Data Sanitization: Improving the Forensic Utility of Anomaly Detection Systems

Gabriela F. Cretu, Angelos Stavrou, Salvatore J. Stolfo and Angelos D. Keromytis
Department of Computer Science, Columbia University
{gcretu, angel, sal, angelos}@cs.columbia.edu

Abstract

Anomaly Detection (AD) sensors have become an invaluable tool for forensic analysis and intrusion detection. Unfortunately, the detection accuracy of all learning-based ADs depends heavily on the quality of the training data, which is often poor, severely degrading their reliability as a protection and forensic analysis tool. In this paper, we propose extending the training phase of an AD to include a sanitization phase that aims to improve the quality of unlabeled training data by making them as "attack-free" and "regular" as possible in the absence of absolute ground truth. Our proposed scheme is agnostic to the underlying AD, boosting its performance based solely on training-data sanitization. Our approach is to generate multiple AD models for content-based AD sensors trained on small slices of the training data. These AD "micro-models" are used to test the training data, producing alerts for each training input. We employ voting techniques to determine which of these training items are likely attacks. Our preliminary results show that sanitization increases 0-day attack detection while maintaining a low false positive rate, increasing confidence to the AD alerts. We perform an initial characterization of the performance of our system when we deploy sanitized versus unsanitized AD systems in combination with expensive host-based attack-detection systems. Finally, we provide some preliminary evidence that our system incurs only an initial modest cost, which can be amortized over time during online operation.

1 Introduction

A network-based intrusion detector can be used as an online traffic/input-filtering subsystem, or as a forensic tool to identify likely data that created a fault in a system after the fact. As "signature-based" network intrusion detection systems (NIDS) appear to become obsolete in detecting zero-day malicious traffic, effective anomaly detection that models normal traffic well remains an open problem. Ideally, an anomaly detector should achieve 100% detection accuracy, *i.e.,* true attacks are all identified, with 0% false positives.

However, the particular modeling algorithm one uses to compute a model of "normal" data can fail for various reasons. In particular, for unsupervised AD systems, we do not have a ground truth to compare to and verify our testing results. An attack contained in the training data would "poison" the normal model, rendering the AD system incapable of detecting future instances of this or similar attacks. The danger of having an AD system with false negatives becomes a limiting factor of the size of the training set [10]. Even in the presence of ground truth or supervised/manual training, creating a single model of normal traffic which includes all non-attack traffic can result in under-fitting and over generalization. Under-fitting can lower the detection capabilities of an AD sensor, thereby increasing false negatives.

We propose a new forensic/defense strategy that introduces a novel sanitization phase. Our goal is to remove both attacks and non-regular traffic from the training data, improving the accuracy of the training and increasing confidence in the alerts generated by the anomaly detector.

Our approach is two-pronged: initially, we modify the training phase of an AD system. Instead of using a normal model generated by a single AD sensor trained on a single large set of data, we employ multiple AD instances trained on smaller time-based slices of the same data set. We call such normality models *micro-models*. These micro-models are a very localized view of the training data. Armed with the micro-models, we can assess the quality of the training data and automatically detect and remove any attacks or anomalies 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 that spans a suffi-

ciently large time interval, an attack or other anomaly will appear only in a few of the time slices. To identify these anomalies, we test each packet of the training data set against the produced AD micro-models. Using a voting scheme, we determine which of the packets can be consider as abnormal and need to be removed from the 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 containing packets that are closer to what we consider the "normal model" of the application. Using this sanitized training set, we are able to generate a single *sanitized model* from a single AD instance. This model has been generated using a sanitized data set that contains fewer attacks, and it therefore improves detection performance when used during the operational phase. We establish evidence for this conjecture in Section 3. Note, however, that the generated model might have an increased false positive rate, since rare but legitimate events will be removed from the data set during the sanitization phase.

To mitigate the effects of false positives, we use a heavily instrumented host-based "shadow" server system akin to a honeypot [1] that can determine whether a packet is a true attack. By diverting all suspect data to this oracle, we will be able to identify true attacks by way of detecting malicious or abnormal actions performed by the server when processing the suspicious data. Note that such a heavily instrumented shadow server is assumed to be (and in practice *is*) substantially slower (usually orders of magnitude slower) than the native application to be protected (another approach would be to correlate the alert with input from another anomaly detector [6]).

Note that we do not claim to have solved the false positive problem; rather, we pose a different performance objective for anomaly detectors, *i.e.,* that within realistic operational environments, the key objective is to optimize the security and performance throughput of the system under protection. Our target is to limit the amount of network traffic that would be processed by the shadow server, and to identify the normal, attack-free data that can processed by the native service or application without instrumentation.

2 Sanitization Architecture

Supervised training using labeled datasets appears to be an ideal cleaning process [2, 5, 7]. Unfortunately, the size and complexity of the training datasets obtained from real-world network traces makes such labeling infeasible. In addition, semi- or even unsupervised training [8] using an automated process or an oracle is computationally demanding and may lead to an augmented and under-trained normality 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 are data patterns or traffic which are not attacks and at the same time appear infrequently or for a very short period of time. We seek to amend this by introducing 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.

Towards this goal, 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 may not be the case in some circumstances, *e.g.,* during a DDoS attack or during the propagation phase of a particularly virulent 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 future research. Furthermore, although we cannot predict the time when an attack appears in a training set, the attack itself usually does 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. 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.

On the other hand, using a large training set increases the probability that an individual datum appears normal: the datum has more appearances in the data set and, consequently, the probability to be considered "normal" increases. Moreover, increasing the amount of training data can significantly increase the presence of malcode in the dataset. To mitigate that, we use micro-models in an ensemble arrangement[3]: each model processes the training data individually limiting potential malware poisoning of the training resisting data to a small subset of the micro-models. Our algorithm operates in two stages: the first computes a number of micro-models over the training data, and the second phase computes a "sanitized" model and an "abnormal" model by running another training dataset

through the generated micro-models.

2.1 Micro-models

Based on our observations, if we partition a large training dataset into a number of smaller, time-delimited training sets, we may compute a minority set of partitions (micro-datasets) that contain attack vectors (not known *a priori*).

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

where md_i is the micro-dataset starting at time $(i-1)*g$ and, g is the granularity for each micro-dataset. We define the model function AD:

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

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" from Equation (1) to compute a *micro-model*, M_i.

$$M_i = AD(md_i) \tag{3}$$

Given our assumptions, we expect a distinct attack that is concentrated in (or around) time period t_j to only affect a small subset of the 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. Additional experimentation is needed to further understand the impact of different values of g on the accuracy of the AD.

2.2 Sanitized and Abnormal Models

In the second phase, we compute a new AD model using the set of previously computed micro-models to sanitize either the same or a second set of training data. The choice of splitting the training dataset into two sets represents the worst case scenario: a very large dataset for building the micromodels is needed and there is a storage limitation. Hence, the AD sensor is required to generate the micro-models online using a fraction of the necessary storage (the models are smaller than the raw traffic). After generating the micro-models, a second dataset is tested (online or offline) by all of the micro-models M_i. Each test results in a new labeled data set with every packet P_j labeled as *normal* or *abnormal*:

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

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.

At this point, we observe that 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:

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

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. Interesting avenues for future research can explore other weighting strategies.

Let us 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. Our conjecture, however, holds that only a minority of the micro-models includes the same attack vector as M_i. Thus, we use the results of 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, from which we construct a model of abnormal data, M_{abn}.

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

$$M_{san} = AD(T_{san}) \tag{7}$$

$$T_{abn} = \bigcup \{P_j \mid SCORE(P_j) > V\} \tag{8}$$

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

where V is a voting threshold. In the case of simple (that is, unweighted) voting, V relates directly to 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 our approach can help generate sanitized models for a wide range of anomaly detection systems. As a preliminary form of support for this hypothesis, we evaluate our approach with two AD sensors in Section 3.

3 Evaluation

In this section we provide some preliminary experimental results that seek to quantify the increase in the detection accuracy of out-of-the-box content-based anomaly detection systems when we apply training data sanitization. The goal of our system is to detect all true attacks and at the same time maintain or even reduce the total number of generated alerts. We evaluate our approach using two different scenarios. In the first scenario, we measure the detection accuracy 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" 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 [12] and Payl [11]. Both detectors, like most anomaly detection sensors, have a training and a testing phase. Although dependent on a clean initial model, 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 these detectors and their algorithms, as they are not germane to the topic of this paper. We refer the interested reader to the respective publications.

Our experimental corpus consists of 500 hours of real network traffic, which translates into approximately four million packets. We split 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) were used for testing the ADs. In addition, to validate our results, we used the last 100 hours to generate the sanitized model while testing on the other 100-hour dataset.

3.1 Experimental Results

In our initial experiment, we measured the detection performance for both Anagram and Payl when used as stand-alone online anomaly detectors. We then repeated the same experiments using the same setup and network traces but including the sanitization phase. Table 1 presents our findings, which show that by using a sanitized training dataset, we boost the detection capabilities of both AD sensors. The results summarize the false positive (FP) and the true positive (TP), and also the absolute numbers of false alerts (FA) and true alerts (TA) rates as averaged values obtained when using both voting techniques, a granularity of 3-hour and for the range of values V which maximizes our performance (in our case $V \in [0.15, 0.45]$). These results show that we can maximize the detection of the real alerts while generating very low false positives rates. It is important to notice that without sanitization, the normal models used by Anagram were poisoned with attacks and thus unable to detect new attack instances appearing in the testing data. Therefore, making the AD sensor more sensitive, *e.g.,* changing its internal detection threshold, would only increase the false alerts without increasing the detection rate. In this experiment, the traffic contains instances of *phpBB* forum attacks (*mirela, cbac, nikon, criman*).

Table 1. AD sensors comparison

Sensor	FP (%)	FA	TP(%)	TA
Anagram	0.07	544	0	0
Anagram with Snort	0.04	312	20.20	20
Anagram with Sanitization	**0.10**	776	**100**	99
Payl	0.84	6,558	0	0
Payl with Sanitization	**6.64**	**70,392**	**76.76**	**76**

When the worm packets are included in the normal model, we achieve a detection rate of 0% with Anagram. When using previously known malcode information (using Snort signatures represented in a "malicious 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. Worse yet, 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. The detection capabilities of the system rely on the accuracy of the sanitized model and for that reason, once the sanitized and the abnormal models are computed, n-grams that are considered abnormal are removed from the sanitized model.

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 [4]. Our traces contain such attacks, which is the reason why, when we use the sanitized version of Payl, we can only get a 76.76% worm detection rate, instead of 100%. The sanitization phase is a necessary requirement 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 ratio imply better results.

Table 2. AD sensors comparison

Sensor	TP/FP
Anagram	0
Anagram with Snort	505
Anagram with Sanitization	**1000**
Payl	0
Payl Sanitization	**11.56**

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 reenforced 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 Performance Evaluation

Another aspect of an anomaly detection system that we would like to analyze is its impact on the average time that it takes to process a request. In addition, we measure the overall computational requirements of a detection system consisting of an AD sensor and a host-based sensor (shadow server). In this configuration, 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 intended service. Our goal is to create a system that does not impose a 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.

In our experimental setup, we used two well-known instrumentation frameworks: STEM [9] and DYBOC [1]. STEM exhibits a 4,400% overhead when an application such as Apache is completely instrumented to detect attacks. On the other hand, DYBOC (which requires access to application source code for the instrumentation) has a lighter instrumentation, providing a faster response, but still imposes at least a 20% overhead on the server performance.

To calculate the total overhead, we used the same method used by Wang *et al.* [12]. We define the latency of such an architecture as following: $l' = (l*(1-fp))+(l*O_s*fp)$, where l is the standard (measured) latency of a protected service, O_s is the shadow server overhead, and fp is the AD false positive rate.

To quantify the performance loss/gain from using the sanitization phase, we compare the average latency of the system when using Payl and Anagram with sanitized and non-sanitized training data. From Table 3, we observe that for both sensors there is not a significant increase in the alert rate after sanitizing the training data.

Table 3. Latency of the anomaly detectors

Sensor	STEM	DYBOC
No-sensor	$44*l$	$1.2*l$
Anagram	$1.031*l$	$1.0001*l$
Anagram with Snort	$1.0172*l$	$1.0000*l$
Anagram with sanitization	$\mathbf{1.0430*l}$	$\mathbf{1.0002*l}$
Payl	$1.3612*l$	$1.0016*l$
Payl with sanitization	$\mathbf{3.8552*l}$	$\mathbf{1.0132*l}$

4 Future work

A weakness of the local sanitization architecture arises in the presence of a long-lasting attack in the initial set of training data, poisoning all the micro-models. To counter such attacks, we propose as future work to extend our sanitization scheme to allow

sharing of models of malicious traffic generated by collaborating remote sites, and to sanitize the local training data to a greater extent. These models, which can be privacy-preserving, capture characteristics of malicious behavior (rather than normal behavior, which is the default AD operation). This approach may not apply to polymorphic attacks, since each propagation attempt will display a distinct attack vector that may be captured in different malicious 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. However, additional testing is needed to determine how well our scheme (with or without the collaborative sanitization extensions) can cope with polymorphism or long-term training attacks.

5 Conclusions

We introduce a novel sanitization method that boosts the performance of out-of-the-box anomaly detectors, elevating them to a first-rate and *dependable* forensics and alert analysis tool. Our approach is simple and general, and can be applied to a wide range of unmodified AD sensors without incurring significant additional computational cost other than the initial testing phase. Preliminary experimental results indicate that our system can serve as an efficient and accurate online packet classifier. The alerts generated by the "sanitized" AD model are a small fraction of the total traffic, and the overall signal-to-noise ratio is improved compared to the original unsanitized AD model. As a result, the ability to detect attacks, both in real time and in post-processing, is significantly improved.

Acknowledgements We would like to thank the anonymous reviewers for their comments and constructive criticisms of this work. This research was sponsored by the Air Force Research Laboratory under agreement number FA8750-06-2-0221, and by the National Science Foundation under grants CNS-06-27473 and CNS-04-26623. We authorize the U.S. Government to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or the U.S. Government.

References

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis. Detecting Targeted Attacks Using Shadow Honeypots. In *Proceedings of the* 14^{th} *USENIX Security Symposium*, pages 129–144, August 2005.

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

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

[4] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic Blending Attacks. In *Proceedings of the* 15^{th} *USENIX Security Symposium*, pages 241–256, July/August 2006.

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

[6] O. Kreidl and T. Frazier. Feedback control applied to survivability: a host-based autonomic defense system. In *IEEE Transactions on Reliability*, volume 53, pages 148–166, 2004.

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

[8] M. Ramadas, S. Ostermann, and B. Tjaden. Detecting Anomalous Network Traffic with Self-Organizing Maps. In *Proceedings of the* 6^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2003.

[9] S. Sidiroglou, M. E. Locasto, S. W. Boyd, and A. D. Keromytis. Building a Reactive Immune System for Software Services. In *Proceedings of the USENIX Technical Conference*, pages 149–161, June 2005.

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

[11] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Proceedings of the* 8^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2005.

[12] K. Wang, J. J. Parekh, and S. J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *Proceedings of the* 9^{th} *Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.