STAND: Sanitization Tool for ANomaly Detection

Gabriela F. Cretu, Angelos Stavrou, Salvatore J. Stolfo, Angelos D. Keromytis
{gcretu, angel, sal, angelos}@cs.columbia.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.

General Terms

Security, Measurement, Experimentation

Keywords

anomaly detection, sanitizing training data, micro-models, training attacks

1. INTRODUCTION

Recent research indicates that signature-based network intrusion detection systems (NIDS) are quickly becoming ineffective in identifying malicious traffic [13, 6, 19]. Unfortunately, the threat posed by polymorphic attack engines will overwhelm signature-based detectors. Relying on anomaly-based detection (AD) sensors to detect 0-day attacks has become a necessity rather than an option. However, effective anomaly detection requires highly accurate modeling of normal traffic — a process that remains an open problem and the subject of this paper. 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 the following problems:

- The generated model of normal traffic can under-fit the actual normal traffic. Under-fitting for an AD system means that the AD system will flag traffic as "normal" even if this traffic does not belong to the real normal model leading to an overly generalized model of what we deem as "normal" traffic. Attackers who have sufficient room to disguise their exploit as normal can bypass a poorly defined normality model, thus increasing the "false negatives" of the AD sensor.

- The model of normal traffic can over-fit the training data: non-attack traffic that is not observed during training can be regarded as anomalous. Over-fitting may generate an excessive amount of false alerts or "false positives."

- 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. In short, the AD system may produce false negatives. This risk becomes a limiting factor of the size of the training set [21].

- Even in the presence of ground truth, creating a single model of normal traffic which includes all non-attack traffic can result in under-fitting and over generalization.

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 and monolithic normality model is the product of a training phase that traditionally uses all 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 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. Therefore, we produce 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 represent 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 AD micro-models. Using a voting scheme, we can determine which of the packets can be considered abnormal and needs to be removed from our training set. In our analysis, we explore the efficiency and tradeoffs of simple majority voting and more advanced 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.

One situation where our assumptions do not hold is the case 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 provides a major advance over the closest prior work [2]. In addition, we extend that short work by conducting a more thorough analysis and experimental evaluation of our technique.

Our novel distributed strategy leverages the location diversity of collaborating sites to exchange information that can be used to clean each site's training data set. Indeed, we introduce a two-phase training process: initially, we compute the AD models of what we deem as "normal" and "abnormal" from the training set locally 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 the remotely received "abnormal" models, 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.

We consider 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 and to sanitize the training set. As a second scenario, we assume that we have an latency-expensive oracle to help us classify "suspect data" and to differentiate between false positives and true positives. In practice, our oracle consists of a heavily instrumented host-based "shadow" server system akin to a honeypot 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 detecting malicious actions performed by the server when processing the suspicious data. However, to achieve high (close to 100%) accuracy for the oracle, a heavily instrumented shadow system is required. Such systems perform substantially slower, usually orders of magnitude slower, than the native, uninstrumented application [1]. Therefore, if we take into account the oracle constraints, we should focus on producing a sensor that identifies few "suspect data" items that are subjected to further but time-expensive tests. 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."

This second scenario is used to demonstrate that failure to substantially reduce the false positive rate of a network AD sensor does not render the sensor useless: the false positive rate can be mitigated the use of an expensive host-based instrumented shadow server. Therefore, false positives do not incur damage to the system under protection, and do not flood an operational center with too many alarms. Instead, the shadow server processes both true attacks and incorrectly classified packets to validate whether a packet signifies a true attack. These packets are still processed by the intended shadowed application and only cause an increased delay for network traffic incorrectly deemed an attack.

2. LOCAL SANITIZATION

Utilizing an effective sanitization process for an AD training data set is of paramount importance if we want to generate an accurate and precise normal model. 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 the training data sets obtained from real-world network traces makes such labeling infeasible. In addition, semi–supervised or even un–supervised 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 the un–supervised training can detect 100% of the attacks, the resulting normal model can potentially 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 considered non-regular and thus abnormal. This type of data should not form part of the normal model since 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, therefore producing a large, over-estimated and enlarged normal model. We introduce a new unsupervised training approach that attempts to determine both attacks and abnormalities and separate them from what we deem as the actual regular (normal) model, based on some conjectures.

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 [12]. 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.

However, we must be cautious because 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 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.

2.1 Micro-models

Our method of sanitizing the training data for an AD sensor employs the idea of "ensemble methods." Dietterich [8] 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, the epochs can range over the entire set of training data. Our experiments, reported in Section 3, analyze network packet traces captured over approximately 300 hours. We find that a value of g from 3 to 5 hours was sufficient to generate well behaved micro-models.

2.2 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 the same or a second set of training data. By same we refer to the training set used to produce the micro-models. Splitting the training dataset into two sets represents the worst case scenario since 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*:

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

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:

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

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, it 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}.

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

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

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. Our approach is agnostic to the particular AD algorithm in use. Consequently, we believe that our approach can help generate sanitized models for a wide range of anomaly detection systems.

3. EVALUATION OF SANITIZATION

In this section, we quantify the increase in the detection accuracy of any content-based anomaly detection system when we apply training data sanitization. We treat the AD sensor as a "black-box" avoiding using optimizations that are specific to an AD system. We evaluate our approach using two different scenarios. In the first scenario, we measure the performance of the AD sensor with and without sanitization. Additionally, we consider the case where we use the AD as a packet classifier for incoming network traffic: we test each packet and consider the computational costs involved by diverting each alert to a back-end shadow server. Both the feasibility and scalability of this scenario depend mainly on the amount of alerts generated by the AD sensor, since all "suspect-data"(those that sense the AD sensor to generate an alert) are delayed significantly by the shadow server and such data come from both real attacks and false alerts. For our experiments, we use two content-based anomaly detectors Anagram [23] and Payl [22]. These AD sensors have quite different learning algorithms to determine whether they have seen a particular datum before or not. We do not describe the details of the algorithms used during the testing phase, as they are not germane to the topic of this paper.

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 which from now on we will call *www*, *www1* and *lists*. We partitioned these 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, consisting of approximately $775,000$ packets including 99 worm packets for *www1*, $656,000$ packets including 70 worm packets for *www* and $26,000$ packets including 81 worm packets for *lists*, were used for testing. 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.

3.1 Experimental Results

Initially, we measured the detection performance for both Anagram and Payl when used as stand-alone online anomaly detectors without sanitizing the training data. Then, we repeated the same experiments using the same setup and network traces but including the sanitization phase. Table 1 presents our findings, which show that sanitization boost the detection capabilities of both AD 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 3 hours and a value of V which maximizes the detection performance (in our case $V \in [0.15, 0.45]$). We consider the optimal operating point to be the one from the points that maximize the detection of the real alerts and have the lowest false positives rate. We study the optimal operation point later in section 3.2. 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.

Table 1: AD sensors comparison (A=Anagram; A-S=Anagram+Snort; A-SAN=Anagram+sanitization; P=Payl; P-SAN=Payl+sanitization)

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**

It is important to notice 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 the 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 worm 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 ability 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 [23]. Our traces contain such attacks, which is the reason why, when we use the sanitized strategy on Payl, we can only get a maximum 86.54% worm detection rate as opposed to 100%. The sanitization phase is necessary to clear models to reduce false negatives in detecting malcode but not a sufficient one: the actual algorithm used by the sensor is also very important in determining the overall detection capabilities of the sensor.

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 we can observe in table 2. Higher values of the signal-to-noise ration imply better results. There is one exception, for Payl, in case of host *www*, in which the signal-to-noise ratio is slightly decreased, but still the detection rate is higher when using the sanitization technique as opposed to the case in which we don't.

Table 2: AD sensors comparison (signal-to-noise ratio) - higher values mean better results

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**

To stress our system and to validate its operation, we also performed experiments using traffic in which we artificially 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 anomaly detectors when trained with sanitized data. That re-enforced 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

In the previous section, we presented experimental evidence that the sanitization technique can boost the performance of the two chosen anomaly detectors. The results summarized the false positive and the detection rates as averaged values obtained for the optimal parameters. In this section, we explore these parameters and their impact on performance, with a more detailed analysis in case of Anagram. Furthermore, 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 also 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. For the voting algorithms, we employed the voting algorithms proposed in section 2: simple or weighted. The threshold V is a parameter of our system that needs to be determined once and it depends on the training set and the site/application we model. As we show, both the optimal values of V and the micro-model granularity appear to be the same for all the sites we used 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. Between the two voting techniques, we notice a slight improvement in case of the weighted algorithm. We seek a value for V that maximizes detection achieving the lowest possible false positive rate. We can observe that the sanitized model built using the 3-hour micro-models shows better performance, achieving a detection rate of 100% and minimizing the false positive rate. The granularity and the voting threshold are inversely proportional since for the same dataset less models are built when the granularity is increased. In figures 3 and 4 we present the results for *www* and *lists* for granularity of 3-hour and for both types of voting techniques. We can observe that the best cases for these two sites are reached at almost the same value as the ones obtained for *www1*. To verify that, we study the impact of the granularity has on the performance of the system. We fix the voting threshold, and we sample a large range of granularity values. This is a quality analysis that allows us to determine the best granularity. In figure 5, we can observe that indeed the granularity of 3-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 3 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 $V = 0.15$.

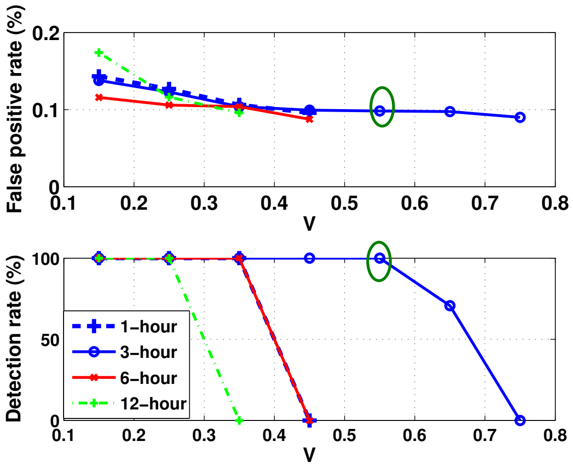

Figure 1: Performance for *www* for 3-hour granularity when using simple voting and Anagram

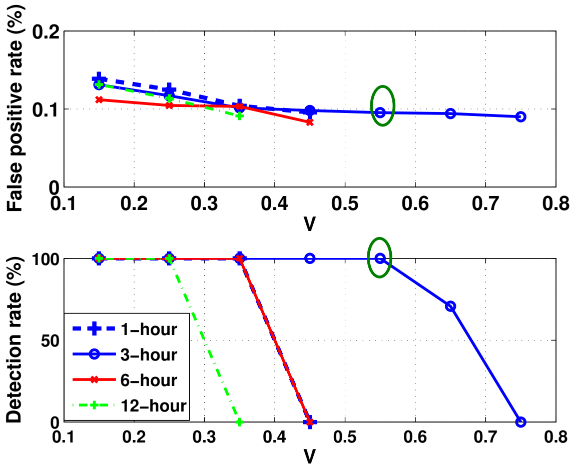

Figure 2: Performance for *www1* when using weighted voting and Anagram

When using Payl, again the granularity of 3-hours performs the best, given the two threshold bounds *0.15* and *0.55*. Payl behaves differently than Anagram though, due to the different type of learning algorithm that it performs.

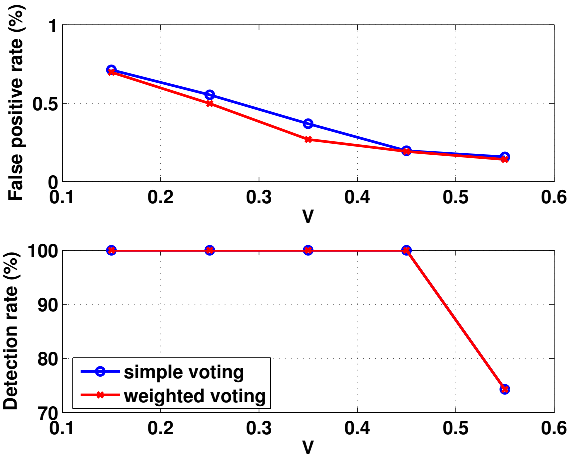

Figure 3: Performance for *www* for 3-hour granularity when using Anagram

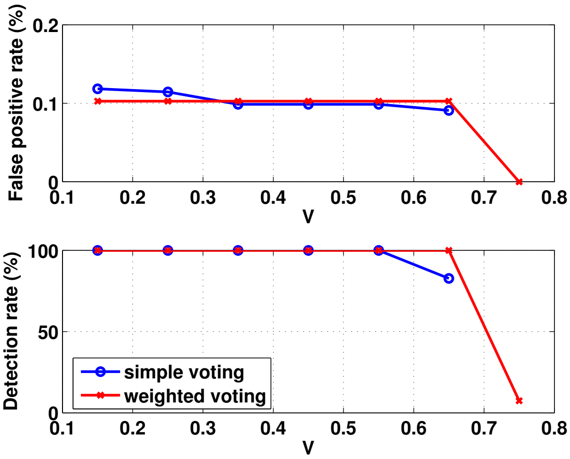

Figure 4: Performance for *lists* for 3-hour granularity when using Anagram

The way the models are built is more dependent on the number of training samples given that models are created for each packet length.

We mentioned in the previous sections that 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 deemed as a false anomaly. To analyze the impact on performance given by the size of the training dataset, we tested our methodology on Anagram using a certain percentage of the micro-models, starting from randomly chosen position in the training dataset, as shown in table 3. We consider the case in which we have the 300 hours of training data and we use a granularity of 3 hour per micro-model, we use the weighted voting algorithm and we fix the threshold $V = 0.45$. We can observe that the false positive rate degrades when only a percentage of the 100 models is used in the voting scheme and also the impact appears on the detection rate as well. 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 detector is more relaxed, the data seen as anomalous by the micro-models will decrease, thus the sanitized model will actually increase its size, exhibiting a smaller false positive rate as shown in figure 7. Although this can be a method to actually improve the false positive rate, it cannot be taken to the extreme. In our experiments the threshold for Anagram was set to $\tau = 0.4$, and we analyzed the effect that the increase/decrease of the internal threshold had over the performance of our system. We can observe that if we increase too much the internal threshold, the false positive decreases along with the detection rate.

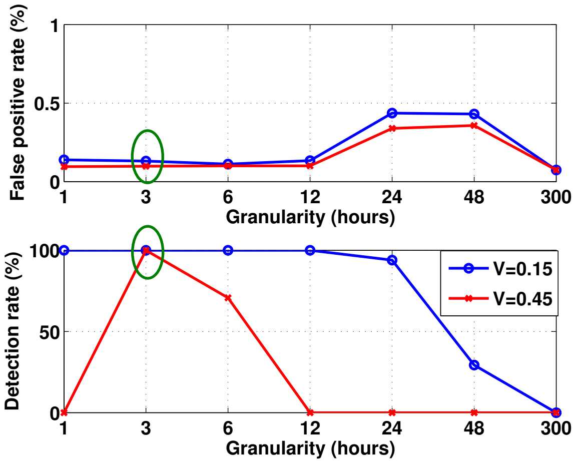

Figure 5: Granularity impact on the performance of the system for *www1* when using Anagram

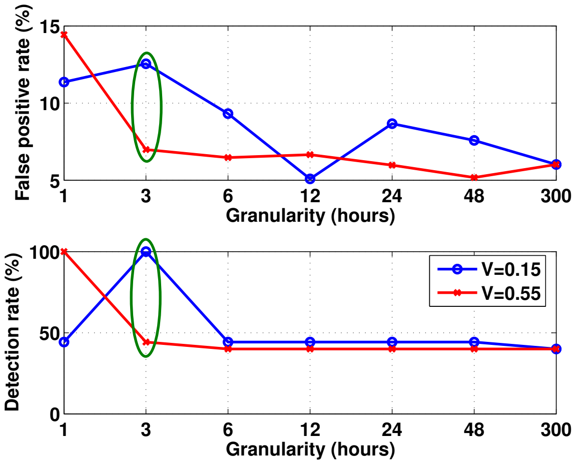

Figure 6: Granularity impact on the performance of the system for *www* when using Payl

3.3 Computational Performance Evaluation

Another aspect of an anomaly detection system that we would like to analyze is its impact on the average time that

Table 3: Impact of the size of the training dataset for *www1*

#micro-models	FP rate	TP rate
100%	0.0986%	100%
75%	0.1188%	100%
50%	0.1305%	100%
25%	0.1602%	91.91%
10%	0.3025%	100%
1%	99.3159%	100%

it takes to process a request. In addition, we estimate the overall computational requirements of a detection system consisting of an AD sensor and a host-based sensor (shadow server). The AD sensor acts as a packet classifier diverting all packets that generate alerts to the host-based sensor while allowing the rest of the packets to reach the native service. Our goal is to create a system that does not incur prohibitive increase in the average request latency and at the same time can scale to millions of service requests. Therefore, we would like the AD to shunt only a small fraction of the total traffic to the expensive shadow servers.

For our performance estimation, we used two well-known instrumentation frameworks: STEM [18] 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 the server performance. Given that we know the ground truth, we can estimate what the answers of the shadow servers would be. Also based on the declared performance of two frameworks in [18] and [1] we can estimate the overall overhead. To compute the overall overhead, we used the same method used in [23] 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.

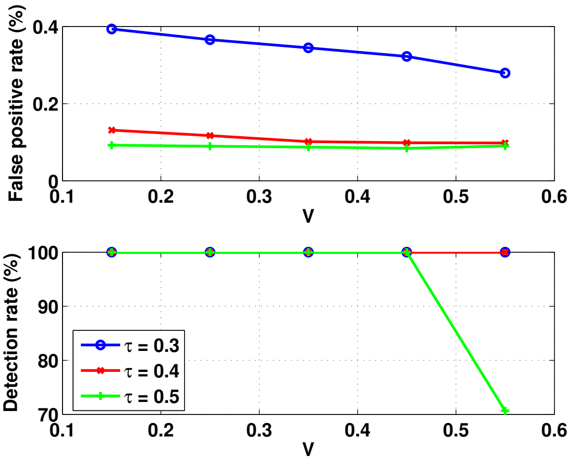

Figure 7: Impact of the anomaly detector's internal threshold for *www1* when using Anagram

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. From Table 4, we see that for both sensors the alert rate does not increase by much after sanitizing the training data, and in some cases fewer number of packets will have to be processed by the shadow server (in case of *lists* when using Payl).

Table 4: 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

We presented experiments on real traffic with real worm and attacks each one appearing in a small fraction of the micro-models we generated. However, there can be cases where some worms appear in all micro-models and also in the training dataset of the sanitized model. This scenario represents what is called a long-lasting training attack, where the adversarial continuously targets a particular site such that the modeling process is disturbed.

To test our methodology in such extreme case, we artificially injected every micro-model and also the dataset from which the sanitized model was computed with with one specific worm packet (in our case *mirela*). In table 5 we present the results obtained with the sanitized model un-poisoned and poisoned. The results were obtained when using Anagram, weighted voting, granularity of 3-hour and V=0.35.

Table 5: Long lasting training attacks

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

It is obvious that this method of evading our architecture would have an impact on the performance of our system. That is the reason why we need to further investigate ways to alleviate the impact of the the long-lasting training attacks.

4. COLLABORATIVE SANITIZATION

As observed in section 3.4, a weakness of the local sanitization architecture arises in the presence of a long-lasting attack in the initial set of training data. Because this attack data spans 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 at large would require an adversary with significant time and 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[1]. 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).

4.1 Cross-Sanitization

In some sense, attack vectors that saturate training data define normal traffic patterns. Local knowledge alone may not provide enough evidence to weed out some long-term 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 model of normal data.

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 *not* 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 $M_{abn_1} \dots M_{abn_M}$ 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 both our approaches (*i.e.,* local and collaborative sanitization) 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:

$$M_{cross} = M_{san} - \bigcup \{M_{abn_i} \cap M_{san}\} \tag{3}$$

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 re-compute the sanitized model using the information from M_{san} and $M_{abn_1} \dots M_{abn_M}$. The dataset used in the second phase of the local sanitization is tested against M_{san} (we identify the packets that are normal, respectively, 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, $M_{abn_1} \dots M_{abn_M}$. Note that $TEST(P_j, M_{abn_i}) = 0$ means that the packet is labeled normal by an abnormal model, which translates in the fact the packets 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 6: *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($T'_{san}$)
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($T'_{san}$)
```

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 [22], (*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 in-

[1] To alleviate the privacy concerns of sharing content, these models may incorporate privacy-preserving representations [14].

creases 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. Our experiments explore the use of vanilla "model differencing" and leave analysis of complex models as future work.

5. PERFORMANCE OF COLLABORATIVE SANITIZATION

In this section we show that even if the local sanitization fails to detect an attack, we can compensate by using the external information received from other collaborating sites. Furthermore, we show that in case of polymorphic attacks the performance of local architecture cannot be affected. For the experiments presented in this section, we used only Anagram, leaving Payl as a future work. Our tests 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 some of the collaborative sites are poisoned by a long lasting training attack, but still 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. That means that it is possible that after cross-sanitation the sanitized model becomes smaller, and 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 worms that are exhibited by our data. When one site is poisoned, we consider that the other two are not. Every poisoned host receives from its collaborative sites their abnormal models M_{abn} in order to cross-sanitize its own model, M_{pois}. Table 7 presents the average performance of the system before and after cross-sanitization, when using the direct and indirect model differencing.

Table 7: 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 case of the 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 8).

Table 8: 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**

To be noted that for our implementation, we stored the Anagram models as hash sets. In case of space constraints there are other data structures that can be used to store the models, like bloom filters used in [23], which do not store the whole information. As we can observe in table 8, the size of the models has decreased, which led to an increase in the false positive rate. As we mentioned before this is a big disadvantage of the method, as it is very dependent on sites diversity and in the same time it can be used as an attack tool by an adversarial collaborator (we need to have a credential system for the collaboration).

In order to improve our method for cross-sanitization, we can use the indirect model differencing approach, which assumes testing 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 determine and eliminate from the training dataset used in computing the local poisoned model the packets that actually poisoned the model. The most challenging part of this method is to set the internal threshold of Anagram when testing the traffic against the abnormal models. A very intuitive approach is to actually use the inverse value of the normal thresholding. That means that if the internal threshold for Anagram when testing against the normal model was τ the threshold for abnormal models would be 1-τ. In our experiments we used the internal threshold as 0.4, which led for the abnormal models to a threshold of 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 7, the false positive rate is improved while having the same detection rate of 100%. The improvement of the false positive rate is reflected in the size of the cross-sanitized models (see table 8).

In terms of computational performance, as expected, the indirect model differencing is more expensive than the direct model differencing (see table 9). There is a trade-off between how fast the cross-sanitization needs to be done and how high the false positive rate is. If a higher false positive rate is allowed, a quicker cross-sanitization can be applied by using the direct differencing; otherwise the best solution is the indirect model differencing. Also any of the methods can be further refined using the input from the shadow server, but introducing more computational effort. This way we can actually cross-sanitize the model optimizing the false positive rate.

Table 9: 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

Another type of attacks are Polymorphic attacks. To test against such attacks, we used one of the most popular polymorphic engines in the wild, CLET [7] and we generated polymorphic samples of shellcode. A shellcode sample has the following structure *[nop...][decoder][encrypted exploit][ret address]*. In our experiments we assume that an attacker tries to perform a training attack using a polymorphic vector (which also would imply that the attack would include polymorphic shellcode).

For our experiments, we used 2100 samples of shellcode generated using CLET. We used 100 micro-models with granularity 3-hour, built on our dataset for *www1* and we poisoned each of them with 20 samples of shellcode. We also poisoned the second dataset from which the sanitized model is built with the 100 shellcode samples left. We re-run the experiments for building the sanitized model. 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. According to our experiments we can assume that the problem of continuous polymorphic attacks can actually be handled by the local architecture.

6. RELATED WORK

Our approach shares elements with the ensemble method [8] because we also construct a set of classifiers and then classify the new data points using a (weighted) vote to decide. However, we modify the training phase by generating models from slices of the training data. In addition, the ensemble method requires the repetition of the learning algorithm several times, each time with a different subset of training examples. Another similar machine learning approach is that of Bagging predictors [3], which presents a learning algorithm with a training set that consists of a sample of m training examples drawn randomly for the initial data set. Cross-validated committees [15] proposes to construct a training model by leaving out disjoint subsets of the training data. ADABoost [9] 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. The methods presented above use supervised learning algorithms. In the case of network traffic we do not have either labeled or purely normal data available, and it is very expensive to classify the data manually. Our method is applied for unsupervised learning algorithms [16, 17], given the conjectures presented in the previous sections.

Unsupervised anomaly detection bears similarity to work in distance–based outliers [4, 10, 11] examine inter-point distances between instances in the data to determine the outliers, but the nature of the outliers is different. In network data, the same intrusion can appear multiple times as very similar instances, but their number is significantly smaller than the one of normal instances.

Previous research [2] attempts to explore the feasibility of cleaning traffic (rather than improving algorithms). Moreover, it contained limited analysis and did not address the problem of a long-lasting attack in the training data. In contrast, we first perform 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 [20] focuses on developing, implementing, and evaluating a range of learning strategies and combining techniques for fraud detection systems. The work presents methods for “meta-learning”, 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 [5] and [24].

The exchange of abnormal models was used in [20] for commercial fraud detection. The results presented in that paper show that fraud detection systems can be substantially improved in stopping losses due to fraud by combining multiple models of fraudulent transaction 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.

7. CONCLUSIONS

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: AD normal models 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 it can be applied to a wide range of unmodified AD sensors 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 server backend. 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 as the original 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 the 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%.

8. REFERENCES

[1] Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., and Keromytis,

A. D. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the* 14^{th} *USENIX Security Symposium* (August 2005).

[2] Anonymous. Data Sanitization: Improving the Forensic Utility of Anomaly Detection Systems. In *Workshop on Hot Topics in System Dependability (HotDep)* (2007).

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

[4] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. LOF: Identifying Density-Based Local Outliers. In *ACM SIGMOD International Conference on Management of Data* (2000).

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

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

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

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

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

[10] Knorr, E. M., and Ng, R. T. Algorithms for mining distance-based outliers in large datasets. In *24th International Conference on Very Large Data Bases* (1998).

[11] Knorr, E. M., and Ng, R. T. Finding intentional knowledge of distance-based outliers.

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

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

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

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

[16] Ramadas, M., Ostermann, S., and Tjaden, B. Detecting Anomalous Network Traffic with Self-Organizing Maps. In *Sixth International Symposium on Recent Advances in Intrusion Detection* (2003).

[17] Robertson, W., Vigna, G., Kruegel, C., and Kemmerer, R. Using Generalization and Characterization Techniques in the Anomaly-based Detection of Web Attacks. In *Network and Distributed System Security Symposium* (2006).

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

[19] Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D., and Stolfo, S. J. On the infeasibility of Modeling Polymorphic Shellcode for Signature Detection. In *Columbia University Computer Science Department Technical Report, CUCS 007-07* (2007).

[20] Stolfo, S., Fan, W., Lee, W., Prodromidis, A., and Chan, P. 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).

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

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

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

[24] Wolpert, D. Stacked Generalization. In *Neural Networks* (1992), vol. 5, pp. 241–259.