Adaptive Anomaly Detection via Self-Calibration and Dynamic Updating

Gabriela F. Cretu-Ciocarlie[1], Angelos Stavrou[2], Michael E. Locasto[2], and Salvatore J. Stolfo[1]
[1]Department of Computer Science, Columbia University
{gcretu, sal}@cs.columbia.edu
[2]Department of Computer Science, George Mason University
{astavrou,mlocasto}@gmu.edu

Abstract. The deployment and use of Anomaly Detection (AD) sensors often requires the intervention of a human expert to manually calibrate and optimize their performance. Depending on the site and the type of traffic it receives, the operators might have to provide recent and sanitized training data sets, the characteristics of expected traffic (*i.e.* outlier ratio), and exceptions or even expected future modifications of system's behavior. In this paper, we study the potential performance issues that stem from fully automating the AD sensors' day-to-day maintenance and calibration. Our goal is to remove the dependence on human operator using an unlabeled, and thus potentially dirty, sample of incoming traffic.
To that end, we propose to enhance the training phase of AD sensors with a self-calibration phase, leading to the automatic determination of the optimal AD parameters. We show how this novel calibration phase can be employed in conjunction with previously proposed methods for training data sanitization resulting in a fully automated AD maintenance cycle. Our approach is completely agnostic to the underlying AD sensor algorithm. Furthermore, the self-calibration can be applied in an online fashion to ensure that the resulting AD models reflect changes in the system's behavior which would otherwise render the sensor's internal state inconsistent. We verify the validity of our approach through a series of experiments where we compare the manually obtained optimal parameters with the ones computed from the self-calibration phase. Modeling traffic from two different sources, the fully automated calibration shows a 7.08% reduction in detection rate and a 0.06% increase in false positives, in the worst case, when compared to the optimal selection of parameters. Finally, our adaptive models outperform the statically generated ones retaining the gains in performance from the sanitization process over time.
Key words: anomaly detection, self-calibrate, self-update, sanitization

1 Introduction

In recent years, network anomalies such as flash crowds, denial-of-service attacks, port scans and the spreading of worms and botnets pose a significant threat for large-scale networks. The capability to automatically identify and diagnose anomalous behavior both in the network and on the host is a crucial component of most of the defense and failure recovery systems currently deployed in enterprises and organizations. Indeed,

Anomaly Detection (AD) sensors are becoming increasingly popular: host-based [24] and network-based [16, 17, 21, 25, 30] intrusion detection systems rely heavily on AD components to maintain their high detection rates and minimize the false positives even when other, non-AD sensors are involved in the detection process.

A major hurdle in the deployment, operation, and maintenance of AD systems is the calibration of these sensors to the protected site characteristics and their ability to "adapt" to changes in the behavior of the protected system. Our aim is to automatically determine the values of the critical system parameters that are needed for both training and long-term operation using only the intrinsic properties of existing behavioral data from the protected host. To that end, we first address the training stage and calibration of the AD sensor. We use an unlabeled, and potentially dirty sample of the training set to construct micro datasets. On one hand, these datasets have to be large enough to generate models that capture a local view of normal behavior. On the other hand, the resulting micro-models have to be small enough to fully contain and minimize the duration of attacks and other abnormalities which will appear in a minority of the micro datasets. To satisfy this trade-off, we generate datasets that contain just enough data so that the arrival rate of new traffic patterns is stable. The micro-models that result from each data set are then engaged in a voting scheme in order to remove the attacks and abnormalities from the data. The voting process is automatically adapted to the characteristics of the traffic in order to provide separation between normal and abnormal data.

The second objective is to maintain the performance level of the AD sensors over a medium or long time horizon, as the behavior of the protected site undergoes changes or evolution. This is not an easy task [21] because of the inherent difficulty in identifying the rate of change over time for a particular site. However, we can "learn" this rate by continuously building new micro-models that reflect the current behavior of the system: every time a new model is added to the voting process, an old model is removed in an attempt to adapt the normality model to the observed changes. Without this adaptation process, legitimate changes in the systems are flagged as anomalous by the AD sensor leading to an inflation of alerts. In contrast, our framework was shown to successfully adapt to modifications in the behavior of the protected system. Finally, our approach is agnostic to the underlying AD sensor, making for a general framework that has the potential to improve the general applicability of AD in the real world.

1.1 Contributions

Our target is to create a fully automated protection mechanism that provides a high detection rate, while maintaining a low false positive rate, and also adapts to changes in the system's behavior. In [4, 5], we have explored the basic problem and proposed the sanitization techniques for multiple sites using empirically determined parameters. We also presented a distributed architecture for coping with long-lasting attacks and a shadow sensor architecture for consuming false positives (FP) with an automated process rather than human attention.

Here, we apply those insights to the problem of providing a run-time framework for achieving the goals stated above. This is a significant advance over our prior work which, while not requiring a manually cleaned data set for training, relied on empirically

determined parameters and human-in-the-loop calibration methods. Along these lines, our current work provides the following contributions:

- Identifying the intrinsic characteristics of the training data, such as the arrival rate of new content and the level of outliers (*i.e.* self-calibration)
- Cleansing a data set of attacks and abnormalities by automatically selecting an adaptive threshold for the voting method presented previously based on the characteristics of the observed traffic resulting in a sanitized training data set (*i.e.* automatic self-sanitization)
- Maintaining the performance we gained by applying the sanitization methods beyond the initial training phase and extending them throughout the lifetime of the sensor by continuously updating the self-calibrated and self-sanitized model (*i.e.* self-update)

2 Ensemble Classifier using Time-based Partitions

In [4, 5], we focused on methods for sanitizating the training data sets for AD sensors. This resulted in better AD sensor performance (*i.e. higher detection rate while keeping the false positives low*). Here, we attempt to fully automate the construction of those models by calibrating the sanitization parameters using the intrinsic properties of the training data. We briefly describe the sanitization technique and the empirical parameters that it requires in order to operate optimally. Indeed, to cleanse the training data for any AD sensor, we harnessed the idea of an "ensemble classifier", defined by [6] as "a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples." One option for generating such an classifier ensemble is to peruse the available training data by splitting them into smaller data sets used to train instances of the AD sensor. The inherent assumption is that *attacks and abnormalities are a minority compared to the entire set of training data*. This is certainly true for training sets that span a long period of time. Therefore, we proposed the use of *time-delimited slices* of the training data. Indeed, consider a large training data set *T* partitioned into a number of smaller disjoint subsets (micro-datasets):

$$T = \{md_1, md_2, \ldots, md_N\}, \tag{1}$$

where md_i is the micro-dataset starting at time $(i-1) * g$ and, g is the granularity for each micro-dataset.

We can now apply a given anomaly detection algorithm. We define the model function *AD* to be:

$$M = AD(T), \tag{2}$$

where *AD* can be any chosen anomaly detection algorithm, *T* is the training data set, and *M* denotes the model produced by *AD* for the given training set. This formulation enables us to maintain the stated principle of being agnostic to the inner workings of the AD sensor - we treat it as a black box whose first task is to output a normality model for a data set provided as input.

We use each of the "epochs" md_i to compute a *micro-model* $M_i = AD(md_i)$ and generate the classifier ensemble. We posit that each distinct attack will be concentrated in (or around) a certain time period, affecting only a small fraction of the micro-models: M_j computed for time period t_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. We use this ensemble classifier for identifying attacks and abnormalities in the data. Our expectation is that the ensemble will be a more efficient tool that the sum of its parts, with the effects of attacks and other abnormalities contained in individual micro-models rather than contaminating the entire data set.

A key parameter of the aforementioned sanitization method is the automatic selection of the optimal time granularity for different training data sets. Intuitively, choring a smaller value of the time granularity g always confines the effect of an individual attack to a smaller neighborhood of micro-models. However, excessively small values can lead to under-trained models that also fail to capture the *normal* aspects of system behavior. One method that ensures that the micro-models are well-trained is based on the rate at which new content appears in the training data [30]. This has the advantage of relying exclusively on intrinsic properties of the training data set. By applying this analysis, we can then identify for each md_i the time granularity that ensures a well-trained micro-model and thus attaining a balance between the two desiderata presented above.

We consider the training data set as a sequence of high-order n-grams (therefore a stream of values from a high-dimensional alphabet). When processing this data, for any time window tw_i, we can estimate the likelihood L_i of the system seeing new n-grams, and therefore new content, in the immediate future based on the characteristics of the traffic seen so far:

$$L_i = \frac{r_i}{N_i}, \tag{3}$$

where r_i is the number of *new unique* n-grams in the time window tw_i and N_i is the *total* number of *unique* n-grams seen between tw_0 and tw_i.

Assuming that the data processed by the system is not random, the value of L_i decreases much faster than the time necessary to exhaust the space of possible n-grams. We are interested in determining the stabilization point for which the number of new grams appears at a low rate, thus looking for the the knee of the curve. In order to detect the stabilization point, we use the linear least squares method over a sliding window of points (in our experiments we use 10 points) to fit a line, $L_i'(t) = a + b * t$. When the regression coefficient *b* approaches zero (0), we consider that the input has stabilized as long as the standard deviation of the likelihood is not significant. In our experiments, we discovered that we can relax the above assumptions to an absolute value lower than 0.01 for the regression coefficient *b* while the standard deviation of the likelihood is less than 0.1. The time interval between tw_0 and tw_i is then set as the desired time granularity for computing the micro-models as described above.

Our experimental corpus, used throughout the experiments in this paper, consists of 500 hours of real network traffic from each of two hosts, *www1* and *lists*. *www1* is a gateway to the homepages of students in the Computer Science Department running several dozen different scripts, while *lists* hosts the Computer Science Mailing Lists. The two servers exhibit different content, diversity and volume of data. We partitioned

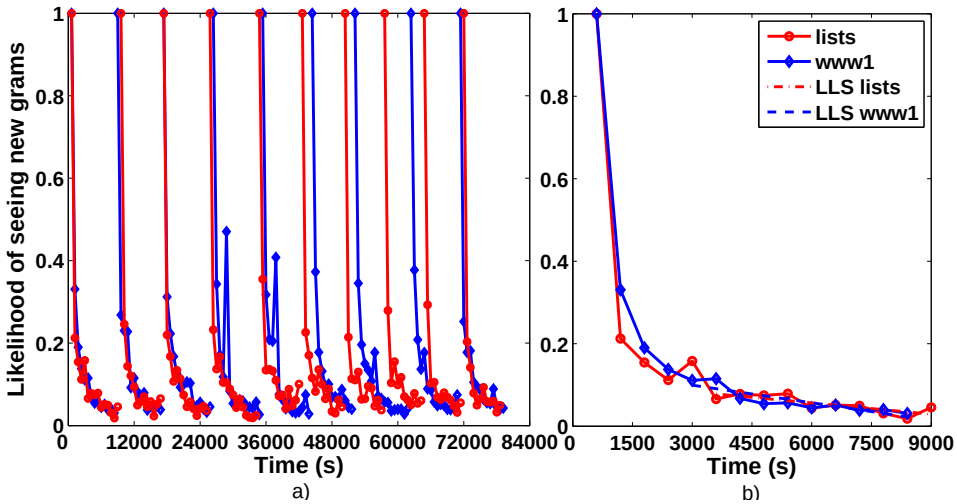


Fig. 1. Time granularity detection($|tw| = 600s$): a) first 10 micro-models (after each model, L is reset); b) zoom on the first model

the data into three separate sets: two used for training and one used for testing. The first 300 hours of traffic in each set was used to build micro-models. Figure 1 shows the granularity detection method used to characterize both data sets. Figure 1 (a) presents the time granularity for the first ten micro-models. L is reset immediately after a stabilization point is found, and we begin to generate a new model. At a first glance, both sites display similar behavior, with the level of new content stabilizing within the first few hours of input traffic. However, they do not exhibit the same trend in the likelihood distribution, L_{www1} presenting more fluctuations. Figure 1 (b) presents a zoom on the first micro-model time granularity detection. The solid lines show the evolution of the L_i likelihood metric over time(we use n-grams of size n=5). The dotted lines show the linear least squares approximation for the stabilization value of tw_i, which is used to compute the time granularity g_i.

Figure 2 illustrates the automatically generated time granularities over the first 300 hours of traffic for both *www1* and *lists*. The average value for *www1* is $g = 8562s$ ($\approx$ 2 hours and 22 minutes), while the standard deviation is $1300s$ ($\approx$ 21 minutes). For *lists* the average time granularity is $g = 8452s$ ($\approx$ 2 hours and 20 minutes), while the standard deviation is $819.8s$ ($\approx$. 13 minutes). In the next section, we will present an extensive comparison between the performance of the sanitized models that use the automated parameters versus the ones built using the empirically determined parameters.

3 Adaptive Training using Self-Sanitization

Once the micro-models are built, they can be used, together with the chosen AD sensor, as a classifier ensemble: a given network packet, which is to be classified as either

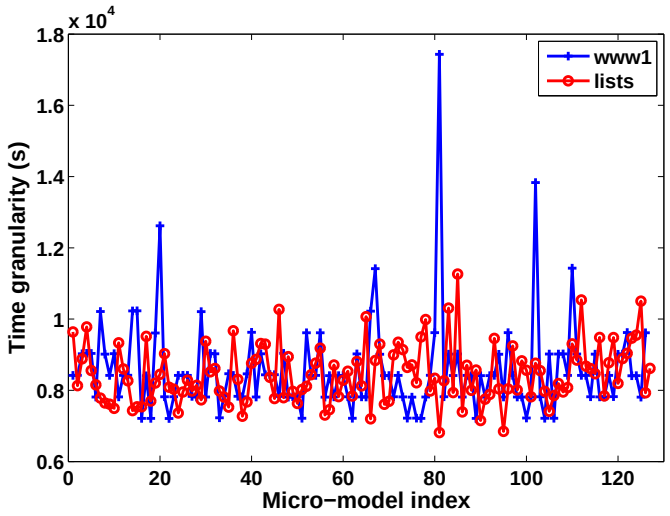


Fig. 2. Automatically determined time granularity

normal or anomalous, can be tested, using the AD sensor, against each of the micro-models. One possibility would be to apply this testing scheme to the same data set that was used to build the micro-models (we call this process *introspection*). Another option is to apply the micro-model testing to a second set of the initially available traffic, of smaller size. The ultimate goal is to effectively sanitize the training data set and thus obtain the clean training data set needed for anomaly detection.

Once again, we treat the AD sensor at a general level, this time considering a generic $TEST$ function. For a packet P_j part of the tested data set, each individual test against a micro-model results in a label marking the tested packet either as *normal* or *abnormal*:

$$L_{j,i} = TEST(P_j, M_i) \tag{4}$$

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 data set through a voting scheme, which assigns a final score to each packet:

$$SCORE(P_j) = \frac{1}{W} \sum_{i=1}^{N} w_i \cdot L_{j,i} \tag{5}$$

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. In our previous work we observed that the weighted

version performs slightly better, so throughout this paper we will use the weighted voting scheme.

The set of micro-models is now ready to be used as an overall packet classifier. Recall our assumption that only a minority of the micro-models will be affected by any given attack or anomaly. Based on the overall score assigned by the set of micro-models, we split the training data into two disjoint sets: T_{san}, containing the packets deemed as normal, and T_{abn}, containing the abnormalities/attacks:

$$T_{san} = \bigcup \{P_j \mid SCORE(P_j) \leq V\} \tag{6}$$

$$T_{abn} = \bigcup \{P_j \mid SCORE(P_j) > V\}, \tag{7}$$

where V is a *voting threshold* used to differentiate between the two sets. Next we will present our method for automatically computing the value of V that effectively provides this separation, based on the characteristics of the traffic. Once the disjoint data sets are constructed, we can apply the modeling function of the AD sensor and obtain compact representations of both normal and abnormal traffic:

$$M_{san} = AD(T_{san}) \tag{8}$$

$$M_{abn} = AD(T_{abn}) \tag{9}$$

3.1 Voting Threshold Detection

Our goal is to automatically determine the voting threshold, V. In order to establish an effective value for it, we must first analyze the impact of the voting threshold on the number of packets that are deemed normal. The extreme values have an obvious effect: a threshold of $V = 0$ (very restrictive) means that a packet must be approved by all micro-models in order to be deemed normal. In contrast, a threshold of $V = 1$ (very relaxed) means that a packet is deemed as normal as long as it is accepted by at least one micro-model. In general, for a given value V_i we define $P(V_i)$ as the number of packets deemed as normal by the classifier ($SCORE(P_j) < V_i$). The behavior of this function for intermediate values of V_i is highly dependent on the particular characteristics of the available data. For a particular data set, we can plot the function $P(V)$ by sampling the values of V at a given resolution; the result is equivalent to the *cumulative distribution of the classification scores over the entire data set*. This analysis can provide insights into three important aspects of our problem: the intrinsic characteristics of the data (number and relevance of outliers), the ability of the AD sensor to model the differences in the data, and the relevance of the chosen time granularity.

To illustrate this concept, we will use as an example the $www1$ data set and the Anagram [30] sensor. Figure 3 shows the result of this analysis for time granularity ranging from 1 to 100 hours. We notice that, as the time granularity increases, the plot "flattens" towards its upper limit: the classifier loses the ability to discriminate as the micro-models are fewer in number and also more similar between themselves. We also notice that for V very close to 1, all the plots converge to similar values; this is an indicator of the presence of a number of packets that are highly different from the rest of the data in the set.

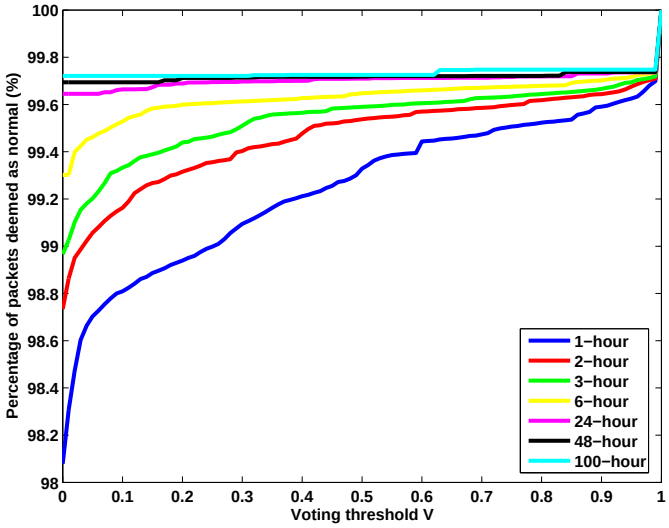


Fig. 3. Impact of the voting threshold over the number of packets deemed as normal for different time granularities

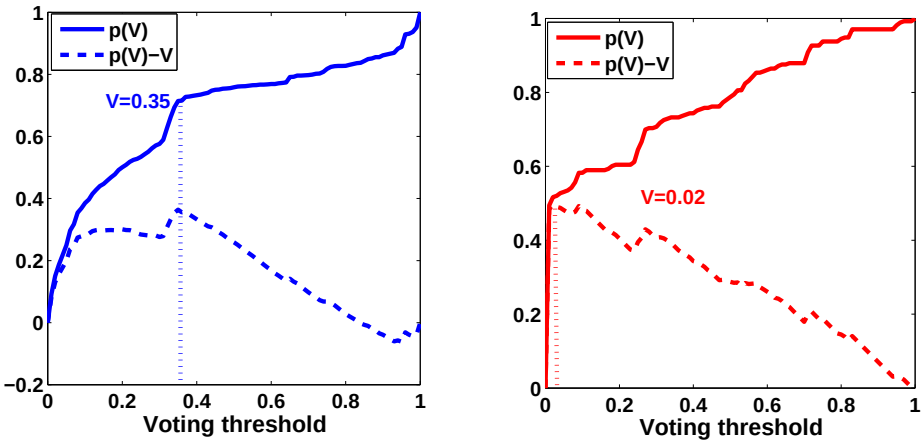
Intuitively, the optimal voting threshold V is the one that provides the best separation between the normal data class and the abnormal class. The packets that were voted normal for $V = 0$ are not of interest in the separation problem because they are considered normal by the full majority of the micro-models and the choice of V does not influence them. So the separation problem applies to the rest data for which $V > 0$; thus, we normalize $P(V)$ as follows:

$$p(V_i) = \frac{P(V_i) - P(0)}{P(1) - P(0)} \tag{10}$$

The separation problem can be now considered as the task of finding the smallest threshold (minimize V) that captures as much as possible of the data (maximize $p(V)$). Therefore, if the function $p(V) - V$ exhibits a strong global maximum, these two classes can be separated effectively at the value that provides this maximum.

We have applied this method to both data sets considered in this paper, using Anagram. The profiles of both $p(V)$ (solid lines) and $p(V) - V$ (dotted lines) are shown in Figure 4. In each case, we have marked the value of V that maximizes $p(V) - V$. In both graphs, the maximum of $p(V) - V$ corresponds to a "breaking point" in the profile of $p(V)$ (in general, any changes in the behavior of $p(V)$ are identified by local maxima or minima of $p(V) - V$). The value of the global maximum can be interpreted as a confidence level in the ability of the micro-model classifier to identify outliers, with larger values indicating a high discriminative power between the normal data and the abnormalities/attacks. A low value (and therefore a profile of $p(V)$ following the $x = y$ line) shows that the two classes are not distinct. This can be indicative of a poorly cho-

sen time granularity, an AD sensor that is not sensitive to variations in the data set, or both. We consider this to be a valuable feature for a system that aims towards fully autonomous self-calibration: failure cases should be identified and reported to the user rather than silently accepted.



(a) (b)

Fig. 4. Determining the best voting threshold for: (a) *www1*; (b) *lists*.

Once the value of the voting threshold V has been determined, the calibration process is complete. We note that all the calibration parameters have been set autonomously based exclusively on observable characteristics of the training data. The process can therefore be seen as a method for characterizing the combination of AD sensor - training data set, and evaluating its discriminative ability.

3.2 Analysis of Self-Sanitization Parameters

To evaluate the quality of the models built using the automatically determined sanitization parameters, we compare their performance against the performance of the sanitized models built using empirically determined parameters. There is a fundamental difference between the two types of models: for the first one the sanitization process is completely hands-free, not requiring any human intervention, while for the latter, exhaustive human intervention is required to evaluate the quality of the models for different parameter values and then to decide on the appropriate parameter values.

There are two parameters of interest in the sanitization process: the set of values for the time granularity and the voting threshold. We will therefore compare the models built using empirically determined parameters against the models built using:

- a fixed time granularity and automatically determined voting threshold;
- automatically determined time granularities and fixed voting threshold;

– both time granularity and voting threshold determined automatically.

Figures 5 and 6 present the false positive and detection rates for models built using different sanitization parameters. The traffic contains instances of phpBB forum attacks (mirela, cbac, nikon, criman) for both hosts that are analyzed.[1] Each line shows the results obtained as the voting threshold was sampled between 0 and 1, with the granularity value either fixed at a given value (usually 1, 3 or 6 hours) or computed automatically using the method described earlier.

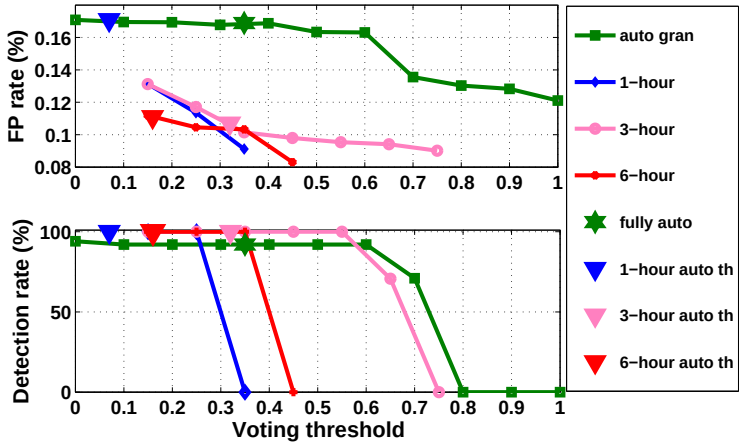


Fig. 5. Model performance comparison for *www1*: automated vs.empirical

We note that the time granularity values empirically found to exhibit high performance were 1-, 3- and 6-hour for *www1*, respectively 3-hour for *lists*. For each of these values, we analyzed the performance of the models built with an automatically determined voting threshold. For each line representing a given granularity value, the triangular markers represent the results obtained with the automatically determined voting threshold. We observe that the voting threshold is placed in the safety zone for which the 100% detection rate is maintained for both *www1* and *lists*, while exhibiting a low false positive rate ($< 0.17\%$).

In the case of automated time granularity (the actual values are presented in figure 2), we initially explored the performance of the models determined for different values of the voting threshold, ranging from 0 to 1, with a step of 0.1. In figure 5, for the same fixed threshold, the detection rate is 94.94% or 92.92% compared to the 3-hour granularity (empirical optimal - 100%), while maintaining a low false positive rate

[1] Throughout the paper, we refer to detection and false alert 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.

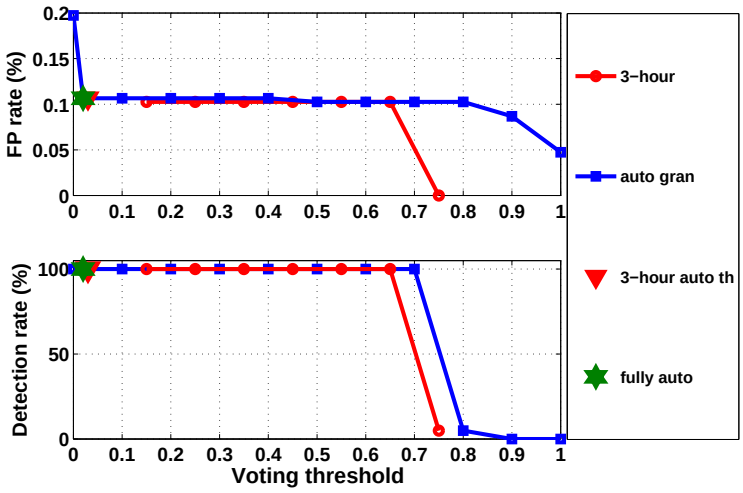


Fig. 6. Model performance comparison for *lists*: automated vs.empirical

($< 0.17\%$). In figure 6, the results are almost identical to the empirically determined optimal (3-hour granularity).

Table 1. Empirically vs. automatically determined parameters

Parameters	www1		lists	
	FP(%)	TP(%)	FP(%)	TP(%)
N/A(no sanitization)	0.07	0	0.04	0
empirical	0.10	100	0.10	100
fully automated	0.16	92.92	0.10	100

When we use both the set of time granularities and the voting threshold determined automatically, the system is fully autonomous. In figures 5 and 6, this is indicated by replacing the triangular marker with a star-shaped one. Table 1 also summarizes the values of false positive (FP) and true positive (TP) for the fully automated sanitized model, the empirical optimal sanitized model and the non-sanitized model. With automated parameters, for *lists* we achieve the same values as in the case of empirically determined parameters, while for *www1* the values differ, but we observe that in the absence of the sanitization process the detection rate would be 0. The most important aspect is that the fully-automated sanitization still significantly improves the quality of the AD models while setting its parameters based only on the intrinsic characteristics of the data and without any user intervention.

4 Self-Updating Anomaly Detection Models

We presented a method that generates automatically self-sanitized AD models. However, the way users interact with systems can evolve over time [9], as can the systems themselves. As a result, the AD models that once represented the normal behavior of a system can become obsolete over time. Therefor, the models need to adapt to this phenomenon, usually referred to as *concept drift*. As shown in [18], online learning can accommodate changes in the behavior of computer users. Here, we also propose to use an online learning approach to cope with the concept drift, in the absence of ground truth.

Our approach is to continuously create micro-models and sanitized models that incorporate the changes in the data. An aging mechanism can be applied in order to limit the size of the ensemble of classifiers and also to ensure that the most current data is modeled. When a new micro-model, μM_{N+1} is created, the oldest one, μM_1, is no longer used in the voting process (see figure 7). The age of a model is given by the time of its creation.

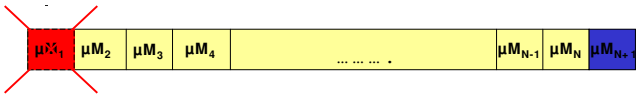


Fig. 7. Incremental learning aging the oldest micro-model

Every time a new micro-model is generated, a new sanitized model is created as well. In the previous section, we used the micro-models in a voting scheme on a second data set, which was processed into a sanitized and an abnormal model. For the online sanitization we will use what we call *introspection*: the micro-models are engaged in a voting scheme against their own micro-datasets[2]. This alternative gives us the ability to apply the self-sanitization processes in an online fashion, without having to also maintain a second dataset strictly for model creation. When a new sanitized model is built, it is immediately used for testing the incoming traffic until a new sanitized model is built.

Concept drift appears at different time scales and our micro-models span a particular period of time. Thus, we are limited in observing drift that appears at scales that are larger than the time window covered by the micro-datasets. Any changes that appear inside this time window are susceptible to being rejected by the voting process rather than being accepted as legitimate evolution of the system. In our online sanitization experiments we use 25 classifiers in the voting process (covering $\approx$ 75 hours of real time traffic) such that we can adapt to drifts that span more than 75 hours of traffic.

We cannot distinguish between a legitimate change and a long-lasting attack that slowly pollutes the majority of the micro-models. A well-crafted attack can potentially

[2] We recall that we define a micro-dataset as the training dataset used for building a micro-model.

introduce malicious changes at the same or even smaller rate of legitimate behavioral drift. As such, it can not be distinguished using strictly introspective methods that examine the characteristics of traffic. However, the attacker has to be aware, guess, or brute-force the drift parameters to be successful with such an attack. In previous work [4], we presented a different type of information that can be used to break this dilemma: alert data from a network of collaborative sites. Another potential solution that we intend to explore as future work, is to employ as feedback information the error responses returned by the system under protection (*e.g. the HTTP reply as an error page*). We plan to explore the conjecture that we can indeed ferret out attacks of certain classes by observing the error responses returned from different sub-systems or software modules.

4.1 Self-Update Model Evaluation

To illustrate the self-update modeling, we first apply the online sanitization process for the first 500 hours of traffic using Anagram as the base sensor. Figures 2 and 8 present the fully automated sanitization parameters: the time granularity for each micro-model used in the creation of the new sanitized models, respectively the voting threshold for each newly created sanitized model.

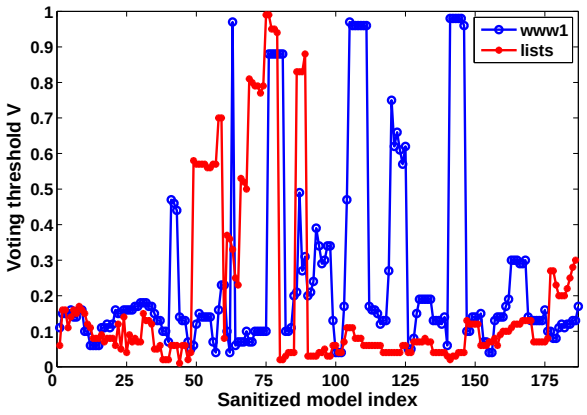
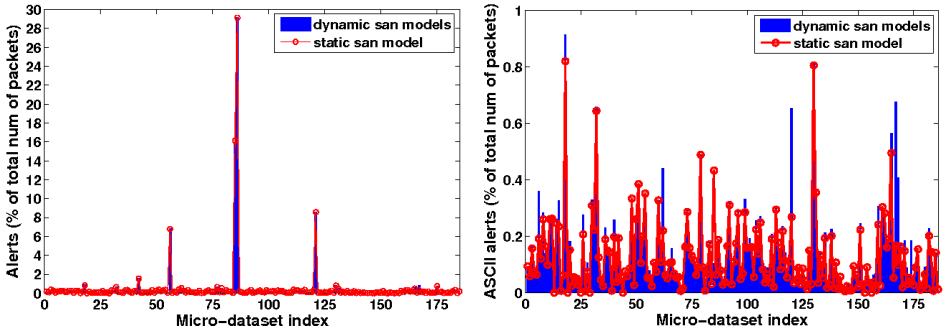


Fig. 8. Automatically determined voting threshold for *www1* and *lists*

If we didn't employ a model update mechanism, a sanitized model would be built only once. Thus, we call the first sanitized model a *static sanitized model*. Because in the online sanitization process, the models change continuously we consider them *dynamic sanitized models*. To analyze how the online sanitization performs, in figure 9 we compare the static sanitized model alert rate against the dynamic sanitized models alert rate for *www1*.

Figure 9 (a) presents the total number of alerts for each micro-dataset tested with both the static and dynamic models. We first notice that, for a few micro-dates the alert



(a) (b)

Fig. 9. Alert rate for *www1*: (a) both binary and ascii packets; (b) ascii packets.

rate reaches levels up to 30% for both model types. After analyzing the alert data, we determined that the high alert rate was generated not by abrupt changes in the system's behavior, but rather by packets containing binary media files with high entropy. This type of data would be considered anomalous by AD sensors such as Anagram. Thus the recommendation is to divert all the media traffic to specialized detectors which can detect malicious content inside binary media files. Figure 9 (b) presents the alert rate after ignoring the binary packets. We can observe that there is no significant difference between the alert rate exhibited by the static and dynamic sanitized models. Thus we can conclude that there are no fundamental changes over the 500 hour period.

In terms of performance, table 2 presents both the false positive rate (including the binary packets) and the detection rate for *www1* and *lists*. Abrupt changes in the voting threshold (as shown in figure 8) determine the creation of more restrictive models, thus the increase in the detection rate and/or the false positive rate. For *www1* the signal-to-noise ratio (*i.e.* TP/FP) is improved from 155.21 to 158.66, while for *lists* it decreases from 769.23 to 384.61.

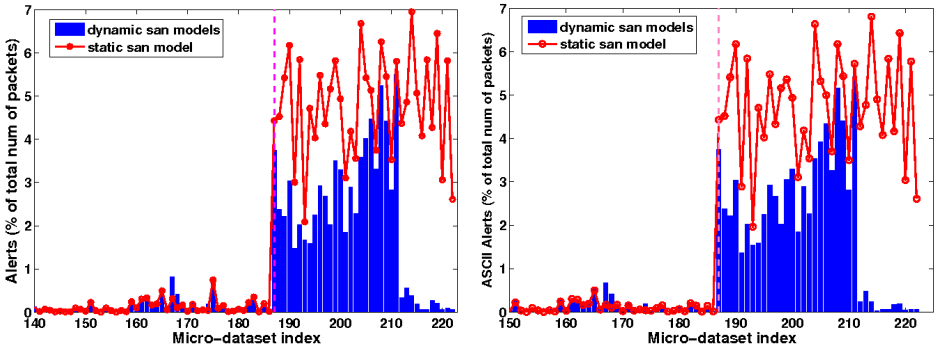
Table 2. Static model vs. dynamic models alert rate

Model	www1		lists	
	FP(%)	TP(%)	FP(%)	TP(%)
static model	0.61	94.68	0.13	100
dynamic models	0.62	98.37	0.26	100

We also investigated concept drift appearing at larger scale such as weeks and months, as opposed to days. For this, we tested our method for traffic from the same site, collected at months difference. Figure 10 presents the alert rate for both static and dynamic models, with and without the binary packets. Vertical lines mark the boundary between new and old traffic. We can observe that when changes happen in the system, the alert rate increases for both static and dynamic models. After the dynamic models

start updating to the new data, there is a drop in the alert rate, back to levels below 1%. For the static model, the alert rate stays at about 7%, demonstrating the usefulness of a self-updating sanitization process.

Figure 11 presents the raw number of alerts that our system returns on an hourly basis. We note that spikes in the number of alerts can render manual processing difficult, especially when there are changes in the system under protection and the models gradually adapt to the new behavior. However, manual processing of alerts is not the intended usage model for our framework; our ultimate goal is to build a completely hands-free system that can further identify the true attacks from the false positives. In previous work [4] we have proposed using a shadow sensor architecture such as the ones presented in [1, 22] to automatically consume and validate the false positives. Our study of computational performance presented in [4] shows that, with this architecture, the false positives can be consumed automatically and neither damage the system under protection nor flood an operational center with alarms.



(a) (b)

Fig. 10. Concept drift detection for *www1* - alert rate for (a) both binary and ascii packets; (b) ascii packets. Vertical lines mark the boundary between new and old traffic

4.2 Computational Performance Evaluation

To investigate the feasibility of our online technique we have to analyze the computational overhead that it implies. Ignoring the initial effort of building the first batch of micro-models and the sanitized model, we are interested in the overhead introduced by the model update process. Table 3 presents a breakdown of the computational stages of this process.

The overhead has a linear dependency on the number and the size of the micro-models. For *www1*, we used 25 micro-models per sanitization process and the size of a micro-model was on average 483 KB (trained on 10.98 MB of HTTP requests). The experiments were conducted on a PC with a 3GHz Intel(R) Xeon(R) CPU with 4 cores

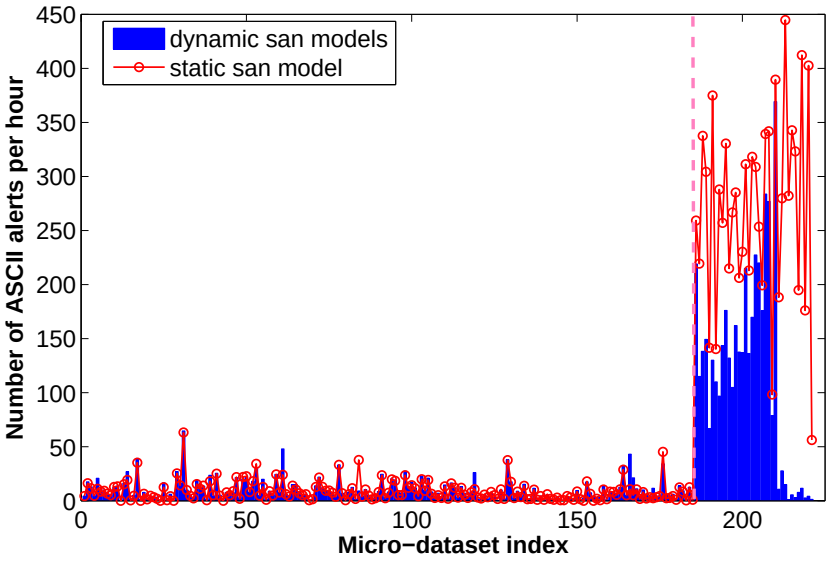


Fig. 11. Number of ASCII alerts per hour for *www1*. The vertical line marks the boundary between new and old traffic

Table 3. Computational performance for the online automated sanitization for *www1*

Task	Time to process
build and save a new micro-model	7.34 s
test its micro-dataset against the older micro-models	1 m 12 s
test the old micro-datasets against the new micro-model	1 m 58 s
rebuild and save the sanitized model	3 m 03 s

and 16G of RAM, running Linux. This level of performance is sufficient for monitoring and updating models on the two hosts that we tested in this paper, as it exceeds the arrival rate of HTTP requests. In the case of hosts displaying higher traffic bandwidth, we can also exploit the intrinsic parallel nature of the computations in order to speed up the online update process: multiple datasets can be tested against multiple models in parallel, as the test for each dataset-model pair is an independent operation. In future work, we will implement a parallel version of this algorithm to test these assumptions.

5 Related work

We have previously explored the feasibility of sanitizing training datasets using empirically determined parameters [4, 5]. This paper presents methods that make the process automatic, by generating the sanitization parameters based only on the intrinsic characteristics of the data and by also coping with concept drift. The sanitization process can

be viewed as an ensemble method [6] with the restriction that our work is an unsupervised learning technique. We generate AD models from slices of the training data, thus manipulating the training examples presented to the learning method. Bagging predictors [2] also use a learning algorithm with a training set that consists of a sample of *m* training examples drawn randomly for the initial data set. 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.

MetaCost [7] 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. JAM [27] 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]. For more details on meta-learning techniques we can also refer the reader to a more comprehensive survey [23].

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 [8] have shown how to evade anomaly classifiers by constructing polymorphic exploits that blend with normal traffic (a sophisticated form of mimicry attack [28]), and Song *et al.* [26] 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 [19].

The problem of determining anomaly detection parameters have been studied before. Anagram [30] determines the model stability automatically based on the rate at which new content appears in the training data. pH [24] proposes heuristics for determining an effective training time, minimizing the human intervention as well. Payl [29] has a calibration phase for which a sample of test data is measured against the centroids and an initial threshold setting is chosen. The thresholds are updated throughout a subsequent round of testing. In [17], the authors propose a web-based anomaly detection mechanism, which uses a number of different models to characterize the parameters used in the invocation of the server-side programs. For these models, dynamic thresholds are generated in the training phase, by evaluating the maximum score values given on a validation dataset. PCA-based techniques for detecting anomalous traffic in IP networks became popular in the past years. [21] talks about the difficulty of tuning the parameters for these techniques and discusses pollution of the normal subspace.

The concept of updating an AD sensor in order to mirror valid changes in the protected system's behavior is discussed in [18]. Most publications which propose updating the model after significant changes to the environment, data stream, or application use supervised learning techniques, such as [12]. Methods of this type maintain an adap-

tive time window on the training data [14], select representative training examples [13], or weigh the training examples [15]. The key idea is to automatically adjust the window size, the example selection, and the example weighting, respectively, so that the estimated generalization error is minimized. Consequently, these methods assume the existence of labeled data which is not the case for the applications that we interested in analyzing. It seems that anomaly detectors would benefit from an additional source of information that can confirm or reject the initial classification, and Pietraszek [20] suggests using human–supervised machine learning for such tuning.

6 Conclusions and Future Work

Anomaly detection sensors have become an integral part of the network and host-based defenses both for large-scale network and individual users. Currently, AD sensors require human operators to perform initial calibration of the training parameters to achieve optimal detection performance and minimize the false positives. In addition, as the protected system evolves over time, the sensor's internal state becomes more and more inconsistent with the protected site. This discrepancies between the initial normality model and the current system behavior eventually renders the AD sensor unusable.

To amend this, we propose a fully automated framework that allows the AD sensor to adapt to the characteristics of the protected host during the training phase. Furthermore, we provide an online method to maintain the state of the sensor, bounding the deviations due to content or behavioral modifications that are consistent over a period of time. Without this adaptation process and the generation of new normality models which we call "dynamic", legitimate changes in the systems are flagged as anomalous by the AD sensor leading to an inflation of alerts. Our experimental results show that, compared to the manually obtained optimal parameters, the fully automated calibration has either identical, or slightly reduced (by 7.08%) detection rate and a 0.06% increase in false positives. Furthermore, over a very large time window, our dynamic model generation maintains a low alert rate (1%) as opposed to a 7% for a system without updates.

We believe that our system can help alleviate some of the challenges faced as anomaly detection is increasingly relied upon as a first-class defense mechanism. AD sensors can help counter the threat of zero-day and polymorphic attacks; however, the reliance on user input is a potential roadblock to their application outside of the lab and into commercial off-the-shelf software. In this paper we have taken a number of steps towards AD sensors that enable true hands-free deployment and operation.

In the future, we intend to establish this feature of our framework by using more sensors, that either model data in a different way (*e.g.* Payl [29], libanomaly [17], Spectrogram [25]) or target different applications (*e.g.* pH [24]). Despite the best efforts of the research community, no AD sensor has been proposed to date that can detect all attack types while maintaining a low alert rate. A possible option, which we intend to further explore in the future, is to combine the strengths of multiple sensors under a general and unified framework, following the directions traced out in this study.

Finally, the methods presented harness the information contained in the traffic (or behavior in general) of the protected host. Large-scale implementations of AD systems can further benefit by exchanging data, such as micro-models or sanitized and abnormal

models, across different sites. Therefore, the temporal dimension of our online sanitization process can be complemented by a spatial one. We are currently in the process of establishing an information exchange framework that can facilitate these experiments; we plan to report these result in a future study.

Acknowledgments

This material is based on research sponsored by the Air Force Office of Scientific Research AFOSR MURI under contract number GMU 107151AA, by the National Science Foundation under NSF grants CNS-06-27473 and by Google INC. 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. Anagnostakis, K.G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., Keromytis, A.D.: Detecting Targeted Attacks Using Shadow Honeypots. In: Proceedings of the 14^{th} USENIX Security Symposium (2005)
2. Breiman, L.: Bagging Predictors. Machine Learning **24**(2), 123–140 (1996)
3. Chan, P.K., Stolfo, S.J.: Experiments in Multistrategy Learning by Meta-Learning. In: Proceedings of the second international conference on information and knowledge management, pp. 314–323. Washington, DC (1993)
4. Cretu, G.F., Stavrou, A., Locasto, M.E., Stolfo, S.J., Keromytis, A.D.: Casting out Demons: Sanitizing Training Data for Anomaly Sensors. In: In the Proceedings of the IEEE Symposium on Security and Privacy (2008)
5. Cretu, G.F., Stavrou, A., Stolfo, S.J., Keromytis, A.D.: Data Sanitization: Improving the Forensic Utility of Anomaly Detection Systems. In: Workshop on Hot Topics in System Dependability (HotDep) (2007)
6. Dietterich, T.G.: Ensemble Methods in Machine Learning. Lecture Notes in Computer Science **1857**, 1–15 (2000)
7. Domingos, P.: Metacost: A general method for making classifiers cost-sensitive. In: Knowledge Discovery and Data Mining, pp. 155–164 (1999)
8. Fogla, P., Lee, W.: Evading Network Anomaly Detection Systems: Formal Reasoning and Practical Techniques. In: Proceedings of the 13^{th} ACM Conference on Computer and Communications Security (CCS), pp. 59–68 (2006)
9. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A Sense of Self for Unix Processes. In: IEEE Symposium on Security and Privacy (1996)
10. Forrest, S., Somayaji, A., Ackley, D.: Building Diverse Computer Systems. In: Proceedings of the 6^{th} Workshop on Hot Topics in Operating Systems, pp. 67–72 (1997)
11. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: European Conference on Computational Learning Theory, pp. 23–37 (1995)
12. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In: In XVII Brazilian Symposium on Artificial Intelligence (2004)
13. Klinkenberg, R.: Meta-learning, model selection, and example selection in machine learning domains with concept drift. In: Learning – Knowledge Discovery – Adaptivity (2005)

14. Klinkenberg, R., Joachims, T.: Detecting concept drift with support vector machines. In: the Proceedings of the 17th Int. Conf. on Machine Learning (2000)
15. Klinkenberg, R., Ruping, S.: Concept drift and the importance of examples. In: Franke, J., Nakhaeizadeh, G., Renz, I., eds.: Text Mining Theoretical Aspects and Applications (2003)
16. Kruegel, C., Toth, T., Kirda, E.: Service Specific Anomaly Detection for Network Intrusion Detection. In: Symposium on Applied Computing (SAC). Madrid, Spain (2002)
17. Kruegel, C., Vigna, G.: Anomaly Detection of Web-based Attacks. In: ACM Conference on Computer and Communication Security. Washington, D.C. (2003)
18. Lane, T., Broadley, C.E.: Approaches to online learning and concept drift for user identification in computer security. In: 4th International Conference on Knowledge Discovery and Data Mining (1998)
19. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically Generating Signatures for Polymorphic Worms. In: IEEE Security and Privacy. Oakland, CA (2005)
20. Pietraszek, T.: Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID) (2004)
21. Ringberg, H., Soule, A., Rexford, J., Diot, C.: Sensitivity of pca for traffic anomaly detection. In: SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 109–120. ACM, New York, NY, USA (2007). DOI http://doi.acm.org/10.1145/1254882.1254895
22. Sidiroglou, S., Locasto, M.E., Boyd, S.W., Keromytis, A.D.: Building a Reactive Immune System for Software Services. In: Proceedings of the USENIX Technical Conference (2005)
23. Smith-Miles, K.: Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Comput. Surv. **41**(1) (2008). URL `http://dblp.uni-trier.de/db/journals/csur/csur41.html#Smith-Miles08`
24. Somayaji, A., Forrest, S.: Automated Response Using System-Call Delays. In: Proceedings of the 9^{th} USENIX Security Symposium (2000)
25. Song, Y., Keromytis, A.D., Stolfo, S.J.: Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic. In: Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS) (2009)
26. Song, Y., Locasto, M.E., Stavrou, A., Keromytis, A.D., Stolfo, S.J.: On the Infeasibility of Modeling Polymorphic Shellcode. In: ACM Computer and Communications Security Conference (CCS) (2007)
27. Stolfo, S., Fan, W., Lee, W., Prodromidis, A., 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)
28. Wagner, D., Soto, P.: Mimicry Attacks on Host-Based Intrusion Detection Systems. In: ACM CCS (2002)
29. Wang, K., Cretu, G., Stolfo, S.J.: Anomalous Payload-based Worm Detection and Signature Generation. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID) (2005)
30. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In: Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID) (2006)
31. Wolpert, D.: Stacked Generalization. In: Neural Networks, vol. 5, pp. 241–259 (1992)