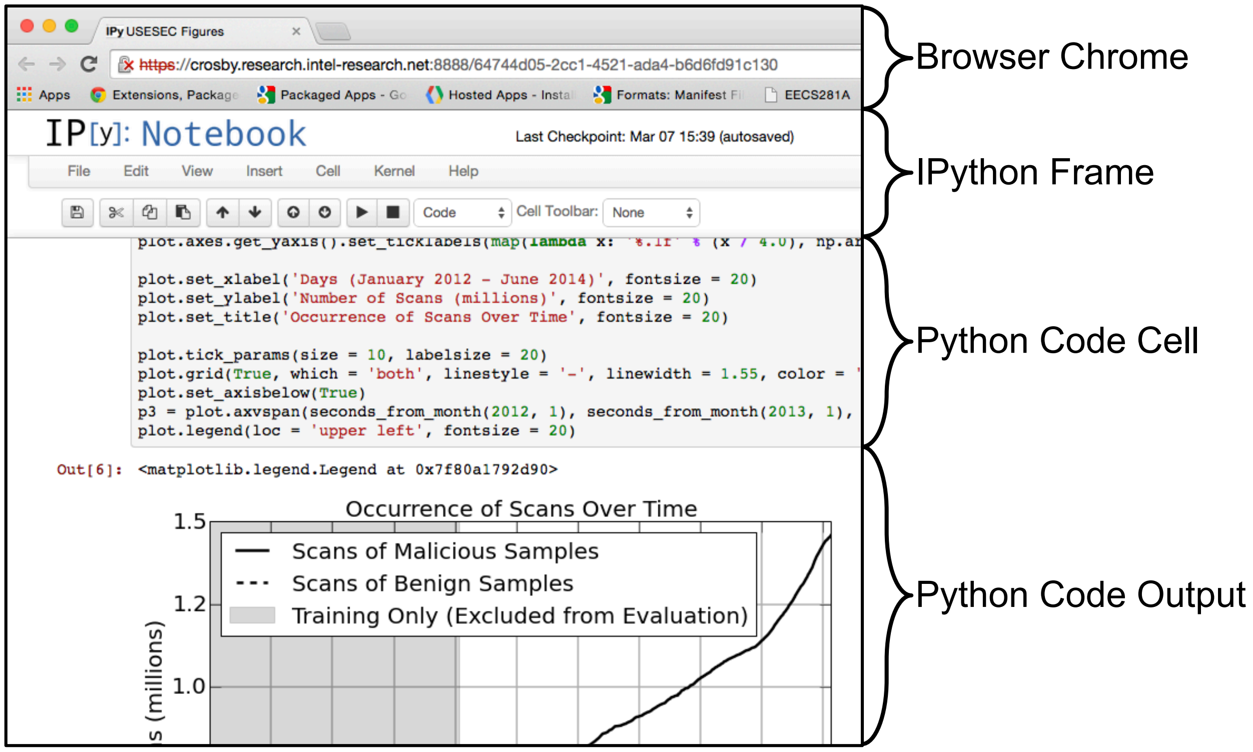
```
--code-dir /home/bmiller1/scratch/code --eval-all --event-stream event_stream
```

```
[08:39:08]  arguments processed & log file initialized
[08:39:08]  events: hdfs://crosby.research.intel-research.net:54310/datasets/placid/combined/events_post_rescan_completion
[08:39:08]  vendors: hdfs://crosby.research.intel-research.net:54310/datasets/placid/combined/vendors_post_rescan_completion
[08:39:08]  cache: hdfs://crosby.research.intel-research.net:54310/home/bmiller1/cache_placid/all_data_010
[08:39:08]  data: hdfs://crosby.research.intel-research.net:54310/datasets/placid/combined/all_data_010
[08:39:08]  launching spark
[08:39:15]  obtaining feature RDD
```

Figure 4.3: The detection platform provides a web interface for users to monitor analysis progression and results.

implement a web user interface to expose platform progress and results to the user. We implement the interface using Flask, a framework for web development within Python [95]. Since Yiu provides a full discussion of the implementation and functionality of the web interface [146], we confine our discussion to the portions of web interface design relevant for detection platform implementation. Figure 4.3 presents a screenshot of the web interface.

The web interface runs as a service within the cluster, separate from any invocation of the Driver module, and displays results for any current or previous invocation of the detection platform. Running the web interface separately from the Driver module allows the interface to remain responsive regardless of load on the Driver module but also complicates communication. We communicate from the Driver module to the web interface via an output directory for each invocation of the detection platform, where the contents of the output directory populate the web interface. The web interface updates regularly, posting updates in real-time for ongoing monitoring of platform execution. The Common module contains logging infrastructure which allows each module to append to a log file that the web interface displays. Similarly, we store the detection predictions in the output directory to facilitate



```
plot.axes.get_yaxis().set_ticklabels(map(lambda x: '%.1f' % (x / 4.0), np.ar

plot.set_xlabel('Days (January 2012 - June 2014)', fontsize = 20)
plot.set_ylabel('Number of Scans (millions)', fontsize = 20)
plot.set_title('Occurrence of Scans Over Time', fontsize = 20)

plot.tick_params(size = 10, labelsize = 20)
plot.grid(True, which = 'both', linestyle = '-', linewidth = 1.55, color = '
plot.set_axisbelow(True)
p3 = plot.axvspan(seconds_from_month(2012, 1), seconds_from_month(2013, 1),
plot.legend(loc = 'upper left', fontsize = 20)
```

```
Out[6]: <matplotlib.legend.Legend at 0x7f80a1792d90>
```

Figure 4.4: IPython facilitates interaction with the Python interpreter, allowing users to conduct interactive data exploration at-scale using distributed computational infrastructure.

display over the web interface.

Although all platform invocations perform logging and store results, we perform other types of analysis for the web interface on-demand only due to computational cost. The Feature Impact and Feature Summary components of the web interface require additional storage and computation. We implement command line options in the Driver module to only compute Feature Impact and Feature Summary at the explicit request of the user.

4.3 IPython Integration

Data exploration and analysis often consists of a series of computations with each successive computation utilizing previous results. Proceeding iteratively, analysis results can prompt further queries that depend on the same intermediate computations as previous results. Python scripts aid analysis by queueing successive computations and enabling reproduction of results during subsequent execution. Unfortunately, iterative execution of scripts to explore a dataset often results in repeated and unnecessary intermediate computations, wasting resources and unnecessarily delaying results.

IPython offers an alternative to repeated execution of batch analysis scripts. IPython

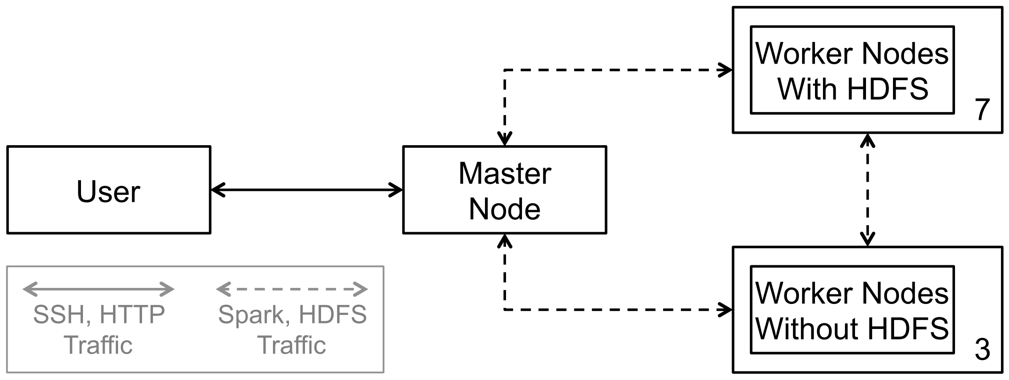


Figure 4.5: Users interact with the Master Node, which manages all cluster services and interacts with the Worker Nodes on behalf of users. Our cluster contains seven worker nodes with HDFS and three worker nodes without HDFS. Restricting storage infrastructure to a subset of nodes facilitates addition and removal of Spark worker nodes without disrupting data storage.

presents a web interface that allows users to enter Python code in *cells* and observe execution results. IPython receives user code via the web interface, submits the code to a Python interpreter, and then displays the output to the user. Output may be either text or graphical, with support from matplotlib [40]. Figure 4.4 presents an example of the IPython user interface with graphical output. Since users may generate or execute new cells in response to the output of previous cells, interactive data exploration may proceed without repeated intermediate computation. Additionally, IPython integrates with Spark to allow for interactive exploration of large-scale data with support from cluster computing resources.

We implement our platform in Python to facilitate interactive exploration of data within the context of our design via integration with IPython. Since each module of our platform can be imported as a Python library, analysts may interactively experiment with underlying data or alterations to individual portions of the design.

4.4 Cluster Architecture

We deploy our detection platform implementation on a cluster managed with VMWare vSphere [134]. Operating on a 778GB dataset, our cluster includes 44 cores with 600GB of RAM. Figure 4.5 presents an overview of our cluster architecture. We divide our resources to form a single *master* and multiple *worker* nodes with resources and services as shown below. Note that *reserved* indicates VMWare guarantees the indicated resource amount will be available to the guest VM, rather than multiplexed among multiple VMs.

Master Configuration (1 node)

- Services: Spark, HDFS, IPython, Platform Web Interface

- 4 Cores (7.7GHz reserved)
- 160GB RAM (160GB reserved)
- 16GB HDD for swap space
- 16GB HDD for root volume
- 50GB HDD for HDFS
- 32GB HDD for Spark spill space
- 100GB HDD for user home directories
- 100GB HDD for `/tmp`

Worker Configuration (10 nodes)

- Services: Spark, HDFS (7 storage nodes only)
- 4 Cores (7.7GHz reserved)
- 48GB RAM (48GB reserved)
- 32GB HDD for swap space
- 16GB HDD for root volume
- 32GB HDD for Spark spill space
- 100GB HDD for HDFS (7 storage nodes only)

Note that only seven of the worker nodes include HDFS storage. Concentrating storage on a subset of worker nodes allows for addition and removal of processing resources without disrupting storage infrastructure. Since our platform design uses both caching and compression, Spark is able to store intermediate results in-memory and minimize the penalty for concentrating persistent disk storage on a subset of nodes.

The memory requirements of the learning process dominate other computational demands influencing the division of hardware resources among nodes. Although the compressed and serialized feature matrix for our dataset requires only 1.7GB on disk, the uncompressed serialized feature matrix requires 11GB of disk space. The uncompressed matrix provides a lower bound estimate of the size of the feature matrix in memory. The learning process duplicates the feature matrix multiple times in memory: first to select rows for a training matrix, then to conduct feature selection, and lastly to select rows and features for an evaluation matrix. Although the learning process duplicates the feature matrix multiple times, only the final training and evaluation matrices need to remain in working memory and intermediate matrices can be swapped out to disk with minimal performance consequences. We find that 44GB of RAM with 32GB of swap space allows the learning process to remain almost entirely in RAM and make sporadic use of disk as necessary.

The vSphere environment provides fixed allocations of RAM and processor cycles (measured in GHz) and allows for flexible allocation of resources among virtual nodes. Having determined the memory required for an individual learning task, we configure nodes to each support a single learning task and distribute processing power evenly among the nodes. Feature extraction parallelizes well and requires holding minimal state in memory, and therefore is able to leverage available processing power on nodes. We configure the master node with a larger memory allocation to support interactive learning experiments by multiple users at the same time. Although vSphere supports over-provisioning resources to increase utilization, we find that reserving full resources for each of our instances increases stability.

Chapter 5

Malware Detection Case Study

In this chapter we introduce the malware detection case study which we use to evaluate our malicious content detection platform. We collect our case study data from VirusTotal, a online service which receives malware binaries from user submissions [133]. VirusTotal scans each submission with a suite of malware detectors and analyzes binaries with both static and dynamic analysis techniques. The case study includes 2.9 million scans of over 1 million binaries appearing on VirusTotal between January 2012 and June 2014. We generate the case study dataset by drawing a random sample from every hour of VirusTotal submissions and including any Windows malware selected in the sample. To improve label quality, we submit a selection of binaries for rescanning in February and March 2015.

We also present the malware case study feature extraction implementation. We include feature extraction in the presentation of the case study since the detection platform requires customized feature extraction for each application domain, even though the broader computational structures are applicable across application domains. We review the types of raw feature data available from VirusTotal, as well as the specific techniques we use to generate feature vectors. We also examine the impact of different types of feature data on classification results, and the features which receive the highest model weight.

We structure the chapter as follows. In Section 5.1 we present an overview of our dataset, and in Section 5.2 we analyze the vendor labels from VirusTotal and present our approach to obtaining accurate labels for evaluation. In Section 5.3 we present our case study feature extraction techniques, and Section 5.4 analyzes the impact of different features on classification results.

5.1 Data Overview

In this section we present an overview of the malware detection case study dataset which we obtain from VirusTotal. Several characteristics that improve the quality of our evaluation distinguish our VirusTotal dataset from a generic dataset containing pairs of samples and

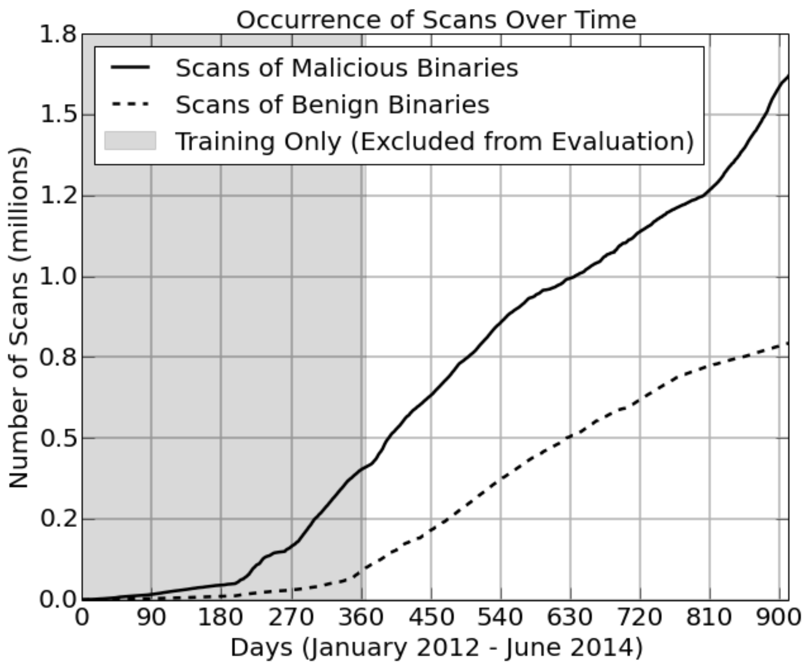
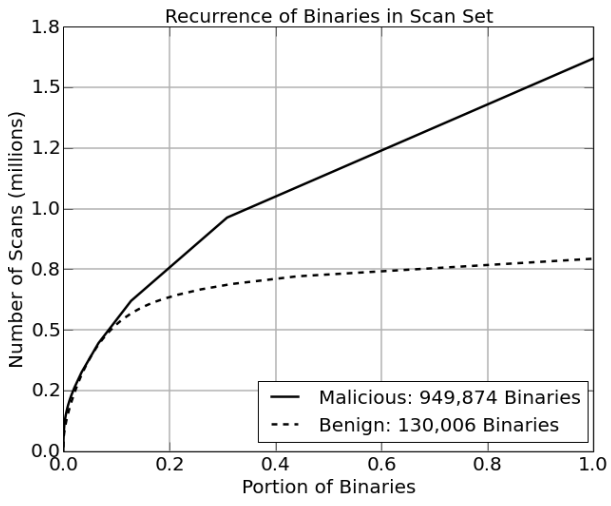


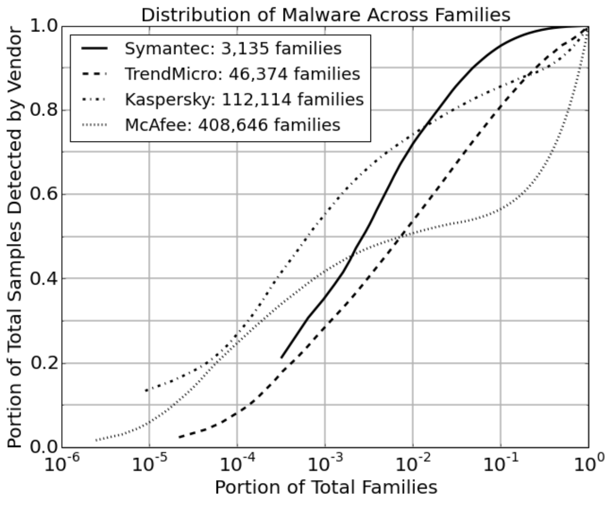
Figure 5.1: Our case study covers submissions to VirusTotal from January 2012 through June 2014. We designate the first year of data for training purposes only, and use the remaining 18 months for evaluating detection performance. Note that there is less data available during the first 200 days due to decreased availability of dynamic analysis data from VirusTotal, not an underlying phenomenon in the submission process.

labels. First, VirusTotal records each submission of a binary, capturing the prevalence of different binaries over time and effects of malware polymorphism. Additionally, VirusTotal scans each submission with up-to-date scanners, even when VirusTotal has already scanned the binary. Repeated scanning of binaries over time captures temporal labeling knowledge necessary for accurate evaluation. Finally, VirusTotal receives hundreds of thousands of submissions daily from all over the world, offering a diverse perspective which helps to accurately capture the relative ordering of binaries.

Since our detection platform explicitly incorporates the progression of time, we begin by examining the distribution of the dataset over time. Figure 5.1 presents the chronological distribution of the case study dataset, which we divide into a training period from January 2012 through December 2012 and an evaluation period from January 2013 through June 2014. 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. The evaluation period contains a steady stream of both benign and malicious scans, with approximately twice as many malicious scans. Although our



(a)



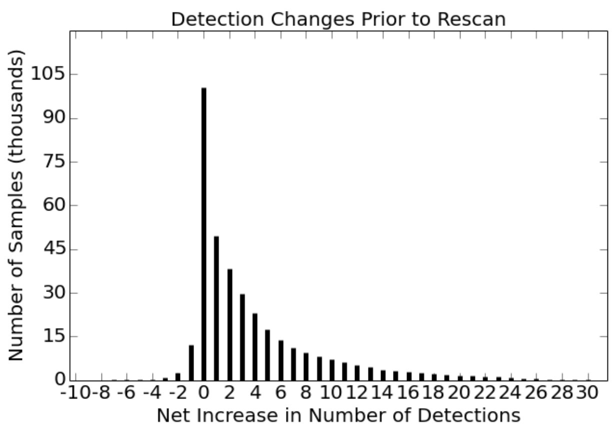
(b)

Figure 5.2: The case study dataset represents a diverse collection of malicious and benign binaries. Figure 5.2a presents the prevalence of repeat binary submissions in our dataset. Reflecting the effects of polymorphism, benign binaries tend to repeat more than malicious binaries, but no small set of binaries dominates all submissions. Figure 5.2b presents the distribution of malware across families, with each vendor regarding 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).

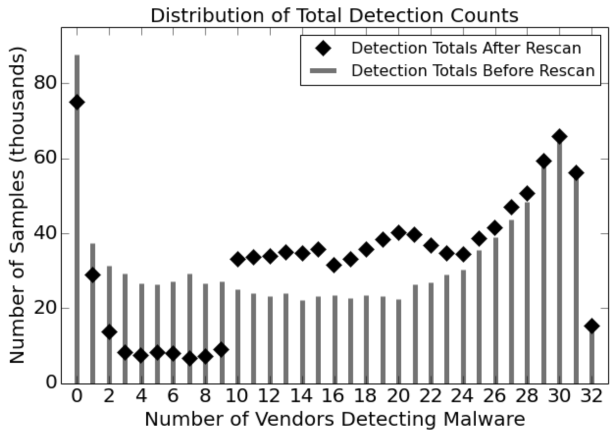
dataset contains 2.9 million scans total, Figure 5.1 reflects only approximately 1.65 million scans of malicious binaries and .85 million scans of benign binaries. The remaining scans are rescans we manually request during dataset collection in February and March 2015 to improve label quality.

Including multiple submissions of individual binaries introduces potential for individual binaries to dominate the dataset. Figure 5.2a presents the prevalence of repeated binaries in our dataset, including the total numbers of scans and unique binaries divided according to label. We exclude rescans we explicitly request in February and March 2015 which are chronologically outside the period of our analysis. We structure Figure 5.2a to present the largest possible number of scans corresponding to any selection of a fixed portion of binaries. Although the number of submissions per binary varies, no small set of binaries dominates the entire dataset. Malicious binaries outnumber benign binaries by 949,874 to 130,006, but benign binaries receive approximately half as many scans as malicious binaries. The presence of polymorphic malware likely accounts for the less frequent submission of malicious binaries. Correspondingly, flat portion of the malicious submission curve indicates that 70% of malicious binaries receive only a single submission.

We also demonstrate that our dataset is diversely composed of different malware families.



(a)



(b)

Figure 5.3: We observe that labels change over time as vendors update signatures and therefore selectively rescan binaries to obtain the best possible evaluation labels. Figure 5.3a presents the net increase in number of detections from first to last scan prior to selective rescanning. Note that we only include binaries with multiple scans. Figure 5.3b presents the distribution of total detection counts before and after rescan of the 352,116 binaries which had fewer than 10 detections. We observe a decrease in binaries with few or zero detections, demonstrating detection of new malware and increased consensus among detectors.

Figure 5.2b 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.

5.2 Labeling Analysis

In this section we examine labeling data from VirusTotal and present our approach to labeling binaries. Although the contents of a binary may not change over time, the communal understanding of whether the binary is malicious does change. We characterize changes in labeling over time, and present a rescanning strategy designed to reduce ambiguity and identify as much malware as possible. We also motivate selection of a detection threshold for harmonizing labels from multiple vendors into a single label. Although we observe scan results from up to 80 different vendor's detectors, some of these detectors are only sporadically present in the data. Since we are interested to observe properties of the detectors in aggregate, we restrict our work to the 32 vendors present in at least 97% of scan results.[1] Note that VirusTotal applies static scanners customized for the VirusTotal environment, so results may differ from retail or enterprise products.

As stated in Section 5.1, each submission to VirusTotal triggers scanning with a suite of detectors regardless of whether the binary has been previously scanned. We refer to the elapsed time between the first submission of the binary and a scan as the *age* of the scan results, and say that binaries only submitted once have labels with age zero. When queried for a binary, VirusTotal does not automatically rescan the binary and instead returns the most recent detection. Given that the frequency of submission varies across binaries, the quality of detection results will vary as some results are younger than others.

Recall that vendors prefer low false positives rates due to the significant negative effects of accidentally marking a benign program, such as a critical system file, as malicious. Figure 5.3a presents the net change in the total number of positive scans between the first and last scan for all binaries in our dataset which VirusTotal has scanned multiple times prior to our selective rescan. First, notice that there are often changes in the number of positives between the first and last scan, demonstrating that labels do not remain static over time. Second, notice that the change overwhelmingly favors increasing the number of positive scan results, confirming that vendors prefer false negatives over false positives and are unlikely to label a binary benign once the binary has been labeled malicious.

Although each binary has detection results from as many as 32 detectors available, we must ultimately label binaries as benign or malicious. We apply a threshold of four detections to label a binary malicious. Figure 5.3a demonstrates that decreases by four or more detections are rare, suggesting that a binary receiving false positives from four vendors is also rare.

Given the demonstrated shift in labels over time, we selectively request VirusTotal to rescan binaries depending on label age and the results of the most recent scan. Due to the rarity of large decreases in number of detection displayed in Figure 5.3a we only rescan binaries with fewer than 10 detections in the most recent scan, leading to rescanning of 352,116 binaries. We conduct rescans in February 2015, 8 months after the end of our evaluation. We refer to the final scan results for a binary, whether we request the final rescan or not, as the *gold label* for the binary.

During rescanning, we observe that malware detections increase and disagreement over whether a binary is malicious decreases. Figure 5.3b presents the distribution of total number

[1]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.

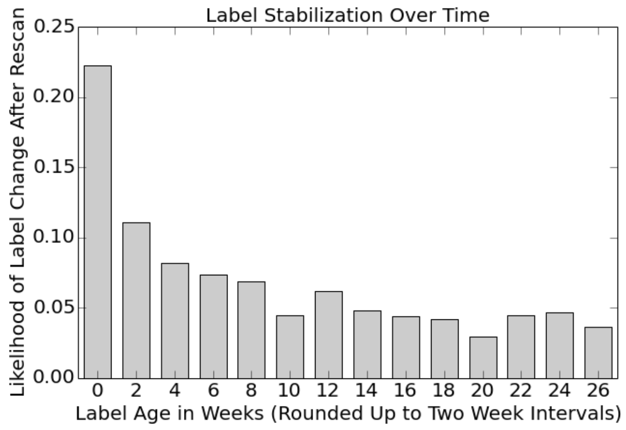


Figure 5.4: Labels are most likely to change in the weeks immediately after a binary appears, and become more stable over time. Figure 5.4 presents the likelihood of label change during selective rescaning as a function of the age of the most recent detection results.

of detections received by each binary, both before and after the selective rescan. Detection results after rescan shift at 10 detections because we only rescan binaries with fewer than 10 detections and detection latency causes total detections to increases over time. The decrease in number of binaries with zero malware detections demonstrates the discovery of new malware, and the decrease in number of binaries with few detections demonstrates emerging consensus among vendors for binaries which initially received few detections. Note that the decrease occurring at 31 and 32 detections occurs due to variations in the set of detectors included in each scan. As detectors are sporadically omitted, binaries scanned with fewer than the full suite of 32 detectors have a lower maximum number of detections, limiting the number of binaries eligible to have the highest detection totals.

Lastly, we examine the delay with which detection occurs. Figure 5.4 examines the likelihood that we observe a label change while re-scanning a binary as a function of the age of the most recent labeling of a binary. We define a label change as crossing the four detection threshold, since changes that do not cross the threshold have no impact on the label for the purposes of our evaluation. Notice that younger labels are more likely to change during a rescan than older labels, confirming that labels are most volatile after a binary first appears.

5.3 Feature Extraction

Having discussed collection, characteristics and labeling of our dataset, we now present our approach to feature extraction. We include feature extraction in our presentation of the case study because feature extraction techniques are application domain specific, unlike the other components of the detection platform which generalize across application domains. Our

Name	Description	Example
Binary Metadata	Metadata from TrID and ExifTool	`PECompact2 compressed`
Digital Signing	Certificate chain identity attributes	`Google Inc`; `Somoto Ltd`
Heuristic Tools	Tools from ClamAV, Symantec	`InstallShield setup`; `DirectShow filter`
Packer Detection	Packer or crypter used on binary	`UPX`; `NSIS`; `Armadillo`
PE Properties	Section hashes, entropies; Resource list, types	`image/x-png`; `hash:eb0c7c289436...`
Static Imports	Referenced library names and functions	`msvcrt.dll/ldiv`; `certcli.dll`

Table 5.1: Static feature extraction covers a broad range of attributes, including signature information, measured properties such as entropy and the results of heuristic tools that operate statically. We apply different vectorization techniques to each attribute according to the type and structure of the data in the attribute.

design process included refinements to feature extraction techniques after having viewed performance on our entire dataset, potentially introducing a temporal inconsistency into our evaluation. We consider the risk of this inconsistency to be minimal since all features ultimately capture broad attributes rather than idiosyncrasies of specific binaries or malware families. We begin by presenting the static and dynamic attributes available from VirusTotal, and then discuss the vectorization techniques we apply to each attribute. Note that since VirusTotal only applies dynamic analysis techniques upon the first analysis of a binary, when vectorizing subsequent submissions we refer to the dynamic analysis from the first submission to generate dynamic features, causing the features to remain constant over time. Since the static features are derived directly from the binaries themselves, and the binaries do not change, the static features also remain constant over time.

Static Attributes of Binaries

VirusTotal applies a suite of static analysis tools to each binary. Table 5.1 presents an overview of the types of static attributes VirusTotal provides, and we discuss each attribute in greater detail below.

Binary Metadata VirusTotal uses the file signature magic number [139] and TrID tool [129] to predict the file type of a binary, and ExifTool [35] to extract a broad range of metadata from the binary. The magic number is a series of bytes appearing at a constant location in the file that correspond to the file format, resulting in a single file type prediction such as `PE32 executable for MS Windows (GUI) Intel 80386 32-bit`. TrID contains a signature library to identify file types from binaries and returns a probabilistic estimate of the file type, such as `Win32 Executable (generic) (52.9%)\nGeneric Win/DOS Executable (23.5%)\nDOS Executable Generic (23.4%)`. ExifTool returns a dictionary with a varied set of keys containing binary metadata. Information returned by ExifTool commonly includes the MIMEType, version numbers and character encodings.

Digital Signing Samples may contain a signature to verify authenticity and establish a chain of trust. VirusTotal reports whether each binary has been signed, the publisher that signed the binary, the date of signing, and the chain of signers. For each party in the chain of signers, VirusTotal reports the identity of the signer, signature thumbprint and serial number. VirusTotal also provides additional fields that include comments, product name, description, copyright, internal name, and publisher, which we use when available.

Heuristic Tools This group of attributes contains the binary output of two black-box heuristic tools which VirusTotal applies to each binary. ClamAV PUA checks for Potentially Unwanted Applications (PUAs) that are not malicious by themselves, but can be used in malicious applications [19]. Symantec Suspicious Insight returns a binary value based on reputation data [122].

Packer Detection Malware creators employ tools known as *packers* or *cryptors* to hide the contents of executables and evade detection. VirusTotal applies three packer/cryptor detecters: Command Unpacker, F-Prot Unpacker and PEiD [83]. Command Unpacker and PEiD each identify a single packer which may have been applied to a binary, and F-Prot Unpacker identifies a list of packers which may have been applied to a binary. Common packer names appearing in detection results include `UPX`, `NSIS` and `Armadillo`.

Portable Executable Format In addition to metadata and signature attributes, VirusTotal provides attributes summarizing the actual contents of the binary. The Portable Executable file format specifies different *sections* within the executable, such as `.text` for executable code, `.data` for initialized variables and `.bss` for uninitialized variables [88]. VirusTotal extracts summary properties from binary sections including section hashes, entropies, lengths and types. The Portable Executable format also specifies a `.rsrc` section for *resources*, such as bitmaps, cursors and icons. VirusTotal summarizes binary resources with resource type (e.g. `RT_BITMAP`, `RT_CURSOR`, `RT_HTML`), language and hash.

Static Imports VirusTotal provides deeper information summarizing the `.idata` section of the binary, which specifies function imports. VirusTotal supplies the name of each library (e.g. `KERNEL32.dll`, `USER32.dll`) which the binary imports functions from, as well as the functions the binary imports from the library. Note that binaries may import functions either by name (e.g. `ExitProcess`, `SetMenu`) or by ordinal reference (e.g. `Ord(490)`).

Dynamic Attributes of Binaries

VirusTotal executes each binary in the Cuckoo sandbox [125] the first time the binary appears. The Cuckoo sandbox produces a JSON report summarizing behavior, including filesystem and registry operations, API calls and network activity. Table 5.2 presents an overview

Name	Description	Example
Dynamic Imports	Dynamically loaded libraries	`shell32.dll`; `dnsapi.dll`
File Operations	Number of operations; File paths accessed	`C:\WINDOWS\system32\mshtml.tlb`
Mutex Operations	Each created or opened mutex	`ShimCacheMutex`; `RasPbFile`
Network Operations	IPs accessed; HTTP requests; DNS requests	`66.150.14.*`; `b.liteflames.com`
Processes	Created, injected or terminated process names	`python.exe`; `cmd.exe`
Registry Operations	Registry key set or delete operations	`SET: ...\WindowsUpdate\AU\NoAutoUpdate`
Windows API Calls	n-grams of Windows API calls	`DeviceIoControl | IsDebuggerPresent`

Table 5.2: Dynamic feature extraction covers a broad range of execution behavior, including filesystem and registry operations, API calls and network activity. We apply different vectorization techniques to each dynamic attribute according to the type and structure of the data in the attribute.

of the dynamic attribute information available from the Cuckoo sandbox report. We discuss each dynamic attribute in greater detail below.

Dynamic Imports The `.idata` section allows binaries to specify libraries to dynamically link at *load-time*, before execution begins. Alternately, binaries may dynamically link libraries at *run-time*, after execution begins. Run-time linking uses the `LoadLibrary` and `LoadLibraryEx` Windows API calls [14]. Cuckoo sandbox records all attempts to load a library, as well as whether the attempt was successful.

File Operations Interaction with persistent storage can allow malicious binaries to achieve a persistent presence on an end host or steal user data. The Cuckoo sandbox reports all filesystem operations, including attempts to open, read, copy, delete, move, download, replace or write files. The report includes any path arguments applicable to the file operation and whether the operation was successful.

Mutex Operations Windows provides primitives such as semaphore and mutex that programs can use to control access to shared resources in multithreaded programs. The mutex primitive implements a lock, allowing a thread to assume possession of the mutex and lock all other threads out until the possessing thread releasees the mutex [12]. The `CreateMutex` and `CreateMutexEx` Windows API calls allow threads to create and name a new mutex. Cuckoo sandbox records each mutex creation and access operation, as well as whether the operation was successful.

Network Operations Network connectivity allows malicious binaries to conduct coordinated activities as part of a larger malicious effort, such as a distributed denial of service attack or spam campaign. Network connectivity also allows exfiltration of sensitive user data. Although VirusTotal makes the full network activity of a binary available as a packet capture (pcap) file, we operate on the summarized network activity available through the Cuckoo sandbox report. The report includes all DNS activity from the binary, including the

	Name	**Null Attribute**	**Categorical**	**String**	**Ordinal**	**Sequential**
Static	Binary Metadata	✓	✓	✓		
	Digital Signing	✓	✓	✓		
	Heuristic Tools	✓	✓			
	Packer Detection	✓	✓			
	PE Properties	✓	✓	✓	✓	
	Static Imports	✓	✓			
Dynamic	Dynamic Imports	✓	✓			
	File Operations	✓	✓		✓	
	Mutex Operations	✓	✓	✓		
	Network Operations	✓	✓		✓	
	Processes	✓	✓			
	Registry Operations	✓	✓			
	Windows API Calls	✓	✓			✓

Table 5.3: Machine learning algorithms commonly require vectorized representations of samples rather than raw attribute data. We present five techniques for transforming raw attributes into feature vectors, and apply selected techniques to each type of raw attribute data.

requested domain name and the resolution result. The report also includes all hosts the binary contacts and a summary of all TCP and UCP activity including source and destination port and IP address. Lastly, the report summarizes all HTTP requests, including request URLs, methods and headers.

Processes Cuckoo sandbox records any processes the binary creates, injects or terminates. For any action, Cuckoo reports the executable path or name of the process, as well as whether the action was successful. Cuckoo sandbox also reports any shell commands the binary issues.

Registry Operations The *registry* is a system-wide key-value store containing configuration data for both application and system components [13]. The registry is an attractive target for attackers seeking to reconfigure system properties to evade detection or introduce additional vulnerabilities. Cuckoo sandbox reports all registry operations which either set or delete a registry key. For set operations, the report includes the key, the new value, the type of the value, and whether the operation was successful. For delete operations, the report includes the key deleted and whether the operation was successful.

Windows API Calls The Windows system libraries offer a diverse range of application functionality, including device interaction, memory management and threading. Cuckoo sandbox reports the entire call sequence of a binary, including function names, argument names and values, return values and timestamps. While the individual API calls of a binary are useful for detection, the ordering information in the sequence of API calls can further distinguish malicious content.

Approaches to Feature Vectorization

Having discussed the raw attribute data available for each binary, we now present our process for transforming raw data into feature vectors. We present five general techniques for vectorizing attribute data, and identify the techniques which we apply to each type of raw data. Table 5.3 presents an overview of our application of vectorization techniques to raw attribute data.

Null Attribute In some cases an attribute for a given binary may be either missing from the dataset, present the value `None`, or present the empty string. For example, a binary may be unsigned and consequently lack signature information. Since the absence of an attribute reveals information about a binary, we capture the absence of an attribute as a binary feature where the feature is 1 if and only if the attribute is unavailable. We apply null attribute vectorization to all attributes.

Categorical Categorical vectorization applies to values drawn from a finite set which has no inherent order, such as API calls. Each attribute value corresponds to a unique dimension in the feature vector. 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. Categorical vectorization requires maintenance as new binaries appear to introduce additional dimensions for new attribute values.

We apply categorical vectorization to all attributes. In some cases we vectorize a modified version of attribute data to contain the dimensionality of the feature space and help features to generalize across binaries. For example, with Windows API Call data we disregard argument values and assign a dimension to each unique tuple of *(function name, specified argument names)* that appears in the execution trace. When vectorizing URL data, we convert all URLs to lower case and include separate dimensions for keys and values in the query string.

String Many important attributes appear as unbounded strings, such as the comments field of the binary signature. 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 onto a different dimension. Therefore, we capture 3-grams of these strings, where each contiguous sequence of three characters represents a distinct 3-gram. We consider each 3-gram as a separate dimension. However, this approach is still sensitive to variations that alter 3-grams.

To reduce sensitivity to variations that alter 3-grams, we perform an additional string transformation. 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 vectorizing the portable executable resource names, which can exhibit arbitrary byte sequences.

Ordinal Ordinal attributes assume a specific value in an ordered range of possibilities, such as the size of a binary. We describe ordinal attributes using a binning scheme 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 example, we discretize the entropy of a section into 10 bins spanning from 0 to 8 bits per byte. For attributes that vary widely, we use a non-linear scheme to prevent large values from overwhelming small values during training. For example, we discretize the number of written files v to a value i such that $3^i \leq v < 3^{i+1}$, where the exponential sizing of bins accounts for the large dynamic range of filesystem behavior.

Sequential Sequential attribute vectorization applies to token sequences where each token assumes a finite range of values. Sequential attributes are strongly related to free-form string attributes, although the individual tokens are not restricted to being individual characters. We use sequential attribute vectorization 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.

5.4 Feature Impact

Having presented raw feature attributes and vectorization techniques, we now examine the effectiveness of each type of feature for detection. In the interest of understanding the dataset as a whole, we train a model over all data from all dates. We use logistic regression to train a linear model $\mathbf{w}$, allowing us to present the model as a list of features and associated weights. Table 5.4 shows the most positively (correlated with malicious) and negatively (correlated with benign) weighted dimensions of $\mathbf{w}$. For instance, not detecting a packer is the single feature most indicative of benign, while a particular section, as specified by its hash, is the single feature most indicative of malicious.

However, inspecting the weight 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 also interested in grouping low-level features together into high level concepts that reflect the original measurements.

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`

(a)

w	dimension semantic
-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

(b)

Table 5.4: The highest weighted dimensions span diverse attributes, including section and resource hashes, binary signers, network behavior and API calls. Table 5.4a presents the individual dimensions most associated with malicious content, and Table 5.4b presents the individual dimensions most associated with benign content. **w** indicates the weight assigned to the dimension, and *dimension semantic* indicates the semantic significance of the dimension. The pattern of the mutex name in Table 5.4a is a result of string simplification.

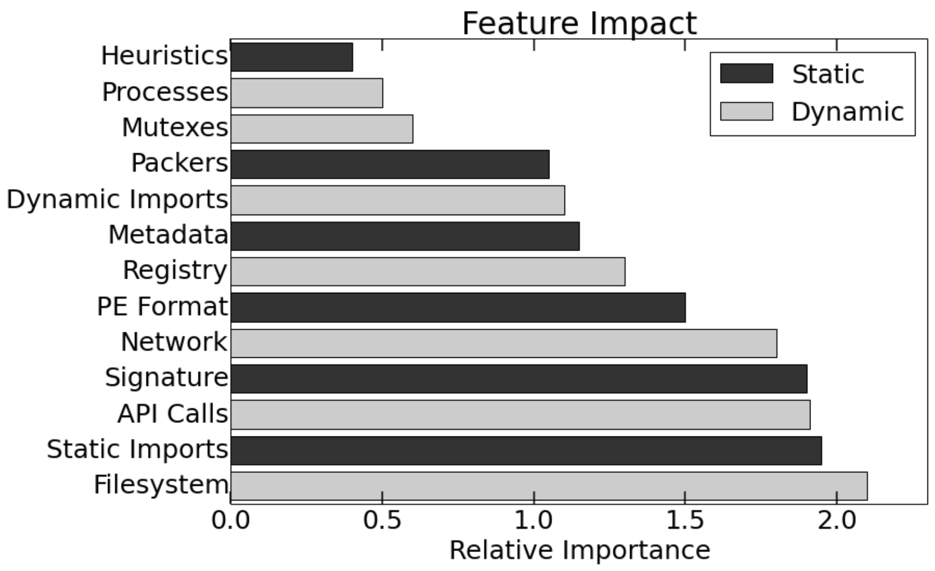


Figure 5.5: Both static and dynamic features have a significant impact on classification outcomes. We calculate feature impact according to Equation 5.1, which takes into account both the weight and prevalence of the feature.

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

$$I_S = \sqrt{\operatorname*{Var}_{\mathbf{x}} \left[\sum_{k \in S} \mathbf{x}_k \mathbf{w}_k \right]} \tag{5.1}$$

Using this ranking method, Figure 5.5 shows the global ranking of the features when grouped by their original measurements. The most important measurements are thus the filesystem operations, static imports, API call sequence and digital signature, while the least useful measurement is the heuristic tools.

Chapter 6

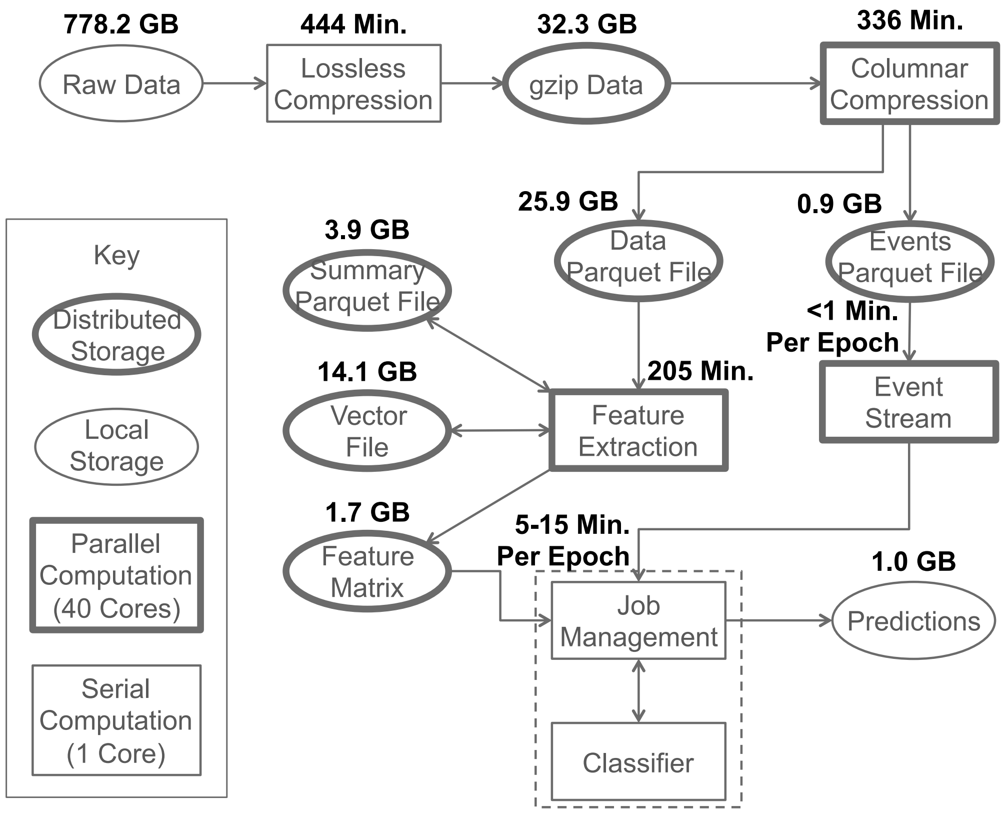
Experimental Results

In this chapter we apply the detection platform implementation to the case study dataset and present empirical results spanning scalability, methodology and detection performance. Our examination of scalability includes both storage and computation requirements, and demonstrates feasibility of working with industry scale data. Our comparison of different policies dividing and labeling samples for training and testing motivates the importance of temporally consistent samples and labels in evaluation. Lastly, our temporally consistent evaluation demonstrates the effectiveness of our strategy for integrating a human reviewer, improving detection rates beyond the best anti-virus vendors on VirusTotal.

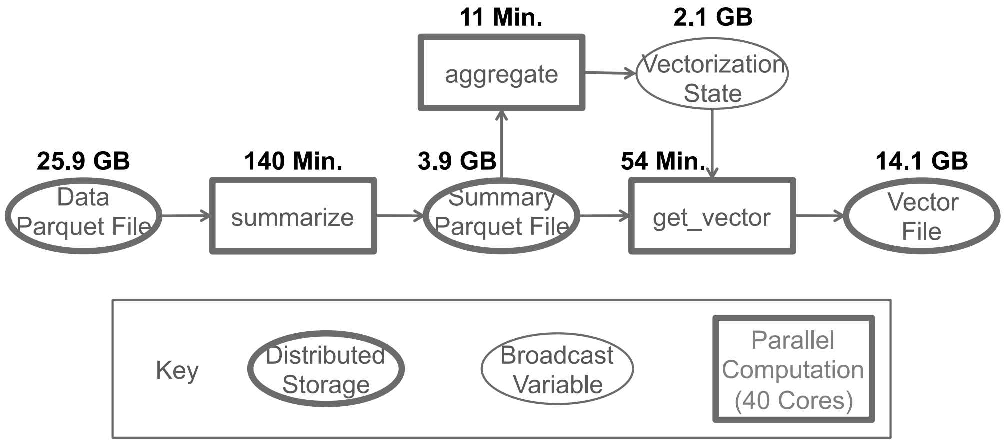
We begin by examining the scalability of the detection platform in Section 6.1. Our evaluation data includes over 1 million samples occupying 778.2GB, exceeding the number of individual samples that industrial anti-virus providers currently observe daily [63]. We find that through data compression and processing we are able to reduce the 778.2GB of sample data to 3.9GB, a 200-fold decrease in storage requirements. We also demonstrate scalable use of computational resources, processing our entire dataset in hours using 40 cores and demonstrating ability to keep pace with the arrival of industry scale data in real time.

In Section 6.2, we examine the impact of variations in evaluation methodology on detection performance measurements. Our results motivate the detection platform's careful handling of temporal consistency for both samples and labels. We find that at a 0.5% false positive rate, temporally inconsistent evaluation can achieve detection as high as 92% while temporally consistent evaluation achieves only 72% detection. We also compare the relative impacts of temporal samples, temporal labels and variations in sample prevalence on detection performance measurements.

Having examined evaluation methodology, we evaluate detection performance in Section 6.3. We find that our strategy for integrating a human reviewer increases detection from 72% to 89% at a 0.5% false positive rate, yielding a substantial performance improvement over anti-virus vendors. To account for possible discrepancies between information available during our analysis and information available to vendors, we also measure detec-



(a) Resource consumption across entire detection platform.



(b) Resource consumption within Feature Extraction module.

Figure 6.1: The detection platform implementation scales to handle large amounts of data through a combination of compression, data reduction and distributed processing. Our evaluation data exceeds the number of samples processed daily by a large anti-virus vendor.

tion with static features and with training labels from a single vendor. We find that although both alternate experimental conditions impact detection performance our detection results continue to surpass the present state of the art.

Lastly, in Section 6.4 we examine the impact on detection performance of variations in reviewer accuracy and availability as well as query strategy for selecting samples for review. We assign true and false positive rates to simulated reviewers, and find that with 5% reviewer false positive rate and 80% reviewer true positive rate our detection decreases only six percentage points. Our examination of reviewer availability reveals that with 80 queries daily on average, equivalent to approximately 7.3% of the entire dataset, we achieve detection similar to using gold-standard labels for the entire dataset. Finally, we find that our query strategy improves detection compared to uncertainty sampling, a technique from prior work in active learning.

6.1 Scalability

In this section we evaluate the scalability of our detection platform implementation using the case study dataset. Since the case study data includes over 1 million samples, our evaluation exceeds the daily load experienced by a large anti-virus vendor [63]. Figure 6.1 presents an overview of the storage and computation requirements of each component of our implementation. We discuss the usage of storage and processing resources below, and conclude with an analysis of resource requirements, potential bottlenecks and strategies for deployment at industrial scale. Note that our examination of scalability does not include the cost of static and dynamic analysis of the binaries themselves, which in the case of our work is performed by VirusTotal. To contain the costs of sample analysis in practice, active learning could select samples for analysis to reduce total workload.

Although the uncompressed case study data occupies 778.2GB, through compression and data reduction techniques we reduce storage consumption 200-fold to 3.9GB. Figure 6.1a depicts the reduction in data size through successive processing steps, including Lossless Compression, Columnar Compression and the summarize component of Feature Extraction. While Lossless Compression and Columnar Compression both attempt to retain the full original data, Columnar Compression achieves more efficient storage by fitting all data to a common schema and removing the need to replicate schema data with each record. Since all feature vectorization depends on the Summary Parquet File, a storage-limited deployment may discard the original Raw Data and all intermediate representations. Since our case study data requires only 3.9GB and exceeds what a large vendor processes daily, we conclude that our implementation meets scalability requirements for raw feature storage for industrial applications.

Separate from the storage of raw feature data, feature vectors, labeling data and predictions also require storage. The Vector File and Feature Matrix store the feature vectors, and measure 14.1GB and 1.7GB respectively for the case study dataset. The Vector File stores `(sample_id, vector)` tuples as isolated serialized python objects, and Feature Matrix stores a single sparse matrix with rows ordered according to `sample_id`. The Feature Matrix uses compression to conserve disk and network resources, and the Vector File remains uncompressed to facilitate integration with distributed learning implementations. The Events Parquet File holds all labeling data and requires less than 1GB, and in an actual deployment would be replaced by a database of current label knowledge. Lastly, Predictions also remains scalable with only 1GB in storage, including vendor labels and other data to facilitate analysis after platform execution.

We now examine scalability of processing requirements, beginning with Lossless Compression, Columnar Compression and the summarize function within Feature Extraction. Compression and summarization execute once per sample, allowing resources to remain proportional to the arrival of samples rather than aggregated data. Figure 6.1a presents the runtime of system components, and Figure 6.1b presents the runtime of feature extraction in greater detail. Note that processing times for parallel and serial computation reflect execution on all 40 cores and a single core respectively. Since the case study dataset exceeds the daily volume of samples processed by vendors and we complete execution in hours, we conclude that the components which execute once per sample could operate at scale on industrial data in real time.

Detection platform components that operate over aggregated data present a potential bottleneck and complicate scalability analysis. The aggregate and get_vector functions within Feature Extraction operate on the case study dataset in approximately one hour, but would require additional runtime or resources to operate on industry scale data accumulated over weeks or months. Since feature vectorization must occur once per epoch, vectorization of 1 million samples per hour over a one week retraining period would allow approximately 168 million samples, or 18 months of data for a large anti-virus vendor [63]. While increased resources beyond the 40-core cluster in our evaluation would allow processing at larger scale, algorithmic improvements to feature extraction could also improve scalability. Although our present implementation re-generates feature vectors for each sample, stateless feature extraction (as discussed in Chapter 3) for some or all types of features would reduce computational burden. Stateless feature extraction requires processing each sample only once, removing the computational burden imposed by accumulated data.

The Classifier module also presents a potential obstacle to scalability by training detection models over aggregated training data. Our detection platform implementation uses the logistic regression implementation in scikit-learn [82], a machine learning library for Python that trains models on a single node. Although our training times of 5-15 minutes per model suggest scalability, the size of the feature matrix extracted from aggregate data would scale to exceed the memory limits of commodity hardware. Distributed learning implementations, such as MLlib [52, 117], offer a scalable alternative to scikit-learn by leveraging multiple machines for feature matrix storage and model training.

Although we discuss approaches to accommodate aggregate data during feature extrac-

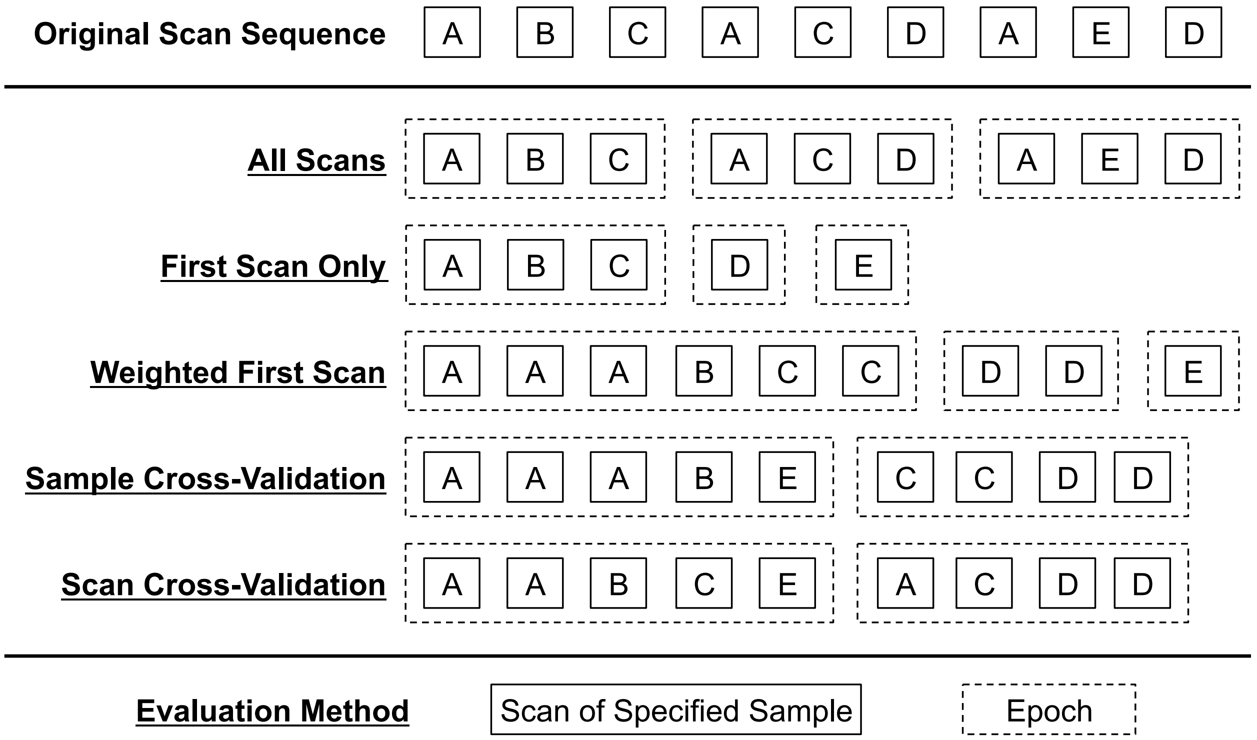


Figure 6.2: We present five approaches to partition samples for training and testing. The approaches vary their handling of temporal consistency and sample prevalence. Epochs correspond to successive retraining periods, where the samples in the first N epochs produce the model for detection in epoch $N + 1$.

tion and learning, our own evaluation operates on randomly subsampled aggregate data and achieves detection improvements over the current state of the art. Consequently, we conclude that including all previous data is *desirable* but not *necessary* for training powerful models for malicious content detection. In practice, a combination of the increased resources, algorithmic improvements (stateless feature extraction, distributed learning) and data subsampling may achieve best results.

6.2 Evaluation Methodology

In this section we examine evaluation methodology. Evaluation requires both selection and labeling of samples for training and testing. We discuss different approaches to selecting and labeling samples for training and testing, and then examine the impact of each approach on detection performance measurements for our case study dataset. We also present our approach to simulating the involvement of a human reviewer.

We begin by discussing different methods dividing data for training and testing during evaluation. Evaluation strategies may incorporate both the order in which samples ap-

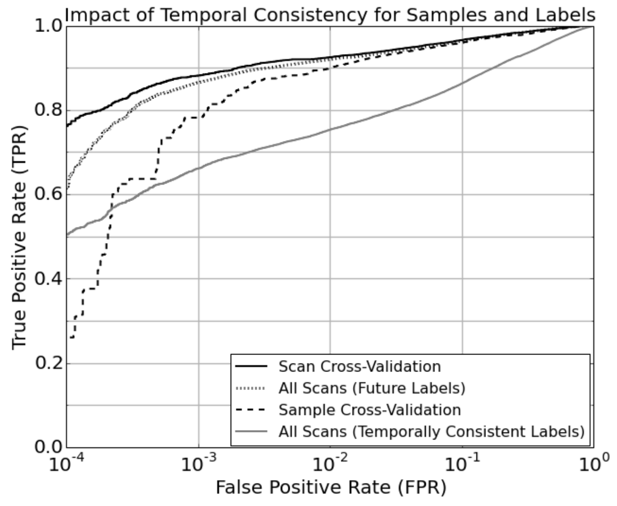
	Epoch 1			Epoch 2			Epoch 3		
Sample	A	B	C	A	C	D	A	E	D
Label at End of Epoch 1	-	-	+	▨	▨	▨	▨	▨	▨
Label at End of Epoch 2	-	-	+	-	+	-	▨	▨	▨
Label at End of Epoch 3	+	-	+	+	+	-	+	-	-
Label Months After Epoch 3	+	-	+	+	+	+	+	+	+

Figure 6.3: Evaluations must carefully handle the labeling of training and testing data to accurately represent detection performance. Although the labels of samples B and C remain constant over time, the labels of A, D and E change as knowledge improves. Use of labeling knowledge from the future inflates detection performance. For example, consider that samples A and D would be easier to detect during Epoch 3 given correctly labeled instances of samples A and D in training data from Epochs 1 and 2.

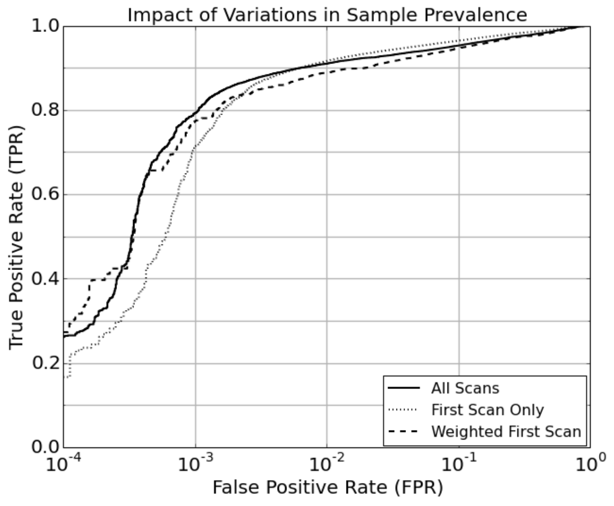
pear over time, as well as the relative prevalence of samples over time. Figure 6.2 presents an overview of different sample ordering approaches which we examine in this work. The Original Scan Sequence presents a hypothetical ordering of scans for five different samples identified A through E. The repeated occurrence of samples in the Original Scan Sequence corresponds to multiple scans of the same sample at different points in time. Each evaluation method uses a different approach to divide the Original Scan Sequence into successive retraining epochs, where the scans in the first N epochs serve as training data for detection in epoch $N + 1$.

We present five evaluation methods for dividing scans into epochs, each leveraging a distinct motivation. The All Scans method maintains the ordering and prevalence of the Original Scan Sequence, partitioning the sequence to create epochs. Allowing samples to occur in multiple epochs implies that the same sample may exist in both training and testing data, and repetition of samples potentially allows individual samples to dominate detection performance results. Consequently, we introduce the First Scan Only method which places each sample into exactly one epoch, and retains only the first occurrence of each sample. Additionally, we develop the Weighted First Scan method which maintains sample prevalence information, but places all scans in the first epoch in which a sample occurs as a compromise between the All Scans and First Scan Only approach.

We also present cross-validation evaluation approaches which divide training and testing data regardless of time. Cross-validation is most appropriate for application domains



(a)



(b)

Figure 6.4: The selection and labeling of samples for training and testing impacts performance measurements. Figure 6.4a compares variations in temporal consistency for samples and labels, with temporally consistent samples and labels underperforming alternative techniques. Figure 6.4b contrasts three approaches to correcting sample prevalence, and reveals that all approaches achieve similar detection performance.

where the data distribution remains constant over time, in contrast with the adversarial introduction of new samples to evade detection. We include cross-validation approaches primarily for comparison to prior work. While Sample Cross-Validation partitions data at the granularity of samples, Scan Cross-Validation partitions individual scans, allowing samples with multiple scans to potentially occur in both training and testing data. We include Scan Cross-Validation since the All Scans method can allow multiple scans of a sample to occur in different epochs, in contrast with Sample Cross-Validation.

Separate from methods for selecting samples for training and testing, evaluation also requires labeling the training and testing samples. Just as evaluations may reflect the emergence of samples over time, evaluations may also reflect the emergence of label knowledge over time. Evaluations that fail to recognize the emergence of samples and label knowledge over time effectively use knowledge from the future, inflating the measured accuracy of the approach. Figure 6.3 presents the evolution of label knowledge over time for the sample sequence in Figure 6.2. Notice that as epochs progress and detection mechanisms improve, some samples change label. For example, Sample A is detected as malware during Epoch 3. To describe accurate evaluation practices, we also include a labeling *after* the end of the final epoch in which additional label updates appear. Accurate evaluation should use training labels available before the corresponding testing epoch, and evaluation labels collected after the end of the final epoch.

We conduct a series of experiments measuring the impact of different evaluation methodologies on detection performance results. Figure 6.4a presents the impact of variations in temporal sample and label consistency, and Figure 6.4b presents the impact of variations in sample prevalence. All experiments use the gold-standard label provided by the final scan of a sample as evaluation ground truth. The Scan Cross-Validation, Sample Cross-Validation and All Samples (Future Labels) experiments also use gold-standard labels for training data, and the remaining experiments use temporally consistent labels. Variations in temporal consistency produce a larger impact on detection performance than variations in sample prevalence. Since the First Scan Only method of evaluation prevents the recurrence of samples in both training and testing data, we exclude any samples from the initial one-year training period from testing in Figure 6.4b, standardizing the set of samples included in testing data.

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 as an oracle that can reveal future labeling knowledge for a sample. 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 future label of the sample.

6.3 Detection Results Overview

In this section we present the effectiveness of our detection platform applied to the case study dataset. Without use of the human reviewer, our detection approach performs comparably to detectors on VirusTotal. With support from the human reviewer, we achieve performance improvements over the vendor labels supplied on VirusTotal. To account for differences in the data available during our training process and data available to vendors operating on VirusTotal, we also present detection results using only static feature data and using only training labels from a single vendor. We find that although both alternative experimental conditions impact detection performance, we continue to exceed the present state-of-the-art.

Impact of Human Reviewer 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 6.5 compares our performance to vendor detectors provided on VirusTotal. Without involvement from the human reviewer our detector achieves 72% detection at a 0.5% false positive rate, performing comparably to the best vendor detectors. With support from the human reviewer, 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 epoch, the performance curve results from varying the same detection

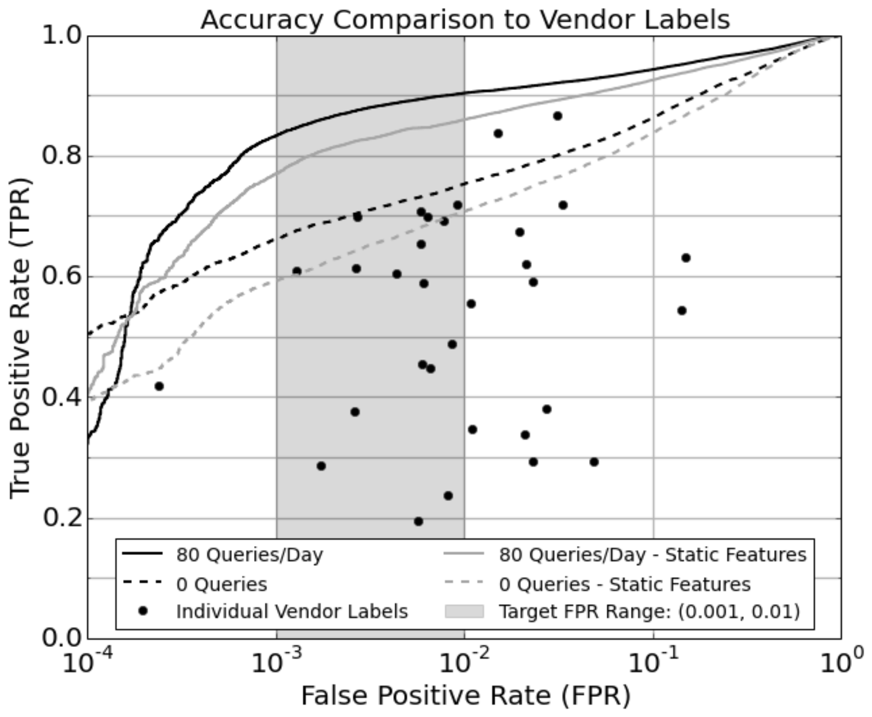
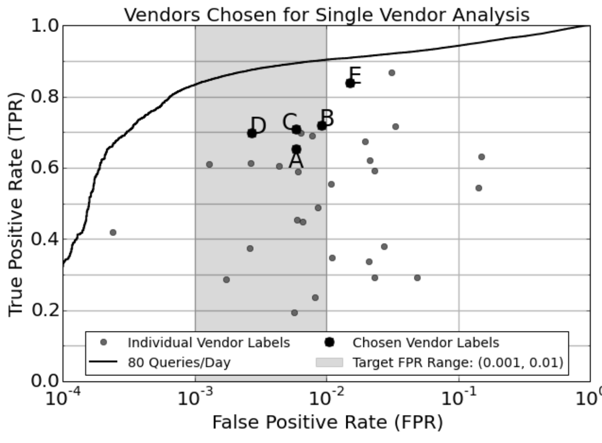


Figure 6.5: Without the human reviewer, detector performance competes with anti-virus labels available on VirusTotal. With the human reviewer, detection improves beyond vendors on VirusTotal. We tune the learning process to maximize detection in the (0.1%, 1%) false positive region.

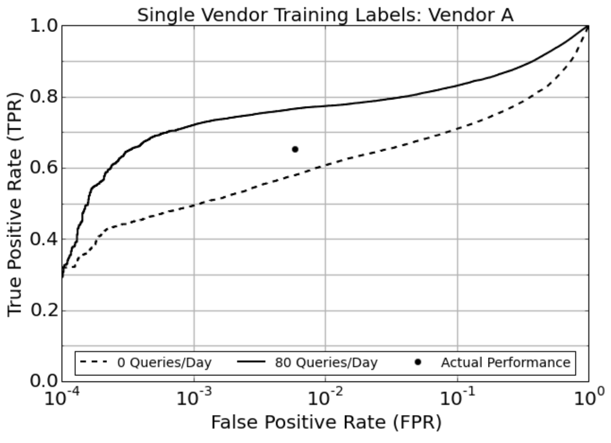
threshold across the results of each individual model.

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 attributes 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 6.5 demonstrates that our performance decreases when restricted to static features but, with support from the human reviewer, continues to surpass vendors, achieving 84% detection at a 0.5% false positive rate.

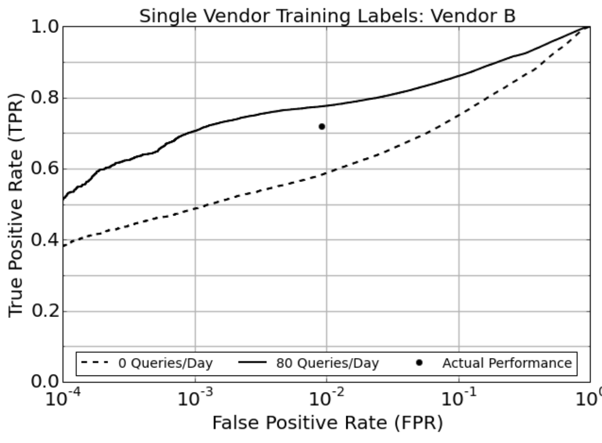
Although our detection performance in Figure 6.5 reflects use of training labels based on detections from all vendors, in practice legal and contractual constraints may force vendors to use only their own prior labels while training. Figures 6.6 and 6.7 present detection performance when using training labels from a single vendor. Our analysis includes AVG, GData, Kaspersky, McAfee and VIPRE. For each vendor, we present the performance of our detector trained on their labels both with and without support from a human reviewer, as well as the actual performance of their detector labels. We find that the human reviewer



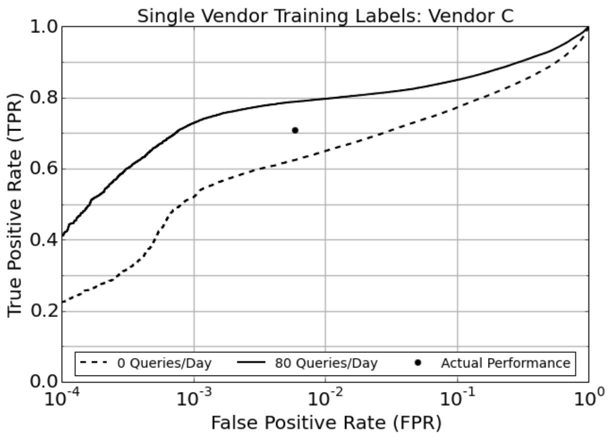
(a)



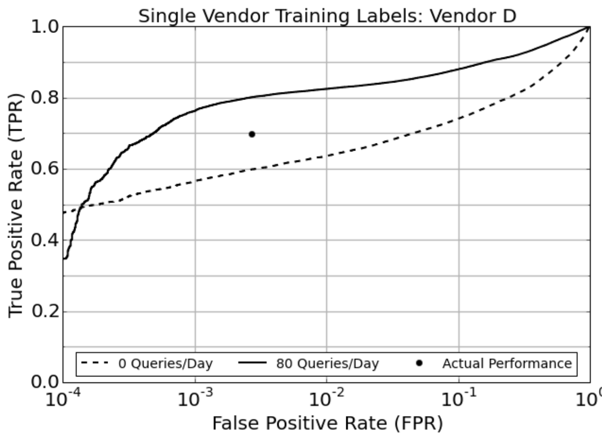
(b)



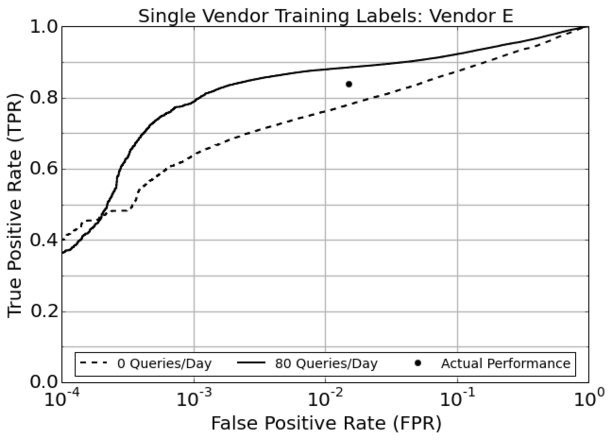
(c)



(d)

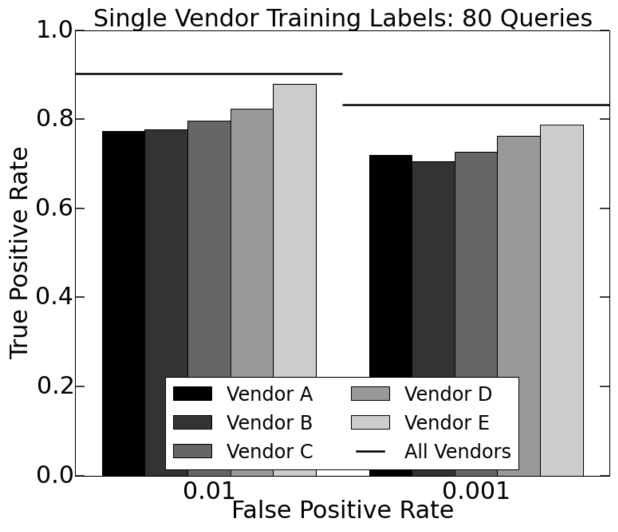


(e)

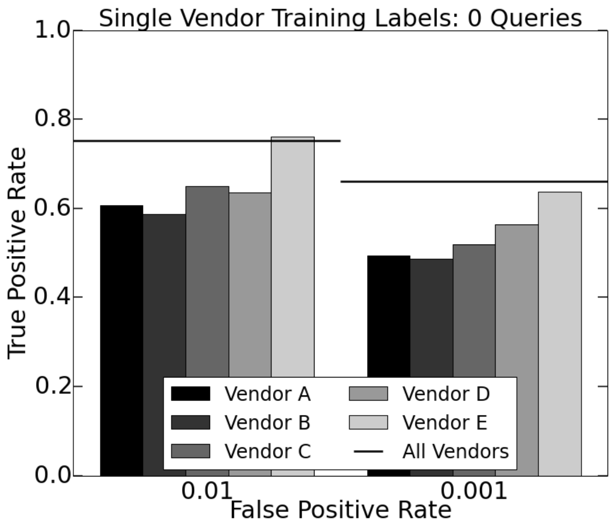


(f)

Figure 6.6: Detection performance using training labels from a single vendor. We analyze AVG, GData, Kaspersky, McAfee and VIPRE. We assign anonymous labels A through E according to true positive rate at 1% false positive rate with 80 queries daily.



(a)



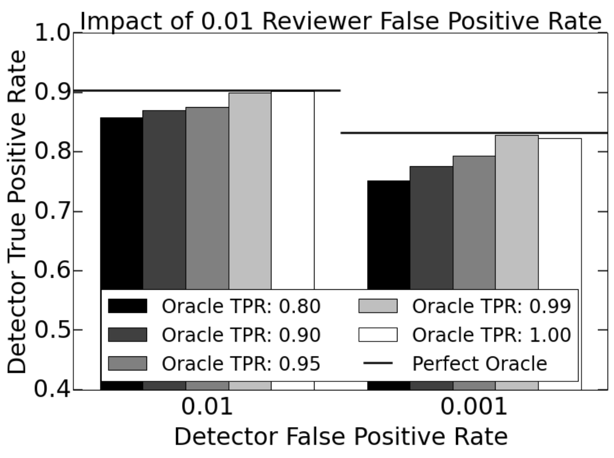
(b)

Figure 6.7: Comparison of single vendor detection results with 80 queries and 0 queries daily. "All Vendors" indicates training labels derived from labeling information drawn from all vendors rather than a single vendor.

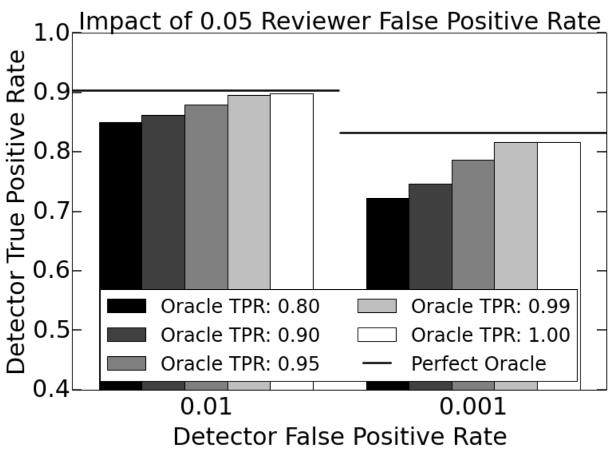
would allow each vendor to improve detection performance relative to their own prior labels.

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. The bias induced by vendor labels is increased by collusion, when vendors intentionally share data and copy or otherwise influence each other's labels. 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.

Novel Sample Detection 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 queries daily to the human reviewer, 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.



(a)



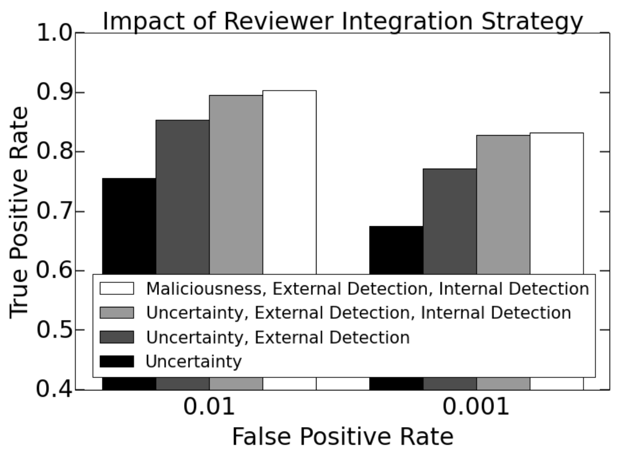
(b)

Figure 6.8: Figures 6.8a and 6.8b present the performance of our detector for imperfect human reviewers with the specified true and false positive rates. For example, given a reviewer 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. Both figures use a human reviewer budget of 80 queries daily with a one week retraining interval.

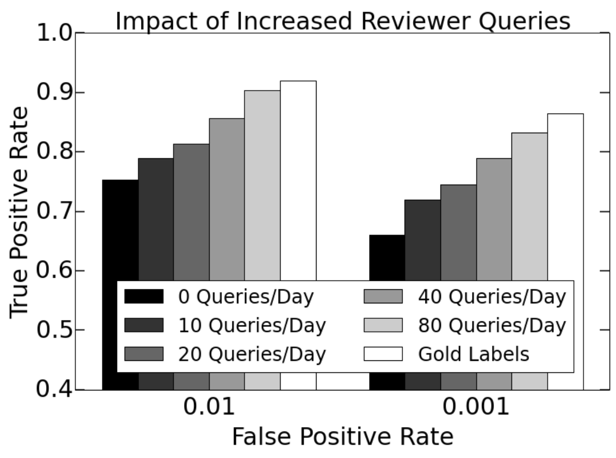
To provide a corresponding analysis of false positives, we measure our performance on the 61,213 samples which have a benign gold label and are not detected as malware by any vendor the first time the sample appears. Of these 61,213 benign samples, our detector labels 2.0% and 0.2% as malicious when 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.

6.4 Impact of Reviewer and Platform Parameters

In this section we examine the impact of variations in reviewer accuracy, availability and query strategy, as well as platform parameters governing retraining frequency and training label quality. We find that detection performance is resilient against reviewer errors, with an 80% reviewer true positive rate and 5% reviewer false positive rate decreasing detection on five percentage points at a 1% false positive rate. We also find that detection performance improvement from reviews is non-linear, with limited reviews providing significant performance improvements. We present the incremental improvement from each component of our reviewer query strategy, and demonstrate a significant improvement over prior techniques from active learning. Lastly, our analysis of retraining frequency finds that frequent retraining improves detection results with diminishing returns around a one week retraining interval.



(a)



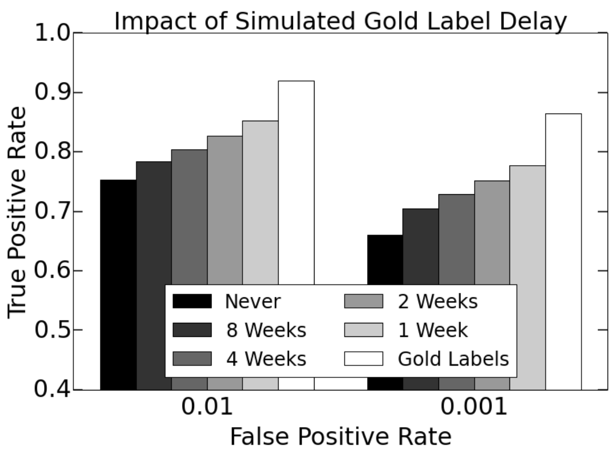
(b)

Figure 6.9: Figure 6.9a presents the impact of each component in our customized reviewer query strategy. We improve detection over the *uncertainty* sampling approach from prior work. Figure 6.9b presents performance for different reviewer query budgets, with significant return on minimal efforts and diminishing returns occurring around 80 queries daily.

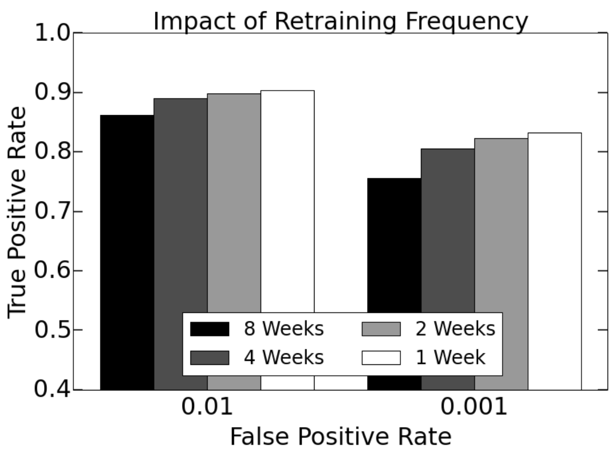
Human Reviewer Accuracy Our detection platform maintains performance even in the presence of an imperfect human reviewer. Although our work models the presence of a human reviewer as an oracle, actual human labelers may make mistakes in identifying malware from time to time. Malware creators may explicitly 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 human reviewers separately, reflecting a reviewer which is more likely to mistake malware for benign software than benign software for malware. Figures 6.8a and 6.8b present detection rates for reviewers with 1% and 5% false positive rates respectively and a range of true positive rates. For example, given a human reviewer 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.

Human Reviewer Integration Strategies Our human reviewer integration strategy represents numerous advances over generic strategies for problems outside of computer security. Figure 6.9a presents the impact of each of the three improvements we introduce and discuss in Chapter 3. For a fixed labeling budget $B = 80$, the uncertainty sampling results in 17 percentage points lower detection rate than the combination of our techniques at 0.1% false positive rate.

Human Reviewer Availability As the allowed budget for queries to the human reviewer increases, the detector accuracy increases since more accurate labels are available. Figure 6.9b presents the increase in accuracy from increased reviewer queries, with the benefit



(a)



(b)

Figure 6.10: Figure 6.10a demonstrates that regular rescans of training data may boost accuracy as training labels are more accurate. Figure 6.10b presents the impact of retraining frequency.

of 80 queries per day on average approaching the benefit of training on gold labels for all binaries. The benefit of human review 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 reviewer queries will vary with the size of the training data, increasing the amount of training data may increase reviewer 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.

Rescanning to Improve Label Quality Separate from querying the human reviewer, 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 6.10a presents the results of our analysis, which we conduct while retraining every seven days without queries to the human reviewer to observe the effects of timely training labels in isolation. Notice that even when gold labels are released after one week, detection remains approximately nine percentage points lower than when training on all gold labels. This phenomenon illustrates the necessity of human review as well as the importance of maintaining timely labels for training data.

Retraining Frequency Lastly, we examine variations in the length epochs governing the frequency of model retraining. We conduct these experiments with 80 reviewer queries per day. Figure 6.10b presents the effect of variations in the retraining period. Notice that the benefit of frequent retraining begins to diminish around two weeks.

Chapter 7

Discussion

This thesis presented a scalable platform for malicious content detection integrating machine learning and manual review. The detection platform design explicitly incorporated the passage of time, modeling the emergence of both samples and label knowledge as time progresses. Designing the detection platform around the passage of time allowed experimentation with different evaluation methodologies which disregard time entirely, recognize only the emergence of samples over time, or recognize the emergence of both samples and labels over time. We implemented the detection platform on top of Apache Spark, a distributed computation framework. Our implementation used approximately four thousand lines of Python and included a web UI for monitoring detection results.

We evaluated the detection platform in the context of a malware detection case study. The case study dataset spanned 30 months of submissions to VirusTotal, and included dynamic and static analysis of binaries as well as vendor labels. We measured change in vendor labels over time, quantifying the tendency of vendors to change labels from benign to malicious. These measurements motivated both temporally consistent labels and delayed labeling, that is postponing labeling until the end of the evaluation period. We experimentally demonstrated that temporally consistent labels can decrease detection from 92% to 72% compared to cross-validation, and that a human reviewer can increase detection back to 89% by labeling 80 samples daily on average. The case study also has demonstrated the scalability of our approach, processing our entire 778GB dataset in approximately 12 hours.

Although our work addresses many challenges in the design and deployment of practical systems for malicious content detection, the solutions we present raise new challenges of their own. We conclude by discussing some open problems and directions for future research.

Alternate Query Strategies

As the arbiter governing which samples are chosen for human review, the query strategy can impact detection performance. Ideally, the number of queries should be proportional to the amount of *novel phenomenon* in the data, where novel phenomena is defined with respect to the feature extraction and learning techniques employed in the system. To move towards this goal, alternate query strategies could incorporate density sampling to identify and avoid resubmission of related samples. Given the high dimensionality of data, identification of related samples may require either supervised dimensionality reduction or similarity measures.

Integrated Human Reviewer

In this work we simulated the involvement of a human reviewer by appealing to future labels for a sample. Recognizing that humans are imperfect, we introduced both a false positive and false negative error rate for the human labeler. Although our approach allows for meaningful experimentation, many open questions remain. For example, what is the error rate of humans in practice? To what extent are errors evenly distributed across samples or clustered around specific families of malicious content? What happens if a reviewer acts maliciously, correctly labeling the majority of samples but mislabeling select samples as an attack on the detection platform? Given the considerable expense of human time, future work should also explore the amount of time which analysts require to label samples in different application domains with varying degrees of skill and accuracy. Further work could develop techniques for submitting samples to an analyst along with contextual data designed to speed up the analysis process without misleading the analyst.

Impact of Evasive Activity

Although our case study includes real, timestamped data spanning 30 months, we did not introduce adversarial phenomenon explicitly designed to evade our detection model or otherwise mislead or pollute the review submission process. While prior work has explored evasion of fixed classifiers, evasion of classifiers in periodically retrained systems is less well understood. For example, consider an attacker who modifies an instance to include negatively weighted features and consequently avoids detection. This strategy may allow the attacker to evade a fixed model. (For a more general discussion of adversarial machine learning attacks in a general context, see [39].) This topic demands further investigation. Note however, that our approach may be more robust against this type of attack than a fixed model approach. It may be the case that generally our detection platform selects the modified attack instance for review and retrains on the correctly labeled instance, leading to a modified and resilient model.

Appendix A

Extended Related Work Tables

In this Appendix we present full versions of the prior work tables contained in the main body of the thesis.

- Table A.1 presents a full view of prior work that does not incorporate the progression of time or prioritization of resources.
- Table A.2 presents a full view of prior work moving towards temporal sample consistency.
- Table A.3 presents a full view of prior work adhering to temporal sample consistency.
- Table A.4 presents a full view of prior work exploring prioritization of computational resources.

Author	Year	Domain	Structure	# Benign	# Malicious	Benign Source	Malicious Source
Schultz et al. [109]	2001	x86	5-fold cross-validation	1,001	3,265	System files, other apps	"Various FTP Sites"
Shih et al. [114]	2005	x86	4-fold cross-validation	335	1,436	Personal, public email sources	Personal, public email sources
Reddy et al. [91]	2006	x86	10-fold cross-validation	250	250	System32 folder in Windows	VX Heavens
Masud et al. [61]	2007	x86	3-fold cross-validation	1,967	1,920	System files, popular apps	VX Heavens
Stolfo et al. [120]	2007	PDF	random split, 80% training	362	616	Google	VX Heavens
Ye et al. [144]	2007	x86	10-fold cross-validation	12K	17K	Windows system files	KingSoft corporation
Masud et al. [62]	2008	x86	3-fold cross-validation	1,967	1,920	System files, popular apps	VX Heavens
Ye et al. [142]	2008	x86	10-fold cross-validation	12K	17K	Windows system files	KingSoft corporation
Menahem et al. [64]	2009	x86	10-fold cross-validation	23K	7,690	*unspecified*	*unspecified*
Santos et al. [105]	2009	x86	random split, 66% training	1,000	1,000	Industry partner	Industry partner
Schmidt et al. [108]	2009	Android	10-fold cross-validation	100	240	Android system files	Google search
Shafiq et al. [113]	2009	x86	10-fold cross-validation	1,000	16K	*unspecified*	VX Heavens, Malfease
Tabish et al. [123]	2009	x86	random split, 50 training samples	1,800	10K	Search engines, local network	VX Heavens
Tian et al. [127]	2009	x86	5-fold cross-validation	161	1,206	Windows system files	CA's VET zoo
Ye et al. [145]	2009	x86	random split, 25% training	8K	32K	KingSoft corporation	KingSoft corporation
Rieck et al. [92]	2010	JavaScript	random split (10x), 75% training	220K	609	Alexa, sanitized with blacklist	Wepawet, SQL Injection, Spam trap
Santos et al. [104]	2010	x86	random split, *unspecified*	13K	13K	System files, popular apps	VX Heavens
Alazab et al. [1]	2011	x86	k-fold cross-validation	15K	51K	*unspecified*	Honeynet Project, VX Heavens
Chau et al. [16]	2011	x86	random split, 90% training	†	†	User Machines	User Machines
Curtsinger et al. [23]	2011	JavaScript	random split (5x), 25% training	8K	919	Alexa top 50 URLs	Nozzle detections
Laskov et al. [54]	2011	PDF	2-fold cross-validation	40K	26K	VirusTotal	VirusTotal
Santos et al. [103]	2011	x86	random split, 10-90% training	1,000	1,000	Industry partner	Industry partner
Canali et al. [10]	2012	x86	10-fold cross-validation	‡	7,200	Anubis+Real World Machines	Anubis+Other
Maiorca et al. [59]	2012	PDF	random split, 50% training	9,989	11K	Yahoo Searches	Contagio
Overveldt et al. [81]	2012	ActionScript	random split, 5% training	691	1,184	Misc. sources, manually verified	Wepawet submissions
Sahs et al. [101]	2012	Android	k-fold cross-validation	2,081	91	*unspecified*	*unspecified*
Sanz et al. [107]	2012	Android	10-fold cross-validation	357	249	Google Play	VirusTotal
Tahan et al. [124]	2012	x86	random split (10x), 50% training	2,627	849	Windows XP Program Files	"The Internet"
Dahl et al. [24]	2013	x86	random split, 90% training	817K	1.84M	Microsoft Data	Microsoft Data
Gascon et al. [30]	2013	Android	random split, *unspecified*	136K	12K	Google Play, Other Markets	Google Play, Other Markets
Stringhini et al. [121]	2013	URLs	10-fold cross-validation	510	1,854	Industry partner	Industry partner
Markel et al. [60]	2014	x86	random split, *unspecified*	42K	123K	System files, popular apps	Open Malware

†Evaluation uses approximately 900 million files collected from users between 2007 and 2010, but does not indicate exactly how many are benign or malicious.
‡Evaluation uses 180GB of execution traces from 10 real-world machines, as well as 36 known-benign samples.

Table A.1: Prior work on malicious content detection that does not incorporate progression of time or resource prioritization. *Structure* refers to the type of evaluation used. For entries of the from "random split (Nx), P% training", N (if present) refers to the number of times the trial is repeated and P refers to the percent of data used in training. "k-fold" indicates the number of folds in the cross-validation was not specified.

Author	Year	Domain	Structure	# Benign	# Malicious	Benign Source	Malicious Source
Lee et al. [55]	1999	Network	Family Aware Evaluation	n/a	n/a	Simulated network data	Simulated network data (38 attack types)
Henchiri et al. [37]	2006	x86	Family Aware Evaluation	1,488	1,512	System files, other apps	"Various FTP Sites"
Moskovitch et al. [74]	2007	Network	Family Aware Evaluation	n/a	n/a	Simulated network data	Simulated network data (5 worm families)
Bach et al. [3]	2008	x86	Randomized Sample Order	1971	1651	System files	Virus Heaven, MITRE Corporation
Gavrilut et al. [31]	2009	x86	Separate Datasets	280K	39K	System files, popular apps	Virus Heaven (train); Wildlist (eval.)
Cova et al. [21]	2010	JavaScript	Separate Datasets	11K	823	Sanitized search results, Alexa	Wepawet, SQL Injection, Spam trap
Heiderich et al.† [36]	2011	JavaScript	Separate Datasets	62K	81	Alexa, sanitized with blacklist	malwaredomainlist.com
Nissim et al. [79]	2012	Network	Family Aware Evaluation	n/a	n/a	Simulated network data	Simulated network data (5 worm families)
Nissim et al. [78]	2014	PDF	Randomized Sample Order	5,145	1,629	Web, Ben-Gurion University	VirusTotal, Contagio
Nissim et al. [80]	2014	x86	Randomized Sample Order	22,735	7,688	System files	Virus Heaven

†Heiderich et al. use separate datasets for benign data only and divide malicious data randomly for training and testing.

Table A.2: Prior work on malicious content detection moving towards temporal sample consistency. For work with separate datasets, the benign and malicious sample counts reflect the datasets combined.

Author	Year	Domain	Structure	Duration	Interval	# Benign	# Malicious	Benign Source	Malicious Source
Kolter et al. [49]	2006	x86	Chronological Datasets	n/a	n/a	1,971	291	System files	Virus Heaven
Moskovitch et al. [72, 71]	2008/9	x86	Periodic Retraining	8 Years	1 Year	22,735	7,688	System files	Virus Heaven
Perdesci et al. [86]	2010	Network	Periodic Retraining	6 Months	1 Month	12M	25,720	Sanitized enterprise network	MWCollect, Malfease, Industry Partners
Shabtai et al. [112]	2012	x86	Periodic Retraining	8 Years	1 Year	22,735	7,688	System files	Virus Heaven
Smutz et al. [116]	2012	PDF	Chronological Datasets	n/a	n/a	104,703	5,297	Contagio (train), Live network (eval.)	Contagio(train), Live network (eval.)
Stokes et al. [119]	2012	x86	Periodic Retraining	13 Months	1 Month	???	???	Microsoft Users	Microsoft Users
Srndic et al. [118]	2013	PDF	Periodic Retraining	14 Weeks	1 Week	32,526	407,037	VirusTotal	VirusTotal

Table A.3: Prior work on malicious content detection with temporal sample consistency. *Chronological Datasets* indicates separate training and evaluation datasets where training data entirely predates evaluation data. *Periodic Retraining* indicates a single dataset with timestamped samples divided into intervals where model training proceeds through successive intervals with evaluation occurring on the upcoming interval.

Author	Year	Domain	Prioritization Strategy	# Benign	# Malicious	Benign Source	Malicious Source
Provos et al. [89]	2008	URLs	Weak Detection	1M	25K	Google	Google
Perdisci et al. [85, 84]	2008	x86	Process Optimization	2,900	2,598	System files, Common programs	Malfease Project
Hu et al. [38]	2009	x86	Similarity Detection	0	102,391	n/a	Symantec
Neugschwandtner et al. [77]	2011	x86	Reward Estimation	0	643,212	n/a	Anubis
Canali et al. [11]	2011	URLs	Weak Detection	19M	132K	Search results	Spam feeds
Jacob et al. [41]	2012	x86	Similarity Detection	0	794,665	n/a	Anubis
Chakradeo et al. [15]	2013	Android	Weak Detection	51,407	923	Google Play	Ndoo, Anzhi, Softandroid, Contagio

Table A.4: Prior work exploring prioritization of computational resources in the context of malicious content detection.

Bibliography

[1] Mamoun Alazab et al. “Zero-day Malware Detection Based on Supervised Learning Algorithms of API Call Signatures”. In: *Proceedings of the Ninth Australasian Data Mining Conference - Volume 121*. AusDM ’11. Ballarat, Australia: Australian Computer Society, Inc., 2011, pp. 171–182. ISBN: 978-1-921770-02-9. URL: http://dl.acm.org/citation.cfm?id=2483628.2483648.

[2] Alisa Shevchenko. *The evolution of self-defense technologies in malware*. Tech. rep. Kaspersky Lab, June 2007. URL: https://securelist.com/analysis/publications/36156/the-evolution-of-self-defense-technologies-in-malware/.

[3] Stephen H. Bach and Marcus A. Maloof. “Paired Learners for Concept Drift”. In: *IEEE International Conference on Data Mining (ICDM)*. 2008, pp. 23–32.

[4] Michael Bailey et al. “Automated Classification and Analysis of Internet Malware”. In: *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007, Proceedings*. Ed. by Christopher Krügel, Richard Lippmann, and Andrew J. Clark. Vol. 4637. Lecture Notes in Computer Science. Springer, 2007, pp. 178–197. ISBN: 978-3-540-74319-4. DOI: 10.1007/978-3-540-74320-0_10. URL: http://dx.doi.org/10.1007/978-3-540-74320-0_10.

[5] Marco Barreno et al. “Can Machine Learning Be Secure?” In: *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. ASIACCS ’06. Taipei, Taiwan: ACM, 2006, pp. 16–25. ISBN: 1-59593-272-0. DOI: 10.1145/1128817.1128824. URL: http://doi.acm.org/10.1145/1128817.1128824.

[6] Marco Barreno et al. “Open Problems in the Security of Learning”. In: *Proceedings of the 1st ACM Workshop on Workshop on AISec*. AISec ’08. Alexandria, Virginia, USA: ACM, 2008, pp. 19–26. ISBN: 978-1-60558-291-7. DOI: 10.1145/1456377.1456382. URL: http://doi.acm.org/10.1145/1456377.1456382.

[7] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. “Architecture of a Morphological Malware Detector”. In: *Journal in Computer Virology* 5.3 (2009), pp. 263–270. DOI: 10.1007/s11416-008-0102-4. URL: https://hal.inria.fr/inria-00330022.

[8] Léon Bottou. "On-line Learning in Neural Networks". In: ed. by David Saad. New York, NY, USA: Cambridge University Press, 1998. Chap. On-line Learning and Stochastic Approximations, pp. 9–42. ISBN: 0-521-65263-4. URL: `http://dl.acm.org/citation.cfm?id=304710.304720`.

[9] Burr Settles. *Active Learning Literature Survey*. Tech. rep. Carnegie Mellon University, Jan. 2010.

[10] Davide Canali et al. "A Quantitative Study of Accuracy in System Call-based Malware Detection". In: *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ISSTA 2012. Minneapolis, MN, USA: ACM, 2012, pp. 122–132. ISBN: 978-1-4503-1454-1. DOI: `10.1145/2338965.2336768`. URL: `http://doi.acm.org/10.1145/2338965.2336768`.

[11] Davide Canali et al. "Prophiler: A Fast Filter for the Large-scale Detection of Malicious Web Pages". In: *Proceedings of the 20th International Conference on World Wide Web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 197–206. ISBN: 978-1-4503-0632-4. DOI: `10.1145/1963405.1963436`. URL: `http://doi.acm.org/10.1145/1963405.1963436`.

[12] Windows Dev Center. *Mutex Objects*. [Online; accessed 2015-06-18]. URL: `%5Curl%7Bhttps://msdn.microsoft.com/en-us/library/windows/desktop/ms684266(v=vs.85).aspx%7D`.

[13] Windows Dev Center. *Registry*. [Online; accessed 2015-06-18]. URL: `%5Curl%7Bhttps://msdn.microsoft.com/en-us/library/windows/desktop/ms724871(v=vs.85).aspx%7D`.

[14] Windows Dev Center. *Run-Time Dynamic Linking*. [Online; accessed 2015-06-18]. URL: `%5Curl%7Bhttps://msdn.microsoft.com/en-us/library/windows/desktop/ms685090(v=vs.85).aspx%7D`.

[15] Saurabh Chakradeo et al. "MAST: Triage for Market-scale Mobile Malware Analysis". In: *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. WiSec '13. Budapest, Hungary: ACM, 2013, pp. 13–24. ISBN: 978-1-4503-1998-0. DOI: `10.1145/2462096.2462100`. URL: `http://doi.acm.org/10.1145/2462096.2462100`.

[16] D.H. Chau et al. "Polonium: Tera-Scale Graph Mining and Inference for Malware Detection". In: *SIAM International Conference on Data Mining* (2011).

[17] Yanpei Chen, Archana Ganapathi, and Randy H. Katz. "To Compress or Not to Compress - Compute vs. IO Tradeoffs for Mapreduce Energy Efficiency". In: *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*. Green Networking '10. New Delhi, India: ACM, 2010, pp. 23–28. ISBN: 978-1-4503-0196-1. DOI: `10.1145/1851290.1851296`. URL: `http://doi.acm.org/10.1145/1851290.1851296`.

[18] Mihai Christodorescu et al. "Semantics-Aware Malware Detection". In: *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. SP '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 32–46. ISBN: 0-7695-2339-0. DOI: 10.1109/SP.2005.20. URL: http://dx.doi.org/10.1109/SP.2005.20.

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

[20] Cloudera. *Choosing a Data Compression Format*. [Online; accessed 2015-05-30]. URL: http://www.cloudera.com/content/cloudera/en/documentation/core/v5-3-x/topics/admin_data_compression_performance.html.

[21] Marco Cova, Christopher Kruegel, and Giovanni Vigna. "Detection and Analysis of Drive-by-download Attacks and Malicious JavaScript Code". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 281–290. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772720. URL: http://doi.acm.org/10.1145/1772690.1772720.

[22] Aron Culotta and Andrew McCallum. "Reducing Labeling Effort for Structured Prediction Tasks". In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 2*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 746–751. ISBN: 1-57735-236-x. URL: http://dl.acm.org/citation.cfm?id=1619410.1619452.

[23] Charlie Curtsinger et al. "ZOZZLE: Fast and Precise In-browser JavaScript Malware Detection". In: *Proceedings of the 20th USENIX Conference on Security*. SEC'11. San Francisco, CA: USENIX Association, 2011, pp. 3–3. URL: http://dl.acm.org/citation.cfm?id=2028067.2028070.

[24] George E. Dahl et al. "Large-scale malware classification using random projections and neural networks". In: *ICASSP*. IEEE, 2013, pp. 3422–3426.

[25] Damballa. *State of Infections Report: Q4 2014*. Tech. rep. Damballa, 2015. URL: http://landing.damballa.com/rs/damballa/images/Damballa_Q4_2014_SOI.pdf.

[26] P. Deutsch. *GZIP File Format Specification Version 4.3*. United States, 1996.

[27] James Dougherty, Ron Kohavi, and Mehran Sahami. "Supervised and Unsupervised Discretization of Continuous Features". In: *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*. Ed. by Armand Prieditis and Stuart J. Russell. Morgan Kaufmann, 1995, pp. 194–202. ISBN: 1-55860-377-8.

[28] Facebook. *ThreatExchange*. [Online; accessed 2015-06-05]. URL: %5Curl%7Bhttps://threatexchange.fb.com/%7D.

[29] Apache Foundation. *Apache Parquet*. [Online; accessed 2015-06-05]. URL: %5Curl%7Bhttp://parquet.apache.org%7D.

[30] Hugo Gascon et al. "Structural Detection of Android Malware Using Embedded Call Graphs". In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. AISec '13. Berlin, Germany: ACM, 2013, pp. 45–54. ISBN: 978-1-4503-2488-5. DOI: 10.1145/2517312.2517315. URL: http://doi.acm.org/10.1145/2517312.2517315.

[31] D. Gavrilut et al. "Malware Detection Using Perceptrons and Support Vector Machines". In: *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World:* Nov. 2009, pp. 283–288. DOI: 10.1109/ComputationWorld.2009.85.

[32] Guofei Gu et al. "BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation". In: *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*. SS'07. Boston, MA: USENIX Association, 2007, 12:1–12:16. ISBN: 111-333-5555-77-9. URL: http://dl.acm.org/citation.cfm?id=1362903.1362915.

[33] Guofei Gu et al. "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-independent Botnet Detection". In: *Proceedings of the 17th Conference on Security Symposium*. SS'08. San Jose, CA: USENIX Association, 2008, pp. 139–154. URL: http://dl.acm.org/citation.cfm?id=1496711.1496721.

[34] Steve Hanna et al. "Juxtapp: A Scalable System for Detecting Code Reuse Among Android Applications". In: *Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA'12. Heraklion, Crete, Greece: Springer-Verlag, 2012, pp. 62–81. ISBN: 978-3-642-37299-5. DOI: 10.1007/978-3-642-37300-8_4. URL: http://dx.doi.org/10.1007/978-3-642-37300-8_4.

[35] Phil Harvey. *ExifTool by Phil Harvey*. [Online; accessed 2015-06-17]. URL: %5Curl%7Bhttp://www.sno.phy.queensu.ca/~phil/exiftool/%7D.

[36] Mario Heiderich, Tilman Frosch, and Thorsten Holz. "IceShield: Detection and Mitigation of Malicious Websites with a Frozen DOM". In: *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*. RAID'11. Menlo Park, CA: Springer-Verlag, 2011, pp. 281–300. ISBN: 978-3-642-23643-3. DOI: 10.1007/978-3-642-23644-0_15. URL: http://dx.doi.org/10.1007/978-3-642-23644-0_15.

[37] Olivier Henchiri and Nathalie Japkowicz. "A Feature Selection and Evaluation Scheme for Computer Virus Detection". In: *Proceedings of the Sixth International Conference on Data Mining*. ICDM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 891–895. ISBN: 0-7695-2701-9. DOI: 10.1109/ICDM.2006.4. URL: http://dx.doi.org/10.1109/ICDM.2006.4.

[38] Xin Hu, Tzi-cker Chiueh, and Kang G. Shin. "Large-scale Malware Indexing Using Function-call Graphs". In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: ACM, 2009, pp. 611–620.

ISBN: 978-1-60558-894-0. DOI: 10.1145/1653662.1653736. URL: http://doi.acm.org/10.1145/1653662.1653736.

[39] Ling Huang et al. "Adversarial Machine Learning". In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. AISec '11. Chicago, Illinois, USA: ACM, 2011, pp. 43–58. ISBN: 978-1-4503-1003-1. DOI: 10.1145/2046684.2046692. URL: http://doi.acm.org/10.1145/2046684.2046692.

[40] J.D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science Engineering* 9.3 (May 2007), pp. 90–95. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.55.

[41] Grégoire Jacob et al. "A Static, Packer-agnostic Filter to Detect Similar Malware Samples". In: *Proceedings of the 9th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA'12. Heraklion, Crete, Greece: Springer-Verlag, 2013, pp. 102–122. ISBN: 978-3-642-37299-5. DOI: 10.1007/978-3-642-37300-8_6. URL: http://dx.doi.org/10.1007/978-3-642-37300-8_6.

[42] Jiyong Jang, David Brumley, and Shobha Venkataraman. "BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Chicago, Illinois, USA: ACM, 2011, pp. 309–320. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046742. URL: http://doi.acm.org/10.1145/2046707.2046742.

[43] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2015-05-30]. 2001–. URL: http://www.scipy.org/.

[44] Boojoong Kang et al. "Malware Classification Method via Binary Content Comparison". In: *Proceedings of the 2012 ACM Research in Applied Computation Symposium*. RACS '12. San Antonio, Texas: ACM, 2012, pp. 316–321. ISBN: 978-1-4503-1492-3. DOI: 10.1145/2401603.2401672. URL: http://doi.acm.org/10.1145/2401603.2401672.

[45] Alex Kantchelian et al. "Approaches to Adversarial Drift". In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. AISec '13. Berlin, Germany: ACM, 2013, pp. 99–110. ISBN: 978-1-4503-2488-5. DOI: 10.1145/2517312.2517320. URL: http://doi.acm.org/10.1145/2517312.2517320.

[46] Alex Kantchelian et al. "Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendors Labels". In: *Proceedings of the 2015 ACM Workshop on Artificial Intelligence and Security*. AISec '15. New York, NY, USA: ACM, 2015.

[47] Sandeep Karanth et al. "ZDVUE: Prioritization of Javascript Attacks to Discover New Vulnerabilities". In: *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. AISec '11. Chicago, Illinois, USA: ACM, 2011, pp. 31–42. ISBN: 978-1-4503-1003-1. DOI: 10.1145/2046684.2046690. URL: http://doi.acm.org/10.1145/2046684.2046690.

[48] Abhishek Karnik, Suchandra Goswami, and Ratan K. Guha. “Detecting Obfuscated Viruses Using Cosine Similarity Analysis.” In: *Asia International Conference on Modelling and Simulation*. IEEE Computer Society, May 21, 2007, pp. 165–170. ISBN: 978-0-7695-2845-8. URL: http://dblp.uni-trier.de/db/conf/asiams/ams2007.html#KarnikGG07.

[49] J. Zico Kolter and Marcus A. Maloof. “Learning to Detect and Classify Malicious Executables in the Wild”. In: *J. Mach. Learn. Res.* 7 (Dec. 2006), pp. 2721–2744. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1248547.1248646.

[50] Jeremy Z. Kolter and Marcus A. Maloof. “Learning to Detect Malicious Executables in the Wild”. In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’04. Seattle, WA, USA: ACM, 2004, pp. 470–478. ISBN: 1-58113-888-1. DOI: 10.1145/1014052.1014105. URL: http://doi.acm.org/10.1145/1014052.1014105.

[51] Deguang Kong and Guanhua Yan. “Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification”. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’13. Chicago, Illinois, USA: ACM, 2013, pp. 1357–1365. ISBN: 978-1-4503-2174-7. DOI: 10.1145/2487575.2488219. URL: http://doi.acm.org/10.1145/2487575.2488219.

[52] Tim Kraska et al. “MLbase: A Distributed Machine-learning System”. In: *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013. URL: http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper118.pdf.

[53] Christopher Kruegel et al. “Scalable, Behavior-Based Malware Clustering”. In: *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*. Jan. 2009.

[54] Pavel Laskov and Nedim Šrndić. “Static Detection of Malicious JavaScript-bearing PDF Documents”. In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACSAC ’11. Orlando, Florida, USA: ACM, 2011, pp. 373–382. ISBN: 978-1-4503-0672-0. DOI: 10.1145/2076732.2076785. URL: http://doi.acm.org/10.1145/2076732.2076785.

[55] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. “A Data Mining Framework for Building Intrusion Detection Models.” In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 1999, pp. 120–132. ISBN: 0-7695-0176-1. URL: http://dblp.uni-trier.de/db/conf/sp/sp1999.html#LeeSM99.

[56] David D. Lewis and William A. Gale. “A Sequential Algorithm for Training Text Classifiers”. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’94. Dublin, Ireland:

Springer-Verlag New York, Inc., 1994, pp. 3–12. ISBN: 0-387-19889-X. URL: http://dl.acm.org/citation.cfm?id=188490.188495.

[57] Justin Ma et al. "Identifying suspicious URLs: an application of large-scale online learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*. Ed. by Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman. Vol. 382. ACM International Conference Proceeding Series. ACM, 2009, pp. 681–688. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553462. URL: http://doi.acm.org/10.1145/1553374.1553462.

[58] M.V. Mahoney and P.K. Chan. "Learning rules for anomaly detection of hostile network traffic". In: *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. Nov. 2003, pp. 601–604. DOI: 10.1109/ICDM.2003.1250987.

[59] Davide Maiorca, Giorgio Giacinto, and Igino Corona. "A Pattern Recognition System for Malicious PDF Files Detection". In: *Proceedings of the 8th International Conference on Machine Learning and Data Mining in Pattern Recognition*. MLDM'12. Berlin, Germany: Springer-Verlag, 2012, pp. 510–524. ISBN: 978-3-642-31536-7. DOI: 10.1007/978-3-642-31537-4_40. URL: http://dx.doi.org/10.1007/978-3-642-31537-4_40.

[60] Z. Markel and M. Bilzor. "Building a machine learning classifier for malware detection". In: *Anti-malware Testing Research (WATeR), 2014 Second Workshop on*. Oct. 2014, pp. 1–4. DOI: 10.1109/WATeR.2014.7015757.

[61] M.M. Masud, L. Khan, and B. Thuraisingham. "A Hybrid Model to Detect Malicious Executables". In: *Communications, 2007. ICC '07. IEEE International Conference on*. June 2007, pp. 1443–1448. DOI: 10.1109/ICC.2007.242.

[62] Mohammad M. Masud, Latifur Khan, and Bhavani M. Thuraisingham. "A scalable multi-level feature extraction technique to detect malicious executables." In: *Information Systems Frontiers* 10.1 (Mar. 12, 2008), pp. 33–45. URL: http://dblp.uni-trier.de/db/journals/isf/isf10.html#MasudKT08.

[63] McAfee Labs. *McAfee Labs Threats Report*. Aug. 2014.

[64] Eitan Menahem et al. "Improving Malware Detection by Applying Multi-inducer Ensemble". In: *Comput. Stat. Data Anal.* 53.4 (Feb. 2009), pp. 1483–1494. ISSN: 0167-9473. DOI: 10.1016/j.csda.2008.10.015. URL: http://dx.doi.org/10.1016/j.csda.2008.10.015.

[65] Tony A. Meyer and Brendon Whateley. "SpamBayes: Effective open-source, Bayesian based, email classification system." In: *CEAS*. June 2, 2006. URL: http://dblp.uni-trier.de/db/conf/ceas/ceas2004.html#MeyerW04.

[66] Microsoft. *Microsoft Interflow: a new Security and Threat Information Exchange Platform.* [Online; accessed 2015-06-05]. URL: %5Curl%7Bhttp://blogs.microsoft.com/cybertrust/2014/06/23/microsoft-interflow-a-new-security-and-threat-information-exchange-platform/%7D.

[67] Brad Miller et al. "Adversarial Active Learning". In: *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*. AISec '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 3–14. ISBN: 978-1-4503-3153-1. DOI: 10.1145/2666652.2666656. URL: http://doi.acm.org/10.1145/2666652.2666656.

[68] Brad Miller et al. "I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis". English. In: *Privacy Enhancing Technologies*. Ed. by Emiliano De Cristofaro and StevenJ. Murdoch. Vol. 8555. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 143–163. ISBN: 978-3-319-08505-0. DOI: 10.1007/978-3-319-08506-7_8. URL: http://dx.doi.org/10.1007/978-3-319-08506-7_8.

[69] Aziz Mohaisen and Omar Alrawi. "AMAL: High-Fidelity, Behavior-Based Automated Malware Analysis and Classification". In: *Information Security Applications - 15th International Workshop, WISA 2014, Jeju Island, Korea, August 25-27, 2014. Revised Selected Papers*. Ed. by Kyung Hyune Rhee and Jeong Hyun Yi. Vol. 8909. Lecture Notes in Computer Science. Springer, 2014, pp. 107–121. ISBN: 978-3-319-15086-4. DOI: 10.1007/978-3-319-15087-1_9. URL: http://dx.doi.org/10.1007/978-3-319-15087-1_9.

[70] Morales. *A New Approach to Prioritizing Malware Analysis*. Tech. rep. Software Engineering Institute, 2014. URL: http://blog.sei.cmu.edu/post.cfm/new-approach-prioritizing-malware-analysis-111.

[71] Robert Moskovitch, Clint Feher, and Yuval Elovici. "A Chronological Evaluation of Unknown Malcode Detection." In: *PAISI*. Ed. by Hsinchun Chen et al. Vol. 5477. Lecture Notes in Computer Science. Springer, May 5, 2009, pp. 112–117. ISBN: 978-3-642-01392-8. DOI: 10.1007/978-3-642-01393-5. URL: http://dblp.uni-trier.de/db/conf/paisi/paisi2009.html#MoskovitchFE09.

[72] Robert Moskovitch, Clint Feher, and Yuval Elovici. "Unknown malcode detection - A chronological evaluation." In: *ISI*. IEEE, Jan. 8, 2009, pp. 267–268. ISBN: 978-1-4244-2414-6. URL: http://dblp.uni-trier.de/db/conf/isi/isi2008.html#MoskovitchFE08.

[73] Robert Moskovitch, Nir Nissim, and Yuval Elovici. "Malicious Code Detection Using Active Learning". In: *Privacy, Security, and Trust in KDD, Second ACM SIGKDD International Workshop, PinKDD 2008, Las Vegas, NV, USA, August 24, 2008, Revised Selected Papers*. Ed. by Francesco Bonchi et al. Vol. 5456. Lecture Notes in Computer Science. Springer, 2008, pp. 74–91. ISBN: 978-3-642-01717-9. DOI: 10.1007/978-3-642-01718-6_6. URL: http://dx.doi.org/10.1007/978-3-642-01718-6_6.

[74] Robert Moskovitch et al. "Improving the Detection of Unknown Computer Worms Activity Using Active Learning". In: *Proceedings of the 30th Annual German Conference on Advances in Artificial Intelligence*. KI '07. Osnabr&#252;ck, Germany: Springer-Verlag, 2007, pp. 489–493. ISBN: 978-3-540-74564-8. DOI: 10.1007/978-3-540-74565-5_47. URL: http://dx.doi.org/10.1007/978-3-540-74565-5_47.

[75] Blaine Nelson et al. "Exploiting Machine Learning to Subvert Your Spam Filter". In: *First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '08, San Francisco, CA, USA, April 15, 2008, Proceedings*. Ed. by Fabian Monrose. USENIX Association, 2008. URL: http://www.usenix.org/events/leet08/tech/full_papers/nelson/nelson.pdf.

[76] Blaine Nelson et al. "Near-Optimal Evasion of Convex-Inducing Classifiers". In: *CoRR* abs/1003.2751 (2010). URL: http://arxiv.org/abs/1003.2751.

[77] Matthias Neugschwandtner et al. "FORECAST: Skimming off the Malware Cream". In: *Proceedings of the 27th Annual Computer Security Applications Conference*. ACSAC '11. Orlando, Florida, USA: ACM, 2011, pp. 11–20. ISBN: 978-1-4503-0672-0. DOI: 10.1145/2076732.2076735. URL: http://doi.acm.org/10.1145/2076732.2076735.

[78] Nir Nissim et al. "ALPD: Active Learning Framework for Enhancing the Detection of Malicious PDF Files". In: *IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014, The Hague, The Netherlands, 24-26 September, 2014*. IEEE, 2014, pp. 91–98. ISBN: 978-1-4799-6363-8. DOI: 10.1109/JISIC.2014.23. URL: http://dx.doi.org/10.1109/JISIC.2014.23.

[79] Nir Nissim et al. "Detecting unknown computer worm activity via support vector machines and active learning". In: *Pattern Anal. Appl.* 15.4 (2012), pp. 459–475. DOI: 10.1007/s10044-012-0296-4. URL: http://dx.doi.org/10.1007/s10044-012-0296-4.

[80] Nir Nissim et al. "Novel active learning methods for enhanced PC malware detection in windows OS". In: *Expert Syst. Appl.* 41.13 (2014), pp. 5843–5857. DOI: 10.1016/j.eswa.2014.02.053. URL: http://dx.doi.org/10.1016/j.eswa.2014.02.053.

[81] Timon Van Overveldt, Christopher Kruegel, and Giovanni Vigna. "FlashDetect: ActionScript 3 Malware Detection". In: *Research in Attacks, Intrusions, and Defenses - 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, September 12-14, 2012. Proceedings*. Ed. by Davide Balzarotti, Salvatore J. Stolfo, and Marco Cova. Vol. 7462. Lecture Notes in Computer Science. Springer, 2012, pp. 274–293. ISBN: 978-3-642-33337-8. DOI: 10.1007/978-3-642-33338-5_14. URL: http://dx.doi.org/10.1007/978-3-642-33338-5_14.

[82] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

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

[84] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. "Classification of Packed Executables for Accurate Computer Virus Detection". In: *Pattern Recogn. Lett.* 29.14 (Oct. 2008), pp. 1941–1946. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2008.06.016. URL: http://dx.doi.org/10.1016/j.patrec.2008.06.016.

[85] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. "McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables". In: *Twenty-Fourth Annual Computer Security Applications Conference, ACSAC 2008, Anaheim, California, USA, 8-12 December 2008*. IEEE Computer Society, 2008, pp. 301–310. ISBN: 978-0-7695-3447-3. DOI: 10.1109/ACSAC.2008.22. URL: http://dx.doi.org/10.1109/ACSAC.2008.22.

[86] Roberto Perdisci, Wenke Lee, and Nick Feamster. "Behavioral Clustering of HTTP-based Malware and Signature Generation Using Malicious Network Traces". In: *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. NSDI'10. San Jose, California: USENIX Association, 2010, pp. 26–26. URL: http://dl.acm.org/citation.cfm?id=1855711.1855737.

[87] F. Perez and B.E. Granger. "IPython: A System for Interactive Scientific Computing". In: *Computing in Science Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

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

[89] Niels Provos et al. "All Your iFRAMEs Point to Us". In: *Proceedings of the 17th Conference on Security Symposium*. SS'08. San Jose, CA: USENIX Association, 2008, pp. 1–15. URL: http://dl.acm.org/citation.cfm?id=1496711.1496712.

[90] Moheeb Abu Rajab et al. "CAMP: Content-Agnostic Malware Protection". In: *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. 2013. URL: http://internetsociety.org/doc/camp-content-agnostic-malware-protection.

[91] D. Krishna Sandeep Reddy and Arun K. Pujari. "N-gram analysis for computer virus detection." In: *Journal in Computer Virology* 2.3 (Feb. 7, 2007), pp. 231–239. URL: http://dblp.uni-trier.de/db/journals/virology/virology2.html#ReddyP06.

[92] Konrad Rieck, Tammo Krueger, and Andreas Dewald. "Cujo: Efficient Detection and Prevention of Drive-by-download Attacks". In: *Proceedings of the 26th Annual Computer Security Applications Conference*. ACSAC '10. Austin, Texas, USA: ACM, 2010, pp. 31–39. ISBN: 978-1-4503-0133-6. DOI: 10.1145/1920261.1920267. URL: http://doi.acm.org/10.1145/1920261.1920267.

[93] Konrad Rieck et al. “Automatic Analysis of Malware Behavior Using Machine Learning”. In: *J. Comput. Secur.* 19.4 (Dec. 2011), pp. 639–668. ISSN: 0926-227X. URL: http://dl.acm.org/citation.cfm?id=2011216.2011217.

[94] Konrad Rieck et al. “Learning and Classification of Malware Behavior”. In: *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA ’08. Paris, France: Springer-Verlag, 2008, pp. 108–125. ISBN: 978-3-540-70541-3. DOI: 10.1007/978-3-540-70542-0_6. URL: http://dx.doi.org/10.1007/978-3-540-70542-0_6.

[95] Armin Ronacher. *Flask*. [Online; accessed 2015-06-18]. URL: %5Curl%7Bhttps://pypi.python.org/pypi/Flask%7D.

[96] Nicholas Roy and Andrew McCallum. “Toward Optimal Active Learning Through Sampling Estimation of Error Reduction”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 441–448. ISBN: 1-55860-778-1. URL: http://dl.acm.org/citation.cfm?id=645530.655646.

[97] Paul Royal et al. “PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware”. In: *Proceedings of the 22Nd Annual Computer Security Applications Conference*. ACSAC ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 289–300. ISBN: 0-7695-2716-7. DOI: 10.1109/ACSAC.2006.38. URL: http://dx.doi.org/10.1109/ACSAC.2006.38.

[98] Benjamin I. P. Rubinstein et al. “Evading Anomaly Detection through Variance Injection Attacks on PCA”. In: *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*. Ed. by Richard Lippmann, Engin Kirda, and Ari Trachtenberg. Vol. 5230. Lecture Notes in Computer Science. Springer, 2008, pp. 394–395. ISBN: 978-3-540-87402-7. DOI: 10.1007/978-3-540-87403-4_23. URL: http://dx.doi.org/10.1007/978-3-540-87403-4_23.

[99] Benjamin I.P. Rubinstein et al. “ANTIDOTE: Understanding and Defending Against Poisoning of Anomaly Detectors”. In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. IMC ’09. Chicago, Illinois, USA: ACM, 2009, pp. 1–14. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644895. URL: http://doi.acm.org/10.1145/1644893.1644895.

[100] Benjamin I.P. Rubinstein et al. “Stealthy Poisoning Attacks on PCA-based Anomaly Detectors”. In: *SIGMETRICS Perform. Eval. Rev.* 37.2 (Oct. 2009), pp. 73–74. ISSN: 0163-5999. DOI: 10.1145/1639562.1639592. URL: http://doi.acm.org/10.1145/1639562.1639592.

[101] J. Sahs and L. Khan. “A Machine Learning Approach to Android Malware Detection”. In: *Intelligence and Security Informatics Conference (EISIC), 2012 European*. Aug. 2012, pp. 141–147. DOI: 10.1109/EISIC.2012.34.

[102] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval.* New York, NY, USA: McGraw-Hill, Inc., 1986. ISBN: 0070544840.

[103] Igor Santos, Javier Nieves, and Pablo Garcia Bringas. "Semi-supervised Learning for Unknown Malware Detection". In: *International Symposium on Distributed Computing and Artificial Intelligence, DCAI 2011, Salamanca, Spain, 6-8 April 2011*. Ed. by Ajith Abraham et al. Vol. 91. Advances in Soft Computing. Springer, 2011, pp. 415–422. ISBN: 978-3-642-19933-2. DOI: 10.1007/978-3-642-19934-9_53. URL: http://dx.doi.org/10.1007/978-3-642-19934-9_53.

[104] Igor Santos et al. "Idea: Opcode-Sequence-Based Malware Detection". In: *Engineering Secure Software and Systems, Second International Symposium, ESSoS 2010, Pisa, Italy, February 3-4, 2010. Proceedings*. Ed. by Fabio Massacci, Dan S. Wallach, and Nicola Zannone. Vol. 5965. Lecture Notes in Computer Science. Springer, 2010, pp. 35–43. ISBN: 978-3-642-11746-6. DOI: 10.1007/978-3-642-11747-3_3. URL: http://dx.doi.org/10.1007/978-3-642-11747-3_3.

[105] Igor Santos et al. "N-grams-based File Signatures for Malware Detection". In: *ICEIS 2009 - Proceedings of the 11th International Conference on Enterprise Information Systems, Volume AIDSS, Milan, Italy, May 6-10, 2009*. Ed. by José Cordeiro and Joaquim Filipe. 2009, pp. 317–320. ISBN: 978-989-8111-85-2.

[106] Borja Sanz et al. "On the automatic categorisation of android applications." In: *CCNC*. IEEE, 2012, pp. 149–153. ISBN: 978-1-4577-2070-3. URL: http://dblp.uni-trier.de/db/conf/ccnc/ccnc2012.html#SanzSLUB12.

[107] Borja Sanz et al. "PUMA: Permission Usage to Detect Malware in Android". In: *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions, Ostrava, Czech Republic, September 5th-7th, 2012*. Ed. by Álvaro Herrero et al. Vol. 189. Advances in Intelligent Systems and Computing. Springer, 2012, pp. 289–298. ISBN: 978-3-642-33017-9. DOI: 10.1007/978-3-642-33018-6_30. URL: http://dx.doi.org/10.1007/978-3-642-33018-6_30.

[108] Aubrey-Derrick Schmidt et al. "Static Analysis of Executables for Collaborative Malware Detection on Android". In: *Proceedings of the 2009 IEEE International Conference on Communications*. ICC'09. Dresden, Germany: IEEE Press, 2009, pp. 631–635. ISBN: 978-1-4244-3434-3. URL: http://dl.acm.org/citation.cfm?id=1817271.1817389.

[109] Matthew G. Schultz et al. "Data Mining Methods for Detection of New Malicious Executables". In: *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. SP '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 38–. URL: http://dl.acm.org/citation.cfm?id=882495.884439.

[110] Guido Schwenk et al. "Autonomous Learning for Detection of JavaScript Attacks: Vision or Reality?" In: *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. AISec '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 93–104. ISBN: 978-1-4503-1664-4. DOI: 10.1145/2381896.2381911. URL: http://doi.acm.org/10.1145/2381896.2381911.

[111] D. Sculley et al. "Detecting Adversarial Advertisements in the Wild". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: ACM, 2011, pp. 274–282. ISBN: 978-1-4503-0813-7. DOI: 10.1145/2020408.2020455. URL: http://doi.acm.org/10.1145/2020408.2020455.

[112] Asaf Shabtai et al. "Detecting unknown malicious code by applying classification techniques on OpCode patterns". English. In: *Security Informatics* 1.1, 1 (2012). DOI: 10.1186/2190-8532-1-1. URL: http://dx.doi.org/10.1186/2190-8532-1-1.

[113] M. Zubair Shafiq et al. "PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime". In: *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*. RAID '09. Saint-Malo, France: Springer-Verlag, 2009, pp. 121–141. ISBN: 978-3-642-04341-3. DOI: 10.1007/978-3-642-04342-0_7. URL: http://dx.doi.org/10.1007/978-3-642-04342-0_7.

[114] Dong-Her Shih, Hsiu-Sen Chiang, and C. David Yen. "Classification methods in the detection of new malicious emails". In: *Information Sciences* 172 (2005), pp. 241–261. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2004.06.003. URL: http://www.sciencedirect.com/science/article/pii/S002002550400180X.

[115] Konstantin Shvachko et al. "The Hadoop Distributed File System". In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. ISBN: 978-1-4244-7152-2. DOI: 10.1109/MSST.2010.5496972. URL: http://dx.doi.org/10.1109/MSST.2010.5496972.

[116] Charles Smutz and Angelos Stavrou. "Malicious PDF Detection Using Metadata and Structural Features". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACSAC '12. Orlando, Florida, USA: ACM, 2012, pp. 239–248. ISBN: 978-1-4503-1312-4. DOI: 10.1145/2420950.2420987. URL: http://doi.acm.org/10.1145/2420950.2420987.

[117] Evan R. Sparks et al. "MLI: An API for Distributed Machine Learning". In: *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*. Ed. by Hui Xiong et al. IEEE Computer Society, 2013, pp. 1187–1192. ISBN: 978-0-7695-5108-1. DOI: 10.1109/ICDM.2013.158. URL: http://dx.doi.org/10.1109/ICDM.2013.158.

[118] Nedim Srndic and Pavel Laskov. "Detection of Malicious PDF Files Based on Hierarchical Document Structure". In: *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. 2013. URL: http://internetsociety.org/doc/detection-malicious-pdf-files-based-hierarchical-document-structure.

[119] Jack W. Stokes et al. "Scalable Telemetry Classification for Automated Malware Detection." In: *ESORICS*. Ed. by Sara Foresti, Moti Yung, and Fabio Martinelli. Vol. 7459. Lecture Notes in Computer Science. Springer, 2012, pp. 788–805. ISBN: 978-3-642-33166-4. DOI: 10.1007/978-3-642-33167-1. URL: http://dblp.uni-trier.de/db/conf/esorics/esorics2012.html#StokesPWFKMTG12.

[120] SalvatoreJ. Stolfo, Ke Wang, and Wei-Jen Li. "Towards Stealthy Malware Detection". English. In: *Malware Detection*. Ed. by Mihai Christodorescu et al. Vol. 27. Advances in Information Security. Springer US, 2007, pp. 231–249. ISBN: 978-0-387-32720-4. DOI: 10.1007/978-0-387-44599-1_11. URL: http://dx.doi.org/10.1007/978-0-387-44599-1_11.

[121] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. "Shady Paths: Leveraging Surfing Crowds to Detect Malicious Web Pages". In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 133–144. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516682. URL: http://doi.acm.org/10.1145/2508859.2516682.

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

[123] S. Momina Tabish, M. Zubair Shafiq, and Muddassar Farooq. "Malware Detection Using Statistical Analysis of Byte-level File Content". In: *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*. CSI-KDD '09. Paris, France: ACM, 2009, pp. 23–31. ISBN: 978-1-60558-669-4. DOI: 10.1145/1599272.1599278. URL: http://doi.acm.org/10.1145/1599272.1599278.

[124] Gil Tahan, Lior Rokach, and Yuval Shahar. "Mal-ID: Automatic Malware Detection Using Common Segment Analysis and Meta-features". In: *J. Mach. Learn. Res.* 13 (Apr. 2012), pp. 949–979. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2343677.

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

[126] K. Thomas et al. "Design and Evaluation of a Real-Time URL Spam Filtering Service". In: *Security and Privacy (SP), 2011 IEEE Symposium on*. May 2011, pp. 447–462. DOI: 10.1109/SP.2011.25.

[127] Ronghua Tian et al. "An automated classification system based on the strings of trojan and virus families". In: *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. Oct. 2009, pp. 23–30. DOI: 10.1109/MALWARE.2009.5403021.

[128] Simon Tong and Daphne Koller. “Support Vector Machine Active Learning with Applications to Text Classification”. In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 45–66. ISSN: 1532-4435. DOI: 10.1162/153244302760185243. URL: http://dx.doi.org/10.1162/153244302760185243.

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

[130] Chih-Fong Tsai et al. “Intrusion detection by machine learning: A review”. In: *Expert Syst. Appl.* (2009), pp. 11994–12000.

[131] Cristina Vatamanu, Dragos Gavrilut, and Razvan Benchea. “A practical approach on clustering malicious PDF documents.” In: *Journal in Computer Virology* 8.4 (2012), pp. 151–163. URL: http://dblp.uni-trier.de/db/journals/virology/virology8.html#VatamanuGB12.

[132] Deepak Venugopal and Guoning Hu. “Efficient Signature Based Malware Detection on Mobile Devices”. In: *Mob. Inf. Syst.* 4.1 (Jan. 2008), pp. 33–49. ISSN: 1574-017X. URL: http://dl.acm.org/citation.cfm?id=1376830.1376833.

[133] VirusTotal. *File Statistics During Last 7 Days*. https://www.virustotal.com/en/statistics/. Retrieved on July 30, 2014.

[134] VMWare. *VMWare vSphere*. June 2015.

[135] S. van der Walt, S.C. Colbert, and G. Varoquaux. “The NumPy Array: A Structure for Efficient Numerical Computation”. In: *Computing in Science Engineering* 13.2 (Mar. 2011), pp. 22–30. ISSN: 1521-9615. DOI: 10.1109/MCSE.2011.37.

[136] Kilian Weinberger et al. “Feature Hashing for Large Scale Multitask Learning”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML ’09. Montreal, Quebec, Canada: ACM, 2009, pp. 1113–1120. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553516. URL: http://doi.acm.org/10.1145/1553374.1553516.

[137] Tom White. *Hadoop: The Definitive Guide*. 1st. O’Reilly Media, Inc., 2009. ISBN: 0596521979, 9780596521974.

[138] Georg Wicherski. “peHash: A Novel Approach to Fast Malware Clustering”. In: *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*. LEET’09. Boston, MA: USENIX Association, 2009, pp. 1–1. URL: http://dl.acm.org/citation.cfm?id=1855676.1855677.

[139] Wikipedia. *List of file signatures*. [Online; accessed 2015-06-17]. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/List_of_file_signatures%7D.

[140] Guanhua Yan, Nathan Brown, and Deguang Kong. "Exploring Discriminatory Features for Automated Malware Classification". In: *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA'13. Berlin, Germany: Springer-Verlag, 2013, pp. 41–61. ISBN: 978-3-642-39234-4. DOI: 10.1007/978-3-642-39235-1_3. URL: http://dx.doi.org/10.1007/978-3-642-39235-1_3.

[141] Chao Yang, Robert Chandler Harkreader, and Guofei Gu. "Die Free or Live Hard? Empirical Evaluation and New Design for Fighting Evolving Twitter Spammers". In: *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*. RAID'11. Menlo Park, CA: Springer-Verlag, 2011, pp. 318–337. ISBN: 978-3-642-23643-3. DOI: 10.1007/978-3-642-23644-0_17. URL: http://dx.doi.org/10.1007/978-3-642-23644-0_17.

[142] Yanfang Ye et al. "An intelligent PE-malware detection system based on association mining". In: *Journal in Computer Virology* 4.4 (2008), pp. 323–334. DOI: 10.1007/s11416-008-0082-4. URL: http://dx.doi.org/10.1007/s11416-008-0082-4.

[143] Yanfang Ye et al. "Automatic malware categorization using cluster ensemble". In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. Ed. by Bharat Rao et al. ACM, 2010, pp. 95–104. ISBN: 978-1-4503-0055-1. DOI: 10.1145/1835804.1835820. URL: http://doi.acm.org/10.1145/1835804.1835820.

[144] Yanfang Ye et al. "IMDS: Intelligent Malware Detection System". In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '07. San Jose, California, USA: ACM, 2007, pp. 1043–1047. ISBN: 978-1-59593-609-7. DOI: 10.1145/1281192.1281308. URL: http://doi.acm.org/10.1145/1281192.1281308.

[145] Yanfang Ye et al. "SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging". In: *Journal in Computer Virology* 5.4 (2009), pp. 283–293. DOI: 10.1007/s11416-008-0108-y. URL: http://dx.doi.org/10.1007/s11416-008-0108-y.

[146] George Yiu. "Building a User Interface for a Temporal Machine Learning System". MA thesis. EECS Department, University of California, Berkeley, May 2015. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-156.html.

[147] Matei Zaharia et al. "Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing". In: *Proceedings of USENIX Conference on Networked Systems Design and Implementation*. San Jose, CA: USENIX Association, 2012.

[148] Zhen Tang, Jeff Schneider. *Analysis of Prioritizing Malware*. Tech. rep. Carnegie Mellon University, Feb. 2014. URL: http://www.ml.cmu.edu/research/dap-papers/tang.pdf.