2008 IEEE Symposium on Security and Privacy

Casting out Demons: Sanitizing Training Data for Anomaly Sensors

Gabriela F. Cretu[1] Angelos Stavrou[2*] Michael E. Locasto[3*] Salvatore J. Stolfo[1]
Angelos D. Keromytis[1]
[1]Department of Computer Science, Columbia University
{gcretu, sal, angelos}@cs.columbia.edu
[2]Department of Computer Science, George Mason University
astavrou@gmu.edu
[3]Institute for Security Technology Studies, Dartmouth College
locasto@cs.dartmouth.edu

Abstract

The efficacy of Anomaly Detection (AD) sensors depends heavily on the quality of the data used to train them. Artificial or contrived training data may not provide a realistic view of the deployment environment. Most realistic data sets are dirty; that is, they contain a number of attacks or anomalous events. The size of these high-quality training data sets makes manual removal or labeling of attack data infeasible. As a result, sensors trained on this data can miss attacks and their variations. We propose extending the training phase of AD sensors (in a manner agnostic to the underlying AD algorithm) to include a sanitization phase.

This phase generates multiple models conditioned on small slices of the training data. We use these "micro-models" to produce provisional labels for each training input, and we combine the micro-models in a voting scheme to determine which parts of the training data may represent attacks. Our results suggest that this phase automatically and significantly *improves the quality of unlabeled training data by making it as "attack-free" and "regular" as possible in the absence of absolute ground truth. We also show how a collaborative approach that combines models from different networks or domains can further refine the sanitization process to thwart targeted training or mimicry attacks against a single site.*

1 Introduction

Anomaly-based classification provides a powerful method of detecting inputs and behavior that are potentially malicious without relying on a static set of signatures or a potentially incomplete behavioral specification. Anomaly sensors include those that classify both network traffic content [13, 29] and sequences of system calls [22]. Although anomaly–based approaches are not perfect [9, 26, 27], recent research indicates that signature–based network intrusion detection systems are quickly becoming ineffective at identifying malicious traffic [4, 17, 23]. In particular, Song *et al.* [23] demonstrate the relative ease with which polymorphic attack engines can overwhelm signature–based detection methods. They conclude that modeling normal behavior or content represents one of a small set of promising alternatives[1]. In short, relying on anomaly detection (AD) sensors to discover 0-day attacks has become a necessity rather than an option.

1.1 Obtaining Clean Data Sets

Current evidence seems to indicate that improving AD sensors merits further attention. Effective anomaly detection, however, requires highly accurate modeling of normal traffic — a process that remains an open problem. In particular, Taylor and Gates [26] point to the problem of polluted or unclean training data sets as a key roadblock to the construction of effective AD sensors. Specifically, "ground truth" for large, realistic data sets is extremely hard to determine.

In a related problem, the intrusion detection community lacks a collection of significant, real-world data sets to test and validate new intrusion detection algorithms. Although an effort to assemble such a collection was made almost a decade ago [14], the resulting data set was flawed in a

*The work described in this paper was performed while these co-authors were at Columbia University.

[1]Their claim rests on the assumption that the set of good input or behaviors is much more constrained than *all* possible bad input or behaviors for realistic applications. We find this argument reasonable, especially since application developers do not operate in an adversarial way: they do not purposefully allow their software to accept widely differing sets of strings.

978-0-7695-3168-7 /08 $25.00 © 2008 IEEE
DOI 10.1109/SP.2008.11

number of ways [15], and there is a growing consensus that future experimental results based on this data set should be ignored. The community, however, is left without any acceptable replacement. As a result, researchers and customers cannot validate the work of other researchers or vendors, especially since placing real, large data sets into wide circulation may reveal sensitive information belonging to the organization kind enough to donate the data. The next best solution involves every organization maintaining a private extensive data collection. Laying aside the challenges involved in addressing the privacy concerns of individuals within the organization, the technical challenge of keeping this data set pristine is currently an open problem.

1.2 Contributions

Creating a robust method of sanitizing data sets seems to be a key challenge for these two complementary open problems. The large data sets required for both typically contain an unpredictable spread of attacks, rare data items, and artifacts of misconfigurations or other errors. The potential size and complexity of these data sets makes manual labeling or removal of attacks a futile exercise.

In this paper, we propose a novel method for sanitizing such data sets to help address these problems. For concreteness, our implementation and experiments focus on the problem of cleaning training data sets of AD sensors. Our work provides a number of contributions:

- We extend the training phase of anomaly sensors with a new sanitization phase that uses our novel micro-models in a voting scheme to eliminate attacks and anomalies from training data
- We built a system to implement our algorithms, and we applied it to training data for two anomaly sensors drawn from the research literature
- We extend the sanitization phase to a novel distributed architecture in order to cross-sanitize the models and remove long-lasting attacks that might otherwise bypass the local sanitization process
- We identify the *false* false positive problem and propose a shadow sensor architecture for consuming false positives (FP) with an automated process rather than human attention

While we [5] have explored the basic problem of similar sanitization techniques for single sites, our distributed strategy (see Section 1.5) provides a major advance over that work. In addition, we now conduct a far more thorough analysis and experimental evaluation of data sanitization techniques.

1.3 Technical Challenges

Ideally, an anomaly detector should achieve 100% detection accuracy, *i.e.,* true attacks are all identified, with 0% false positives. Reaching this ideal is very hard due to a number of problems. First, the generated model can under-fit the actual normal traffic. Under-fitting means that the AD sensor is overly general: it will flag traffic as "normal" even if this traffic does not belong to the true normal model. As a result, attackers have sufficient space to disguise their exploit, thus increasing the amount of "false negatives" produced by the sensor. Second, and equally as troubling, the model of normal traffic can over-fit the training data: non-attack traffic that is not observed during training may be regarded as anomalous. Over-fitting can generate an excessive amount of false alerts or "false positives." Third, unsupervised AD systems often lack a measure of ground truth to compare to and verify against. The presence of an attack in the training data "poisons" the normal model, thus rendering the AD system incapable of detecting future or closely related instances of this attack. As a result, the AD system may produce false negatives. This risk becomes a limiting factor of the size of the training set [25]. Finally, even in the presence of ground truth, creating a single model of normal traffic that includes all non-attack traffic can result in under-fitting and over generalization.

1.4 Solution Outline

These problems appear to stem from a common source: the quality of the normality model that an AD system employs to detect abnormal traffic. This single, monolithic normality model is the product of a training phase that traditionally uses all the traffic from a non-sanitized training data set. Our goal in this paper is to extend the AD training phase to successfully sanitize training data by removing both attacks and non-regular traffic, thereby computing a more accurate anomaly detection model that achieves both a high rate of detection and a low rate of false positives.

To that end, we generalize the notion of training for an AD system. Instead of using a normal model generated by a single AD sensor trained on a single large set of data, we use multiple AD instances trained on small data slices. This process produces multiple normal models, which we call *micro-models*, by training AD instances on small, disjoint subsets of the original traffic dataset. Each of these micro-models represents a very localized view of the training data. Armed with the micro-models, we are now in a position to assess the quality of our training data and automatically detect and remove any attacks or abnormalities that should not be considered part of the normal model.

The intuition behind our approach is based on the observation that in a training set spanning a sufficiently large

time interval, an attack or an abnormality will appear only in small and relatively confined time intervals. To identify these abnormalities, we test each packet of the training data set against the produced micro-models. Using a voting scheme, we can determine which packets to consider abnormal and remove from our training set. In our analysis, we explore the efficiency and tradeoffs of both majority voting and weighted voting schemes. The result of our approach is a training set which contains packets that are closer to what we consider the "normal model" of the application's I/O streams.

This sanitized training set enables us to generate a single *sanitized model* from a single AD instance. This model is very likely free of both attacks and abnormalities. As a result, the detection performance during the testing phase should improve. We establish evidence for this conjecture in the experiments of Section 3, which show a 5-fold increase of the average detection rate. Furthermore, data that was deemed abnormal in the voting strategy is used for building a different model, which we call the *abnormal model*. This model is intended to represent traffic that contains attacks or any data that is not commonly seen during a normal execution of the protected system.

1.5 Distributed Sanitization

Our initial assumptions do not hold when the training set contains persistent and/or targeted attacks, or there exist other anomalies that persist throughout the majority of the training set. To defend against such attacks, we propose a novel, fully distributed collaborative sanitization strategy. This strategy leverages the location diversity of collaborating sites to exchange information related to abnormal data that can be used to clean each site's training data set.

Consequently, our work introduces a two-phase training process: initially, we compute the AD models of "normal" and "abnormal" locally from the training set at each site. In the second phase, we distribute the "abnormal" models between sites, and we use this information to re-evaluate and filter the local training data set. If data deemed normal by the local micro-models happens to belong to a remote "abnormal" model, we inspect or redirect this data to an oracle. Even if the identities of the collaborating sites become known, attacking all the sites with targeted or blending attacks is a challenging task. The attacker will have to generate mimicry attacks against all collaborators and blend the attack traffic using the individual sites' normal data models.

1.6 Evaluation Scenarios

Our evaluation considers two different defense configurations involving AD sensors. In the first case, we measure the increase in detection performance for a simple AD-based defense system when we use the new training phase to sanitize the training set. As a second scenario, we assume that a latency-expensive oracle can help classify "suspect data" and differentiate between false positives (FP) and true positives (TP). In practice, our oracle consists of a heavily instrumented host-based "shadow" server system (similar to strategies proposed by [1, 20]) that determines with very high accuracy whether a packet contains an attack. By diverting all suspect data to this oracle, we can identify true attacks by observing whether the shadow sensor emits an alert after consuming suspicious data. This high accuracy, however, comes at the cost of greatly increased computational effort making the redirection of all traffic to the shadow sensors unfeasible.

Many papers comment on anomaly detectors having too high a false positive rate, thus making them less than ideal sensors. In light of the above scenario, we see such comments as the "*false* false positive problem," as our shadow sensor architecture allows an automated process (instead of a human operator) to consume and vet FPs. We use this scenario to demonstrate that failure to substantially reduce the FP rate of a network AD sensor does not render the sensor useless. By using a host-based shadow sensor, false positives neither damage the system under protection nor flood an operational center with alarms. Instead, the shadow sensor processes both true attacks and incorrectly classified packets to validate whether a packet signifies a true attack. These packets are still processed by the shadowed application and only cause an increased delay for network traffic incorrectly deemed an attack.

2 Local Sanitization

In order to generate an accurate and precise normal model, researchers must utilize an effective sanitization process for the AD training data set. To that end, removing all abnormalities, including attacks and other traffic artifacts, from the AD training set is a crucial first step. Supervised training using labeled datasets appears to be an ideal cleaning process. However, the size and complexity of training data sets obtained from real-world network traces makes such labeling infeasible. In addition, semi–supervised or even unsupervised training using an automated process or an oracle is computationally demanding and may lead to an over-estimated and under-trained normal model. Indeed, even if we assume that unsupervised training can detect 100% of the attacks, the resulting normal model may contain abnormalities that should not be considered part of the normal model.

These abnormalities represent data patterns or traffic that are not attacks, but still appear infrequently or for a very short period of time. For example, the random portion of HTTP cookies and HTTP POST requests may be con-

sidered non-regular and thus abnormal. This type of data should not form part of the normal model because it does not convey any extra information about the site or modeled protocol. Thus, in practice, both supervised and unsupervised training might fail to identify and remove from the training set non-regular data, thereby producing a large and over-estimated normal model. We introduce a new unsupervised training approach that attempts to determine both attacks and abnormalities and separate them from the regular, normal model.

2.1 Assumptions

We observe that for a training set that spans a long period of time, attacks and abnormalities are a minority class of data. While the total attack volume in any given trace may be high, the frequency of specific attacks is generally low relative to legitimate input. This assumption may not hold in some circumstances, *e.g.,* during a DDoS attack or during the propagation phase of a worm such as Slammer. We can possibly identify such non-ideal AD training conditions by analyzing the entropy of a particular dataset (too high or too low may indicate exceptional circumstances). We leave this analysis for the future. Furthermore, although we cannot predict the time of an attack in the training set, the attack itself will manifest as a few packets that will not persist throughout the dataset. Common attack packets tend to cluster together and form a sparse representation over time. For example, once a worm outbreak starts, it appears concentrated in a relatively short period of time, and eventually system defenders quarantine, patch, reboot, or filter the infected hosts. As a result, the worm's appearance in the dataset decreases [16]. We expect these assumptions to hold true over relatively long periods of time, and this expectation requires the use of large training datasets to properly sanitize an AD model. In short, larger amounts of training data can help produce better models — a supposition that seems intuitively reasonable.

We must be cautious, however, as having a large training set increases the probability that an individual datum appears normal (the datum appears more frequently in the dataset; consequently, the probability of it appearing "normal" increases). Furthermore, having the AD system consider greater amounts of training data increases the probability of malcode presence in the dataset. As a result, malcode data can poison the model, and its presence complicates the task of classifying normal data. We next describe how we use micro-models in an ensemble arrangement to process large training data sets in a manner that resists the effects of malcode content in that data.

2.2 Micro-models

Our method of sanitizing the training data for an AD sensor employs the idea of "ensemble methods." Dietterich [7] defines an ensemble classifier as "a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples." Methods for creating ensembles include, among other actions, techniques that manipulate the training examples. Given our assumption about the span of attacks in our training set, it seems appropriate to use time-delimited slices of the training data.

We employ the following strategy: consider a large training dataset *T* partitioned into a number of smaller disjoint subsets (micro-datasets): $T = \{md_1, md_2, \ldots, md_N\}$ where md_i is the micro-dataset starting at time $(i-1) * g$ and, *g* is the granularity for each micro-dataset. We define the model function *AD*: $M = AD(T)$ where *AD* can be any chosen anomaly detection algorithm, *T* is the training dataset, and *M* denotes the model produced by *AD*.

In order to create the ensemble of classifiers, we use each of the "epochs" md_i to compute a *micro-model*, M_i. $M_i = AD(md_i)$. We posit that each distinct attack will be concentrated in (or around) time period t_j affecting only a small fraction of the micro-models: M_j may be poisoned, having modeled the attack vector as normal data, but model M_k computed for time period t_k, $k \neq j$ is likely to be unaffected by the same attack. In order to maximize this likelihood, however, we need to identify the right level of time granularity *g*. Naturally, epochs can range over the entire set of training data. Our experiments, reported in Section 3, analyze network packet traces captured over approximately 500 hours. We find that a value of *g* from 3 to 5 hours was sufficient to generate well behaved micro-models.

2.3 Sanitized and Abnormal Models

After generating the micro-models, we compute a new AD model using the set of previously computed micro-models. In this second phase, we produce a sanitized normal model using either the training set used to produce the micro-models or a second set of training data. Splitting the training data set into two parts represents the worst case scenario, because it assumes that we are not able to store the large dataset necessary to build the micro-models. Hence, the AD sensor is required to generate the micro-models online using a fraction of the necessary space (the models are far smaller than the raw traffic). Then, we can sanitize the training dataset by (online or offline) testing using all the pre-computed micro-models M_i. Each test results in a new labeled data set with every packet P_j labeled as *normal* or *abnormal*:

$$L_{j,i} = TEST(P_j, M_i) \quad (1)$$

where the label, $L_{j,i}$, has a value of 0 if the model M_i deems the packet P_j normal, or 1 if M_i deems it abnormal.

However, these labels are not yet generalized; they remain specialized to the micro-model used in each test. In order to generalize the labels, we process each labeled dataset through a voting scheme, which assigns a final score to each packet:

$$SCORE(P_j) = \frac{1}{W} \sum_{i=1}^{N} w_i \cdot L_{j,i} \quad (2)$$

where w_i is the weight assigned to model M_i and $W = \sum_{i=1}^{N} w_i$. We have investigated two possible strategies: *simple voting*, where all models are weighted identically, and *weighted voting*, which assigns to each micro-model M_i a weight w_i equal to the number of packets used to train it. The study of other weighting strategies can provide an avenue for future research.

To understand the AD decision process, we consider the case where a micro-model M_i includes attack-related content. When used for testing, the AD may label as normal a packet containing that particular attack vector. Assuming that only a minority of the micro-models will include the same attack vector as M_i, we use the voting scheme to split our data into two disjoint sets: one that contains only majority-voted normal packets, T_{san} from which we build the sanitized model M_{san}, and the rest, used to generate a model of abnormal data, M_{abn}.

$$T_{san} = \bigcup \{P_j \mid SCORE(P_j) \leq V\},\ M_{san} = AD(T_{san})$$

$$T_{abn} = \bigcup \{P_j \mid SCORE(P_j) > V\},\ M_{abn} = AD(T_{abn})$$

where *V* is a voting threshold. In the case of unweighted voting, *V* is the maximum percentage of abnormal labels permitted such that a packet is labeled normal. Consequently, it must be the case that $1 - V > N_p$, where N_p is the maximum percentage of models expected to be poisoned by any specific attack vector. We provide an analysis of the impact of this threshold on both voting schemes in Section 3.

After this two-phase training process, the AD sensor can use the sanitized model for online testing. Note that we have described a general approach to sanitization without resorting to the specific details of the AD decision process; it is enough that the AD sensor outputs a classification for each discrete piece of its input (*e.g.*, a network packet or message). Consequently, we believe that our approach can help generate sanitized models for a wide range of anomaly detection systems. In the remainder of this paper, we evaluate our approach on two anomaly sensors drawn from the research literature; we are in the process of evaluating two others (pH and `libanomaly`).

3 Evaluation of Sanitization

In this section, we quantify the increase in the detection accuracy of any content-based AD system when we apply training data sanitization. We treat the AD sensor as a black box to avoid using optimizations that are specific to a particular AD system. In the following experiments, we use two anomaly sensors: Anagram [29] and Payl [28,30]. Both sensors are n-gram content-based anomaly detectors for network packets. Although they both use an n-gram approach, these sensors have very different learning algorithms. The details of these algorithms are beyond the scope of this paper. We refer the interested reader to the citations above.

We evaluate our approach using two different scenarios. In the first scenario, we measure the performance of the sensor with and without sanitization. Additionally, we use the sensor as a packet classifier for incoming network traffic: we test each packet and consider the computational costs involved in diverting each alert to a host-based shadow sensor. Both the feasibility and scalability of this scenario depend mainly on the amount of alerts generated by the AD sensor, since all "suspect-data" (data that causes the sensor to generate an alert) are significantly delayed by the shadow sensor.

Our experimental corpus consists of 500 hours of real network traffic, which contains approximately four million content packets. We collected the traffic from three different hosts: *www*, *www1*, and *lists*. We partitioned this data into three separate sets: two used for training and one used for testing. We use the first 300 hours of traffic to build the micro-models and the next 100 hours to generate the sanitized model.

The remaining 100 hours of data was used for testing. It consists of approximately $775,000$ packets (with 99 attack packets) for *www1*, $656,000$ packets (with 70 attack packets) for *www*, and $26,000$ packets (with 81 attack packets) for *lists*. Given that *www1* exhibits a larger volume of traffic, we chose to perform a more in-depth analysis on its traffic. In addition, we applied a cross-validation strategy: we used the last 100 hours to generate the sanitized model while testing on the other preceding 100-hour dataset.

Throughout the paper, we refer to detection and false positive rates as rates determined for a specific class of attacks that we observed in these data sets. We note that discovering ground truth for any realistic data set is currently infeasible. We are, in part, trying to address this chicken-and-egg problem through this work.

3.1 Experimental Results

Initially, we measured the detection performance of both Anagram and Payl when used as standalone AD sensors without sanitizing the training data. Then, we repeated the

experiments with the same setup and network traces, but we included the sanitization phase. Table 1 presents our findings, which show that sanitization boosts the detection capabilities of both sensors. The results summarize the average values of false positive (FP) and true positive (TP) rates. Both voting methods perform well. We used a granularity of three hours and a value of *V* which maximizes the detection performance (in our case $V \in [0.15, 0.45]$).

The optimal operating point appears to be that which maximizes the detection of the real alerts and has the lowest FP rate. Section 3.2 studies this point in more detail. For Anagram, when the sanitized and abnormal models were created, given the nature of the sensor, the two models were built to be disjoint (no abnormal feature would be allowed inside the sanitized model). The traffic contains instances of phpBB forum attacks (mirela, cbac, nikon, criman) for all three hosts that are analyzed.

Sensor	www1		www		lists	
	FP(%)	TP(%)	FP(%)	TP(%)	FP(%)	TP(%)
A	0.07	0	0.01	0	0.04	0
A-S	0.04	20.20	0.29	17.14	0.05	18.51
A-SAN	**0.10**	**100**	**0.34**	**100**	**0.10**	**100**
P	0.84	0	6.02	40	64.14	64.19
P-SAN	**6.64**	**76.76**	**10.43**	**61**	**2.40**	**86.54**

FP: false positive rate; TP: true positive rate

Sensor	www1	www	lists
A	0	0	0
A-S	505	59.10	370.2
A-SAN	**1000**	**294.11**	**1000**
P	0	6.64	1.00
P-SAN	**11.56**	**5.84**	**36.05**

signal-to-noise ratio (TP/FP); higher values mean better results

Table 1. Impact of the sanitization phase on the AD performance (A = Anagram; A - S = Anagram + Snort; A - SAN = Anagram + sanitization; P = Payl; P - SAN = Payl + sanitization)

Note that without sanitization, the normal models used by Anagram would be poisoned with attacks and thus unable to detect new attack instances appearing in the test data. Therefore, increasing AD sensor sensitivity (*e.g.* changing its internal detection threshold) would only increase false alerts without increasing the detection rate. When using previously known malcode information (using Snort signatures represented in an "abnormal model"), Anagram was able to detect a portion of the attack packets. Of course, this detection model is limited because it requires that a new 0-day worm will not be sufficiently different from previous worms that appear in the traces. To make matters worse, such a detector would fail to detect even old threats that do not have a Snort signature. On the other hand, if we enhance Anagram's training phase to include sanitization, we do not have to rely on any other signature or content-based sensor to detect malware.

Furthermore, the detection capability of a sensor is inherently dependent on the algorithm used to compute the distance of a new worm from the normal model. For example, although Payl is effective at capturing attacks that display abnormal byte distributions, it is prone to miss well-crafted attacks that resemble the byte distribution of the target site [9]. Our traces contain such attacks: we observe this effect when we use the sanitized strategy on Payl, as we can only get a maximum 86.54% attack detection rate. In this case the sanitization phase is a necessary but not sufficient process for reducing false negatives: the actual algorithm used by the sensor is also important in determining the overall detection capabilities of the sensor.

Interestingly, the combination of Payl operating on the *lists* data set without sanitization shows a high FP rate compared to the same case where sanitization is used. After investigating this phenomena, we realized that for a specific packet length, 161, the unsanitized model included a centroid for length 161 that caused many false positives. The sanitized model did not contain a specific centroid created for length 161 (the packets with this length were considered abnormal in the sanitization phase) and the closest length centroid (178) was used for the testing phase.

Overall, our experiments show that the AD signal-to-noise ratio (*i.e.,* TP/FP) can be significantly improved even in extreme conditions, when intrinsic limitations of the anomaly detector prevent us from obtaining a 100% attack detection, as shown in Table 1. Higher values of the signal-to-noise ratio imply better results. There is one exception: Payl used on the *www* data set. In this case, the signal-to-noise ratio is slightly lower, but the detection rate is still higher after using sanitization.

To stress our system and to validate its operation, we also performed experiments using traffic in which we injected worms such as CodeRed, CodeRed II, WebDAV, and a worm that exploits the nsiislog.dll buffer overflow vulnerability (MS03-022). All instances of the injected malcode were recognized by the AD sensors when trained with sanitized data. That result reinforced our initial observations about the sanitization phase: we can both increase the probability of detecting a zero-day attack and of previously seen malcode.

3.2 Analysis of Sanitization Parameters

We have seen how our sanitization techniques can boost the performance of the AD sensors. Our results summarize the FP and the detection rates as averaged values obtained

for the optimal parameters. We next explore these parameters and their impact on performance with a more detailed analysis using Anagram. We show the optimal operating point for any sensor can be identified automatically with offline tuning that requires no manual intervention.

There are three parameters we need to fine-tune: the granularity of the micro-models, the voting algorithm, and the voting threshold. In order to determine a good granularity, we have to inspect the volume of traffic received by each site (given the characteristics of the chosen anomaly detector) such that we do not create models that are undertrained. In our initial experiments, we used 3-hour, 6-hour, and 12-hour granularity. We employed both the simple and weighted voting algorithms proposed in Section 2. The threshold *V* is a parameter that needs to be determined once. It depends on the training set and the site/application modeled by the sensor. As we show, both the optimal values of V and the micro-model granularity appear to be the same for all the sites in our experiments.

In Figures 1 and 2, we present the performance of the system when using Anagram enhanced with the sanitization method applied on the *www1* traffic. We notice that the weighted voting algorithm appears to be a slight improvement. We seek a value for V that maximizes detection and achieves the lowest possible FP rate. We can observe that the sanitized model built using the 3-hour micro-models shows the best performance, achieving a detection rate of 100% and minimizing the FP rate. The granularity and the voting threshold are inversely proportional because for the same data set fewer models are built when the granularity is increased.

In Figures 3 and 4, we present the results for *www* and *lists* for a granularity of three hours and for both types of voting techniques. The best cases for these two sites are reached at almost the same value as the ones obtained for *www1*. We observe that the best case is where *V* has the minimum value 0.01 (this is dependent on the training data set).

To verify these results, we studied the impact that granularity has on the performance of the system. We fix the voting threshold, and we sample a large range of granularity values. This analysis allows us to determine the best granularity. In Figure 5, we can observe that the granularity of three hours performs the best, given the two threshold bounds *0.15* and *0.45* obtained from the previous experiments. For all other values of $V \in (0.15, 0.45)$, the granularity of three hours seemed to be the optimal choice. Notice that for $V = 0.45$, all values of granularity from 3-12 hours are optimal but not for $V = 0.15$.

When using Payl, the granularity of three hours again performs the best, given the two threshold bounds *0.15* and *0.55*. Payl behaves differently than Anagram due to its different learning algorithm. The way the models are built is more dependent on the number of training samples because models are created for each packet length.

As we mentioned previously, our technique assumes the use of a large training dataset in order to increase the probability that an individual datum which is normal is not incorrectly deemed an anomaly. To analyze the impact that training data set size has on performance, we tested our methodology on Anagram using a certain percentage of the micro-models, starting from a randomly chosen position in the training dataset, as shown in Figure 7. This experiment uses 300 hours of training data, a granularity of three hours per mico-model, the weighted voting scheme, and a threshold of $V = 0.45$. The FP rate degrades when only a percentage of the 100 models is used in the voting scheme. Another factor is the relationship between the internal threshold of the sensor, τ, and the voting threshold, V, and the way it influences the performance of the system. Intuitively, if the anomaly sensor is more relaxed, the data seen as anomalous by the micro-models will decrease. As a result, the sanitized model will actually increase in size and exhibit a smaller FP rate as shown in Figure 8. Although using a "relaxed" AD can improve the FP rate, we do not advocate such an approach to the extreme. In our experiments, the threshold for Anagram was set to τ = 0.4, and we analyzed the effect of changing the internal threshold had over the performance of our system. We observed that if we increase the internal threshold too much, the FP rate decreases along with the detection rate.

3.3 Computational Performance Evaluation

Using an AD sensor to classify input, especially if the AD sensor is operating inline with the input stream, can be expensive (in terms of latency for benign requests). We examine the average time it takes to process a request, and the impact that sanitization has on this time. In addition, we estimate the overall computational requirements of a detection system consisting of an AD sensor and a host-based shadow sensor. The AD sensor acts as a packet classifier that diverts all packets that generate alerts to the shadow sensor while allowing the rest of the packets to reach the native service. This architecture effectively creates two service paths.

Our goal is to create a system that does not incur a prohibitive increase in the average request latency *and* that can scale to millions of service requests. Due to the overhead of the shadow sensor, we cannot redirect all traffic to it. Therefore,we want the AD to shunt only a small fraction of the total traffic to the shadow. The shadow sensor serves as an oracle that confirms or rejects the AD's initial classification.

Although one could argue that using a shadow sensor alone is sufficient to protect a system from attack (and therefore we have scant need of a robust anomaly sensor in

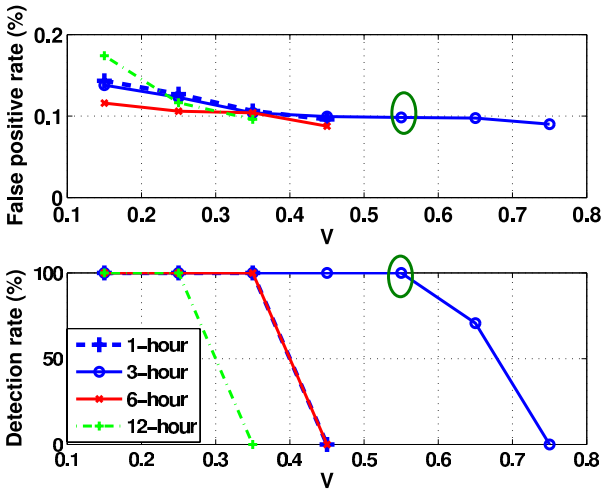

Figure 1. Performance for *www1* for 3-hour granularity when using simple voting and Anagram (V is the voting threshold; see section 2)

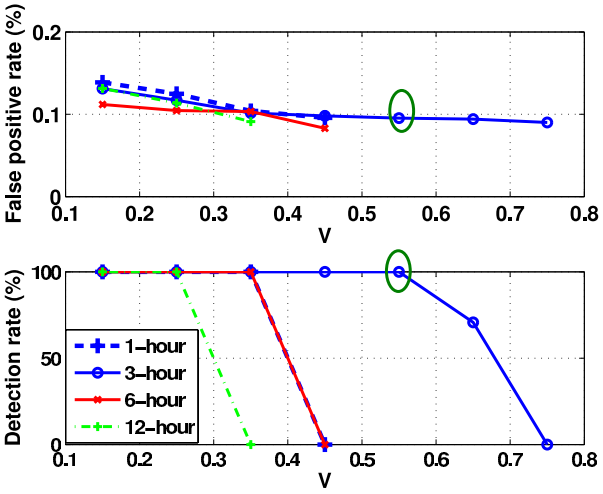

Figure 2. Performance for *www1* when using weighted voting and Anagram (V is the voting threshold)

the first place), shadow sensors have significant shortcomings. First, they impose a hefty performance penalty (due to the instrumentation, which could include tainted dataflow analysis, a shadow stack, control-flow integrity, instruction set randomization and other heavyweight detectors). Using a shadow sensor without the benefit of an AD sensor to pre-classify input would be unacceptable for many environments. Second, a shadow requires synchronization of state between it and the shadowed "production" application. In many environments, this is a difficult task. Finally, shadow sensors have only an incomplete notion of what malicious behavior is: they use instrumentation aimed at detecting certain classes of attacks. Thus, a shadow sensor is not a perfect oracle. It serves only to offer a lower bound on the removal of attacks (and it completely misses abnormalities) if it were used to directly "sanitize" data sets.

For our performance estimation, we used two instrumentation frameworks: STEM [21] and DYBOC [1]. STEM exhibits a 4400% overhead when an application such as Apache is completely instrumented to detect attacks. On the other hand, DYBOC has a lighter instrumentation, providing a faster response, but still imposes at least a 20% overhead on server performance. Given that we know ground truth based on *the attacks these sensors detect*, we can estimate what the answers of the shadow servers would be. We can also estimate the overall overhead based on the reported performance of the frameworks in [21] and [1].

To compute the overall overhead, we borrow the method used in [29], where the latency of such an architecture is defined as following:

$$l' = (l * (1 - fp)) + (l * O_s * fp)$$

where l is the standard (measured) latency of a protected service, O_s is the shadow server overhead, and fp is the AD false positive rate.

To quantify the performance loss/gain from using the sanitization phase, we compare the average latency of the system when using Payl and Anagram with sanitized and non-sanitized training data. Table 2 shows that the alert rate for both sensors does not increase by much after sanitizing the training data, and in some cases fewer numbers of packets will have to be processed by the shadow server (*lists* when using Payl).

Table 2. Latency for different anomaly detectors

Sensor	STEM			DYBOC		
	www1	www	lists	www1	www	lists
N/A	44	44	44	1.2	1.2	1.2
A	1.0301	1.0043	1.0172	1.0001	1.0000	1.0000
A-S	1.0172	1.1247	1.0215	1.0000	1.0005	1.0000
A-SAN	**1.0430**	**1.462**	**1.0430**	**1.0002**	**1.0006**	**1.0002**
P	1.3612	3.5886	28.5802	1.0016	1.0120	1.1282
P-SAN	**3.8552**	**5.4849**	**2.0320**	**1.0132**	**1.0208**	**1.0048**

3.4 Long-lasting Training Attacks

In some cases a worm may appear in all micro-models as well as the training dataset of the sanitized model. This scenario represents what we call a long-lasting training attack. In this attack, the adversary continuously targets a particular site such that the modeling process is disturbed.

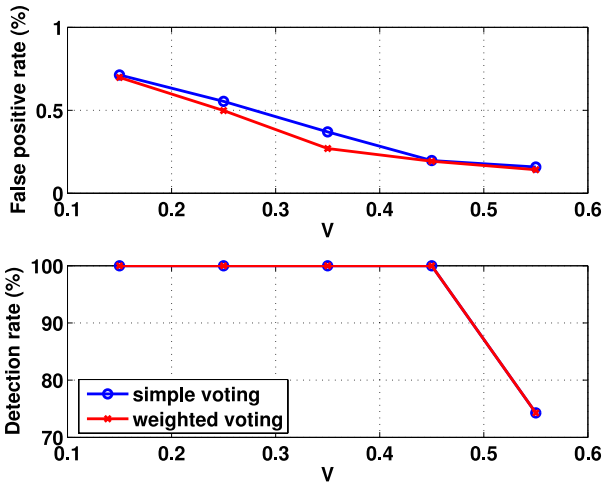

Figure 3. Performance for *www* for 3-hour granularity when using Anagram (V is the voting threshold)

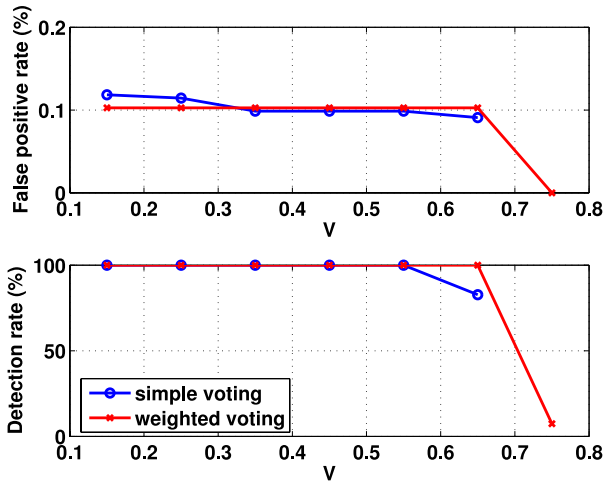

Figure 4. Performance for *lists* for 3-hour granularity when using Anagram (V is the voting threshold)

This situation is not covered by our previous experiments, which injected real worms into real traffic such that each one appeared in a small fraction of the micro-models. To test our methodology in such an extreme case, we injected a specific attack packet (in our case *mirela*) into every micro-model and the dataset from which the sanitized model was computed. Table 3 compares poisoned and "clean" or non-poisoned sanitized models. The results were obtained using Anagram, weighted voting, a granularity of three hours, and $V = 0.35$. We can see that this method can evade our architecture. For this reason, we next investigate ways to alleviate the impact of long-lasting training attacks.

Table 3. Long lasting training attacks

Sanitized	www1		www		lists	
model	FP(%)	TP(%)	FP(%)	TP(%)	FP(%)	TP(%)
non-poisoned	0.13	100	0.26	100	0.10	100
poisoned	0.10	29.29	0.26	38.27	0.10	35.80

4 Collaborative Sanitization

Section 3.4 noted that our local sanitization architecture has a weakness in the presence of long-lasting attacks in the initial set of training data. Because attack data may span multiple micro-models, it can poison a large portion of them. Since we predicate our cleaning capability on micro-model voting, extensive poisoning of the training data would seriously deteriorate our ability to detect long-lived or frequently occurring attack payloads. We hypothesize, however, that the distribution of such long-lived attacks among Internet hosts would require an adversary with significant resources (*e.g.,* a potentially large number of source IP addresses) — a requirement that effectively limits the scope of such attack to few target hosts or networks.

Given this hypothesis, we can counter the effects of such attacks by extending our sanitization mechanism to support sharing models of abnormal traffic among collaborating sites. Sharing these models enables a site to re-evaluate its local training data[2]. Our goal is to enhance the local view of abnormal behavior characteristics (rather than normal behavior characteristics, which cannot be meaningfully shared because they are unique to an individual site). As we will show, **"cross-sanitization"** between sites boosts our ability to remove long-lived or frequent attacks from the training data (regardless of whether or not the attack data is "targeted", *i.e.,* injected specifically to blind the sensor). Collaboration on this scale is not unheard of. Collaboration exists in the real world, and we believe it can exist in the digital world. In particular, examples of organizations employing a collaborative approach to defense include the Department of the Treasury and the FSISAC leveraging the talent and resources of the financial community to protect the community as a whole. The DISA/DoD manages and leverages information from its many customer .mil hosts. Universities with hundreds of divisions and units across schools and departments also follow this model.

4.1 Cross-Sanitization

In some sense, attack vectors that saturate training data define normal traffic patterns. Local knowledge alone may

[2]To alleviate the privacy concerns of sharing content, these models may incorporate privacy-preserving representations [12, 18].

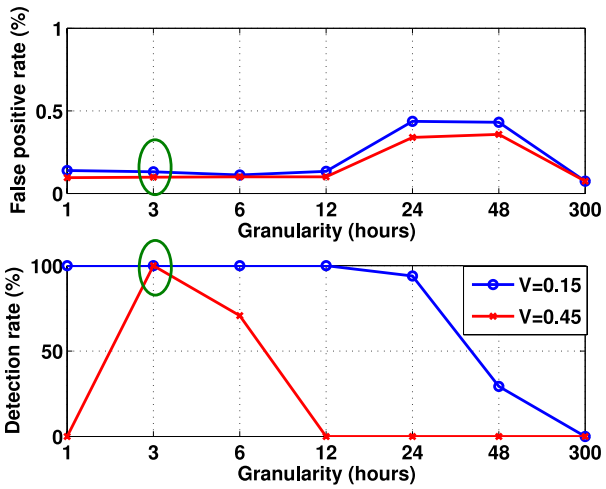

Figure 5. Granularity impact on the performance of the system for *www1* when using Anagram

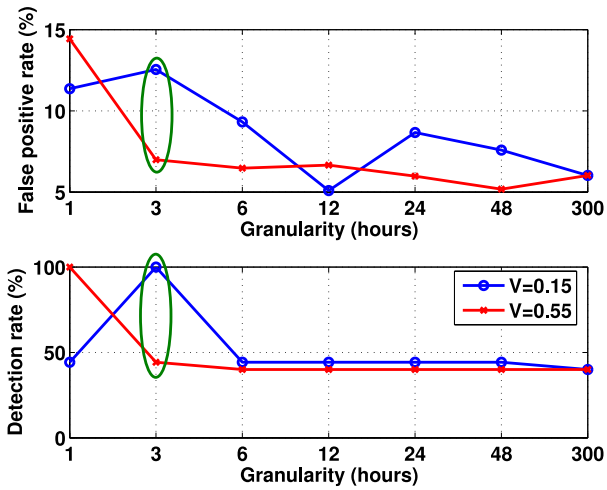

Figure 6. Granularity impact on the performance of the system for *www* when using Payl

not provide enough evidence to weed out consistent attack vectors in training data. To isolate and remove these vectors, we need to incorporate knowledge from some other remote source. This information sharing is the essence of cross-sanitization: comparing models of abnormality with those generated by other sites.

Cross-sanitization compares models of abnormality because normal models are tightly coupled with an individual site's traffic. In contrast, the consistency of characteristics of abnormal packets across sites can help filter out attacks that saturate the training data. Individual sites can utilize this external knowledge to cross-sanitize their training set and generate a better local normal model.

For an attacker to successfully blind each sensor in this type of environment, she would need to identify each collaborator and launch the same training attack on all participating sites for the same time period. Accomplishing this goal requires a significant amount of resources and knowledge. Therefore, we postulate that when a particular site experiences a targeted training attack, the attack data will not appear at all collaborating sites at the same time. As a result, with a large enough group of collaborators, some fraction of sites will have seen the attack, but will *not* have had their model corrupted by it. In this case, sharing abnormal models helps cleanse the local models of sites in the group that have been corrupted. When a site with sanitized model M_{san} receives the abnormal models $M_{abn_1} \ldots M_{abn_M}$ from its collaborators, it needs to compute a new model, M_{cross}. The methods to compute this model are presented in Sections 4.2 and 4.3.

Polymorphic attacks present a special challenge because each propagation attempt will display a distinct attack vector that may be captured in different abnormal models. We conjecture, however, that a polymorphic attack targeting a single site can still be captured by the local sanitization scheme presented in this paper. Section 5 explores how well our approach can cope with polymorphism.

4.2 Direct Model Differencing

Collaborative cross-sanitization requires us to define a method of directly comparing and "differencing" AD models. However, the composition of models may vary across sites depending on the particular AD algorithm in use and the specific representation of the model. If models are directly comparable or a translation method exists (although a full treatment of such a mechanism is beyond the scope of this work, we consider how to deal with complex models in Section 4.3), then we can construct a new local sanitized model from the shared abnormal models as follows:

$$M_{cross} = M_{san} - \bigcup \{M_{abn_i} \cap M_{san}\} \qquad (3)$$

where $M_{abn_i} \cap M_{san}$ represents the features common to both models.

4.3 Indirect Model Differencing

When models are more complex, *e.g.* probabilistic or statistical models, the model differencing computation cannot be applied analytically, but indirectly. Equation (3) is expressed differently, not as model differencing, but as a difference of sets of packets used to compute the models.

We recompute the sanitized model using the information from M_{san} and $M_{abn_1} \ldots M_{abn_M}$. The dataset used

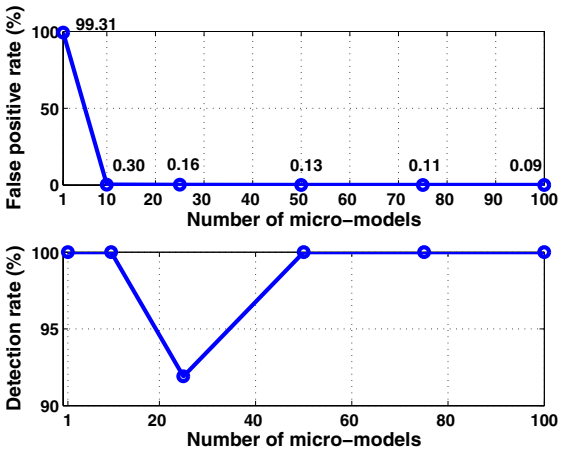

Figure 7. Impact of the size of the training dataset for *www1*

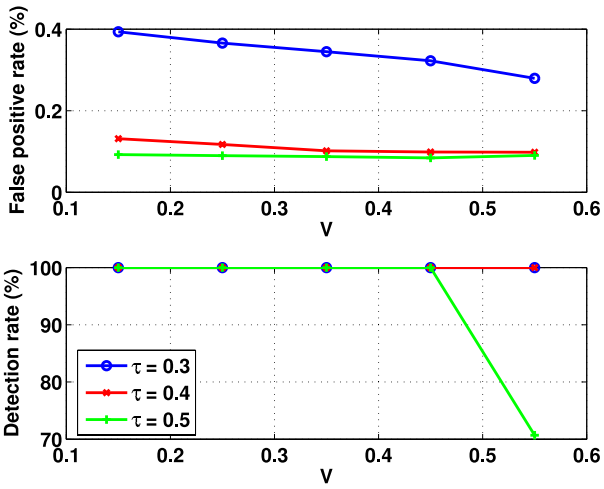

Figure 8. Impact of the anomaly detector's internal threshold for *www1* **when using Anagram**

in the second phase of the local sanitization is tested against M_{san} (we identify the packets that are normal, that is, used for actually computing M_{san}). The packets labeled as normal ($TEST(P_j, M_{san}) = 0$) are also checked against each of the collaborative abnormal models, $M_{abn_1} \ldots M_{abn_M}$. Note that $TEST(P_j, M_{abn_i}) = 0$ means that the packet is labeled normal by an abnormal model, which translates to the fact the packet is abnormal. If at least one of the abnormal models labels a packet P_j as normal (*i.e.,* the packet is considered abnormal by at least one collaborator), then features are extracted from the packet and used for computing the new local abnormal model, otherwise they are used for computing the cross-sanitized model.

Table 4. Recalculating sanitized and abnormal models. These routines use the abnormal models of collaborating peers to regenerate models of both normal and abnormal local data.

ROUTINE CROSSSANITIZED()
$\forall i \in [1..M]$
if 0=TEST(P_j, M_{san}) and 1=TEST(P_j, M_{abn_i})
$T_{cross} \leftarrow P_j$
$M_{cross} \leftarrow$ AD(T_{cross})
ROUTINE CROSSABNORMAL()
$\exists i \in [1..M]$
s.t. 0=TEST(P_j, M_{san}) and 0=TEST(P_j, M_{abn_i})
$T_{cabn} \leftarrow P_j$
$M_{cabn} \leftarrow$ AD(T_{cabn})

4.4 Additional Optimizations

Although direct/indirect model differencing can help identify abnormal samples that have poisoned a site, we must take care during the comparison. Because sites exhibit content diversity [28], (*i.e.,* they do not experience identical traffic flows), an abnormal model from site B may include some common but ultimately legitimate data from site A. In other words, data items that are indeed normal for a particular site can be considered abnormal by others. If site A attempts to identify abnormal content in its local model using cross-sanitization with site B, then A may incorrectly remove legitimate data patterns from its model along with truly abnormal or malicious data patterns. Doing so increases the false positive rate — an increase that may not be matched by an increase in detection rate.

An alternative approach to reconciling different models or disagreements between models involves the use of a shadow server. If the sanitized model and an abnormal model disagree on the label of a packet (for example, the sanitized model labels it normal and the abnormal one as abnormal), we redirect the traffic to the shadow server to determine if the packet causes a real attack. Based on this information the packet is used in the construction of either the local sanitized model or the local abnormal model.

5 Performance of Collaborative Sanitization

This Section shows that even if local sanitization fails to detect an attack, we can compensate by using the external information received from other collaborating sites.

Furthermore, we show that the performance of the local architecture remains unaffected when faced with polymorphic attacks. The experiments in this Section use the Anagram sensor and were conducted on a PC with a 2GHz AMD Opteron processor 248 and 8G of RAM, running Linux.

5.1 Training Attacks

We will assume that at least some of the collaborative sites are poisoned by a long lasting training attack, while others were able to filter it and use it for building the abnormal model. If the targeted site receives an abnormal model that contains an attack vector, the local sanitized model can be "cross-sanitized" by removing the common grams between the two models (direct model differencing). Given the diversity in content exhibited by different sites, the same gram can be characterized differently by different sites. Therefore, it is possible that after cross-sanitation the sanitized model becomes smaller. As an immediate consequence, the false positive rate will increase.

We consider all the possible cases in which each of our three hosts model is poisoned by each of the four attacks present in our data. When one site is poisoned, we consider that the other two are not. Every poisoned host receives the abnormal models M_{abn} of its peers in order to cross-sanitize its own model, M_{pois}. Table 5 presents the average performance of the system before and after cross-sanitization when using direct and indirect model differencing.

Table 5. Performance when the sanitized model is poisoned and after it is cross-sanitized when using direct/indirect model differencing

Model	www1		www		lists	
	FP(%)	DR(%)	FP(%)	DR(%)	FP(%)	DR(%)
M_{pois}	0.10	44.94	0.27	51.78	0.25	47.53
M_{cross} **(direct)**	**0.24**	**100**	**0.71**	**100**	**0.48**	**100**
M_{cross} **(indirect)**	**0.10**	**100**	**0.26**	**100**	**0.10**	**100**

In the case of direct model differencing, once the cross-sanitization is done, the detection rate is improved, but the false positive rate degrades. To further investigate how the cross-sanitization influences the performance of the local systems, we analyze the size of the models (presented in Table 6).

As Table 6 shows, the size of the models has decreased. This decrease leads to an increase in the FP rate. As we mentioned before, this behavior is a disadvantage of our distributed sanitization method, as it depends on site diversity.

Table 6. Size of the sanitized model when poisoned and after cross-sanitization when using direct/indirect model differencing

Model	www1		www		lists	
	#grams	file size	#grams	file size	#grams	file size
M_{abn}	2,289,888	47M	199,011	3.9M	6,025	114K
M_{pois}	1,160,235	23M	1,270,009	24M	43,768	830K
M_{cross} **(direct)**	**1,095,458**	**21M**	**1,225,829**	**24M**	**37,113**	**701K**
M_{cross} **(indirect)**	**1,160,004**	**23M**	**1,269,808**	**24M**	**43,589**	**828K**

Furthermore, this phenomena provides a potential avenue of attack for an adversarial collaborator. We consider defending against this type of attack to be out of the scope of our current efforts, but Byzantine robustness or reputation systems can be applied in the future. Of course, even appropriately authenticated or otherwise trustworthy peers could be exploited after they are included in the collaborative network. We note that dealing with trusted insiders is an inherently hard problem faced by many large-scale collaborative systems, from anonymity networks to mandatory access control frameworks.

In order to improve our method for cross-sanitization, we can use the indirect model differencing approach. This approach tests the poisoned local model and the collaborative abnormal models against the second training dataset used in our local methodology. The goal of this method is to eliminate the packets responsible for poisoning the local model from the training data set. The most challenging part of this method is to set the internal threshold of Anagram when testing the traffic against the abnormal models. An intuitive approach is to actually use the inverse value of the normal thresholding. For example, if the internal threshold for Anagram when testing against the normal model was τ, then the threshold for abnormal models would be 1-τ. In our experiments, we used an internal threshold of 0.4: the threshold for abnormal models becomes 0.6 (analyzing the scores given by the packets that contributed to the poisoning 0.5 would have been also enough). As we can observe in Table 5, the FP rate has improved and the detection rate remains at 100%. The improvement of the FP rate is reflected in the size of the cross-sanitized models (see Table 6).

In terms of computational performance, as expected, the indirect model differencing is more expensive than the direct model differencing (see Table 7). There is a tradeoff between how fast the cross-sanitization needs to be done and how high the FP rate is. If a higher FP rate is allowed, a quicker cross-sanitization can be applied by using the direct differencing; otherwise the best solution is the indirect

model differencing.

Finally, any of the methods can be refined using feedback from a shadow sensor, but this process would introduce more computational effort. The benefit of using a shadow sensor is that we can cross-sanitize the model to optimize the FP rate.

Table 7. Time to cross-sanitize for direct and indirect model differencing

Method	www1	www	lists
direct	13.98s	26.35s	16.84s
indirect	1966.68s	1732.32s	685.81s

5.2 Polymorphic Attacks

Polymorphic attacks are of increasing concern. To test against such attacks, we used a popular polymorphic engine, CLET [6] to generate samples of polymorphic shellcode. In these experiments, we assume that an attacker tries to perform a training attack using a polymorphic vector (which implies that the exploit would include polymorphic shellcode). For the experiments, we used 2100 samples of shellcode generated with CLET. We used 100 micro-models with a three hour granularity derived from our dataset for *www1*. We poisoned each micro-model with 20 samples of shellcode. We also poisoned the data set from which the sanitized model is built with the remaining 100 shellcode samples.

We rebuilt the sanitized model with our system. In the voting strategy, all of the micro-models found the 100 shellcode samples as being anomalous, given that, on average, 82% of the grams from 100 samples were found abnormal by the micro-models. After the sanitized model was computed, we tested it against the testing dataset of 100 hours. As expected, the performance results were identical with the ones given when the sanitized model was constructed without any shellcode samples. These experiments indicate that most common polymorphic attacks can be handled by the local sanitization architecture.

6 Related work

Our approach for sanitizing training datasets shares elements with ensemble methods, which are reviewed in [7]. It is important to note that, while most of these methods traditionally fall in the category of supervised learning algorithms, due the applications of our work (*e.g.* real network traffic), we are forced to use unlabeled training data. In particular, we construct a set of classifiers and then classify the new data points using a (weighted) vote. We generate AD models from slices of the training data, thus manipulating the training examples presented to the learning method. Another similar machine learning approach is that of Bagging predictors [2], which uses a learning algorithm with a training set that consists of a sample of m training examples drawn randomly for the initial data set. The *cross-validated committees* method [19] proposes to construct a training model by leaving out disjoint subsets of the training data. ADABoost [11] generates multiple hypothesis and maintains a set of weights over the training example. Each iteration invokes the learning algorithm to minimize the weighted error and returns a hypothesis, which is used in a final weighted vote.

One of the most similar work to ours is MetaCost [8], which is an algorithm that implements cost-sensitive classification. Instead of modifying an error minimization classification procedure, it views the classifier as a black box, the same as we do, and wraps the procedure around it in order to reduce the loss. MetaCost estimates the class probabilities and relabels the training examples such that the expected cost of predicting new labels is minimized. Finally it builds a new model based on the relabeled data. Our algorithm also labels the training examples and ignores the abnormal ones in the training process.

We have previously explored the feasibility of cleaning traffic [5]. That work contains a very limited analysis and did not address the problem of long-lasting attacks in the training data. In contrast, this paper performs an extended analysis on larger and more realistic data sets to help confirm the hypothesis that cleaning is possible. We also present alternatives that can be used when the local architecture fails due to long lasting training attacks.

JAM [24] focuses on developing and evaluating a range of learning strategies for fraud detection. That work presents methods for "meta-learning" by computing sets of "base classifiers" over various partitions or sampling of the training data. The combining algorithms proposed are called "class-combiner" or "stacking" and they are built based on work presented in [3] and [31]. The exchange of abnormal models was used in [24] for commercial fraud detection. Their results show that fraud detection systems can be substantially improved by combining multiple models of fraudulent transactions shared among banks. We apply a similar idea in the case of network traffic content-based anomaly detection in order to cross-sanitize the normal model.

The perceived utility of anomaly detection is based on the assumption that malicious inputs rarely occur during the normal operation of the system. Because a system can evolve over time, it is also likely that new *non-malicious* inputs will be seen [10]. Perhaps more troubling, Fogla and Lee [9] have shown how to evade anomaly classifiers by constructing polymorphic exploits that blend with nor-

mal traffic (a sophisticated form of mimicry attack [27]), and Song *et al.* [23] have improved on this technique and shown that content–based approaches may not work against all polymorphic threats, since many approaches often fix on specific byte patterns [17].

7 Conclusions

Due to recent advances in polymorphic attacks, we believe that the research community should make a concerted effort to revive the use of content–based anomaly detection as a first-class defensive technique. To that end, we introduce a novel sanitization technique that significantly improves the detection performance of out-of-the-box anomaly detection (AD) sensors. We are the first to introduce the notion of micro-models: models of "normal" trained on small slices of the training data set. Using simple weighted voting schemes, we significantly improve the quality of unlabeled training data by making it as "attack-free" and "regular" as possible. Our approach is straightforward and general, and we believe it can be applied to a wide range of unmodified AD sensors (because it interacts with the training data rather than the AD algorithm) without incurring significant additional computational cost other than in the initial training phase.

The experimental results indicate that our system can serve both as a stand-alone sensor and as an efficient and accurate online packet classifier using a shadow sensor. Furthermore, the alerts generated by the "sanitized" AD model represent a small fraction of the total traffic. The model detects approximately 5 times more attack packets than the unsanitized AD model. In addition, the AD system can detect more threats both online and after an actual attack, since the AD training data are attack-free. In case local sanitization is evaded, we extend our methodology to support sharing models of abnormal traffic among collaborating sites. A site can cross-sanitize its local training data based on the remote models. Our results show that if the collaborating sites were targeted by the same attack and they were able to capture it in their abnormal models, the detection rate can be improved up to 100%.

Obtaining anomaly sensors from the research community is a difficult process; most sensors are heavily protected IP or are under active development. We are, however, currently investigating two additional sensors: one based on `libanomaly` and one based on the pH system [22]. We plan to investigate how to clean training data for these algorithms to help show that our approach extends to other AD sensors.

Acknowledgements

We would like to thank Yingbo Song for helpful feedback and Wei-Jen Li and Vanessa Frias-Martinez for interesting discussions. This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-06-2-0221, the Army Research Office under grant No. DA W911NF-04-1-0442, and by the National Science Foundation under NSF grants CNS-06-27473 and CNS-04-26623. We authorize the U.S. Government to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, August 2005.

[2] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] P. K. Chan and S. J. Stolfo. Experiments in Multistrategy Learning by Meta-Learning. In *Proceedings of the second international conference on information and knowledge management*, pages 314–323, Washington, DC, 1993.

[4] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong. On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits. In *ACM Conference on Computer and Communications Security*, Alexandria, VA, 2005.

[5] G. F. Cretu, A. Stavrou, S. J. Stolfo, and A. D. Keromytis. Data Sanitization: Improving the Forensic Utility of Anomaly Detection Systems. In *Workshop on Hot Topics in System Dependability (HotDep)*, 2007.

[6] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. S. von Underduk. Polymorphic Shellcode Engine Using Spectrum Analysis. *Phrack*, 11(61-9), 2003.

[7] T. G. Dietterich. Ensemble Methods in Machine Learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[8] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[9] P. Fogla and W. Lee. Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. In *Proceedings of the* 13^{th} *ACM Conference on Computer and Communications Security (CCS)*, pages 59–68, 2006.

[10] S. Forrest, A. Somayaji, and D. Ackley. Building Diverse Computer Systems. In *Proceedings of the* 6^{th} *Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[11] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[12] S. S. Janak Parekh, Ke Wang. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *SIGCOMM Workshop on Large Scale Attack Defense*, 2006.

[13] C. Kruegel, T. Toth, and E. Kirda. Service Specific Anomaly Detection for Network Intrusion Detection. In *Symposium on Applied Computing (SAC)*, Madrid, Spain, 2002.

[14] R. P. Lippmann and J. Haines. Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation. In *Proceedings of the Recent Advances in Intrusion Detection (RAID 2000)*, pages 162–182, 2000.

[15] J. McHugh. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. *ACM TISSEC*, 3(4):262–291, 2000.

[16] D. Moore and C. Shannon. The Spread of the Code Red Worm (CRv2). `http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml`.

[17] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *IEEE Security and Privacy*, Oakland, CA, 2005.

[18] J. J. Parekh. *Privacy-Preserving Distributed Event Corroboration*. PhD thesis, Columbia University, 2007.

[19] B. Parmanto, M. P. W., and H. R. Doyle. Improving Committee Diagnosis with Resampling Techniques. *Advances in Neural Information Processing Systems*, 8:882–888, 1996.

[20] H. Patil and C. N. Fischer. Efficient Turn-time Monitoring Using Shadow Processing. In *Proceedings of the* 2^{nd} *International Workshop on Automated and Algorithmic Debugging*, 1995.

[21] S. Sidiroglou, M. E. Locasto, S. W. Boyd, and A. D. Keromytis. Building a Reactive Immune System for Software Services. In *Proceedings of the USENIX Technical Conference*, June 2005.

[22] A. Somayaji and S. Forrest. Automated Response Using System-Call Delays. In *Proceedings of the* 9^{th} *USENIX Security Symposium*, August 2000.

[23] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. On the Infeasibility of Modeling Polymorphic Shellcode. In *ACM Computer and Communications Security Conference (CCS)*, 2007.

[24] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan. Cost-based Modeling for Fraud and Intrusion Detection: Results from the JAM Project. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2000.

[25] K. M. Tan and R. A. Maxion. Why 6? Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 188–201, May 2002.

[26] C. Taylor and C. Gates. Challenging the Anomaly Detection Paradigm: A Provocative Discussion. In *Proceedings of the* 15^{th} *New Security Paradigms Workshop (NSPW)*, pages 21–29, September 2006.

[27] D. Wagner and P. Soto. Mimicry Attacks on Host-Based Intrusion Detection Systems. In *ACM CCS*, 2002.

[28] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.

[29] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.

[30] K. Wang and S. J. Stolfo. Anomalous Payload-based Network Intrusion Detection. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2004.

[31] D. Wolpert. Stacked Generalization. In *Neural Networks*, volume 5, pages 241–259, 1992.