# Polygraph: System for Dynamic Reduction of False Alerts in Large-Scale IT Service Delivery Environments*

Sangkyum Kim
*UIUC*
*kim71@illinois.edu*

Winnie Cheng, Shang Guo,
Laura Luan, Daniela Rosu
*IBM Research*
{*wcheng,sguo,luan,drosu*}*@us.ibm.com*

Abhijit Bose
*Google*
*abose@google.com*

## Abstract

In order to avoid critical SLA violations, service providers use monitoring technology to automate the identification of relevant events in the performance of managed components and forward them as incident tickets to be resolved by system administrators (SAs) before a critical failure occurs. For optimal cost and performance, monitoring policies must be finely tuned to the behavior of the managed components, such that SAs are not engaged for investigation of false alerts. Existing approaches to tuning monitoring policy rely heavily on high skilled SA work, with high costs and long completion times. Polygraph is a novel architecture for automated tuning of monitoring policies towards reducing false alerts. Polygragh integrates multiple types of service management data into an active-learning approach to automated generation of new monitoring policies. SAs can only be involved in the verification of policies with low projected scores. Experiments with a trace of 60K monitoring events from a large IT service delivery infrastructure compare methods for threshold adjustment in alert policy predicates with respect to potential for false alert reduction. Select methods reduce false alerts by up to 50% compared to baseline methods.

## 1 Introduction

Proactive prevention and timely response to failures with minimal operational costs is a major target for service providers in large-scale IT infrastructures. In order to achieve this target, service providers use monitoring infrastructures such as IBM Tivoli 7 [5] and HP ServiceCenter 7 [4], to monitor the performance of the managed components and identify critical events. Such events are forwarded to incident management systems, and actions are taken before Service Level Agreement (SLA) violations occur. The monitoring agents deployed on managed components use pre-defined policies to generate events based on Key Performance Indicators (KPIs) and component execution contexts. Other nodes in the monitoring infrastructure perform event aggregation and incident ticket generation based on policies that aggregate in time and space (i.e., across multiple systems). Eventually, SAs analyze the auto-generated incident tickets, called *alerts*, to prevent or solve failures.

The effectiveness of the monitoring policies in capturing critical events with limited false positives and negatives has impact on the service management costs, including the cost of SA time spent with incident management, and the cost of SLA penalties. Previous work [2] and our observation of large-scale delivery infrastructures confirm the difficulty of configuring the monitoring policies for an effective cost balance because of the complexity of the monitored systems and applications.

This paper addresses the problem of effective configuration of the monitoring policies with an automated approach to dynamically modeling system behavior, generating monitoring policies and deploying them. The novelty draws from the integration of diverse service management content (e.g., incident tickets, system vitals) and operational domains (e.g., customers, clusters) and from the use of machine learning to model system behavior and to assess policy effectiveness. As a result, our system, called *Polygraph*, can automatically reduce the number of false-positive alerts, called *false alerts*, with limited or no impact on the identification of *true alerts*. Thus, Polygraph reduces SA work with both alert handling and policy tuning.

Polygraph builds on two novel principles. First principle is to **learn from SAs**, namely, learn from the resolutions of historical incident tickets handled by SAs about alert instances that can be safely ignored. This principle enables effective assessment of policies without elaborate, resource consuming system analysis. The second

*Partially supported by US National Foundation grant IIS-0905215 and the Blue Waters sustained-petascale computing project under NSF grant OCI 07-25070 and the state of Illinois.

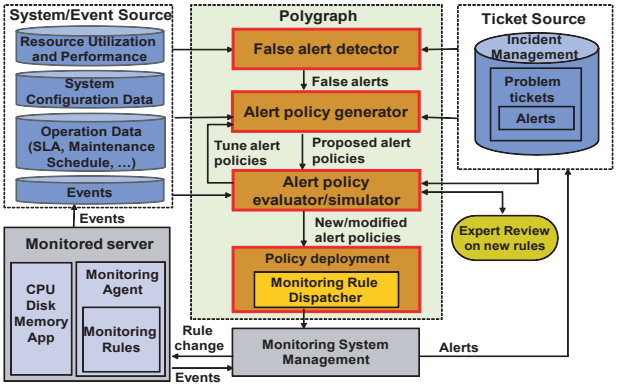


Figure 1: Polygraph system architecture and environment.

principle is to **leverage component similarity in large scale environments** in order to expand the input size for Polygraph learning tasks, and thus improve accuracy. Similarity is also used for policy deployment such that false alerts are reduced even for servers that have not experienced similar events yet, but are likely to experience them in the future. Towards this end, Polygraph uses temporal correlation of system vitals, configuration details, and change operation details.

Figure 1 illustrates the integration of Polygraph within a typical service management infrastructure and its main components. Polygraph has a close interaction with the monitoring infrastructure and leverages data from configuration management databases, repositories of historical system vitals, incident and change management systems, repositories of SLA and maintenance data.

The Polygraph prototype described and evaluated in this paper is focused on the analysis of alerts and generation of new policy. The implementation uses the IBM Tivoli policy specification language. However, our proposal does not depend on a specific monitoring technology, and can be extended to any IT service delivery environment. The evaluation compares several approaches for generation of new monitoring policies., and it is based on over 60K monitoring events, and related incident tickets and system vitals from a large IT service provider.

Overall, this work makes several contributions: (1) Design system architecture for continual refinement and assessment of policies based on integrated exploitation of diverse service management data, (2) Propose techniques for identification of false alerts by mining historical incident resolutions, (3) Propose techniques for generation and assessment of new monitoring policies that can significantly reduce the volume of false alerts.

In this paper, *true alert* identifies an alert instance for which SA intervention is required to solve a critical situation, and *false alert* identifies an alert instance that is cleared without any fix performed by SA, yet involves the SA for system status checks.

Next section discusses related work, Section 3 describes Polygraph architecture and implementation details, Section 4 presents the evaluation of Polygraph prototype. Finally, Section 5 summarizes our results. Refer to [7] for more extensive evaluation results and related work.

## 2 Related Work

There are several well-known commercial and community-supported platforms for monitoring datacenter and IT infrastructure components, including IBM Tivoli Monitoring [6], and HP OpenView [4]. Most of these products support alert-suppression based on statically specified policies. Our proposal uses policies dynamically generated by mining alert streams and other service management data.

Previous work in the area of incident prevention and resolution in large scale IT infrastructures used incident classification techniques based on performance metrics [1]. Polygraph uses a similar approach of learning from historical service management data in order to discover alert patterns and generate new policies.

In the area of dynamic generation of alert policy, the closest to our work is the approach in [2] for automated and adaptive threshold setting based on application Service Level Objectives (SLOs). This approach is hard to apply in large IT infrastructure due to the complexity of the method and the difficulty to acquire component dependency graphs.

A large body of work relates to false-alert reduction in intrusion detection systems (IDS). The use of data mining methods applied to historical data [9], and multi-stage analysis architectures render our work similar to some IDS solutions [8]. However, the binary IDS alert model (real vs. false attack) is different from the model of performance monitoring alerts, which includes quantified resource usage levels. Thus, this work cannot leverage the IDS methods.

Previous work [3] underlines the difficulty in classifying rare events because traditional learning classifiers are often biased for the most common events. The authors argue that if the events are rare and not too costly, the learning algorithms can do little to improve. When events have high costs (e.g., SLA penalties), a larger number of false alarms must be tolerated. This matches a best practice in IT Service Delivery for management of high-risk SLOs.

## 3 Polygraph Framework

For a competitive IT service delivery infrastructure, the monitoring infrastructure must minimize the number of false alerts in order to keep operational costs low and

must maximize the number of true alerts in order to prevent SLA failures. This goal can be achieved with detailed analysis and fine-tuning of alert policies. However, even with state-of-the-art tooling, this approach requires substantial SA effort and skills, prohibiting large-scale adoption.

Polygraph, the framework proposed in this paper, specifically addresses these limitations through the automation of monitoring policy evaluation and fine-tuning. The goal of Polygraph is to identify false alerts and design new monitoring policies that lower the occurrences of false alerts with negligible impact of true alerts.

The Polygraph method for false-alert reduction comprises: (1) *learning* the pattern of true and false alerts, (2) *generation* of new policy based on the alert patterns, (3) *assessment* of new policy impact. Figure 1 illustrates the component architecture that implements this method by integration with the overall service delivery infrastructure. Polygraph comprises of four functional components (see Figure 1): **False Alert Detector** performs the analysis of current alert specification effectiveness and false-alert detection. The module distinguishes false and true alerts by learning from SA's assessment of past incident resolutions. Alerts are associated with details about related policy and KPI thresholds, which are used in next stages.

**Monitoring Policy Generator** performs the generation of monitoring policies based on observed false-alert patterns. Data from similar servers is integrated in order to improve result quality.

**Monitoring Policy Evaluator** performs the evaluation of newly generated monitoring policies with focus on SLA impact (minimize missed true alerts) and work reduction (maximize eliminated false alerts). Evaluation is based on simulation against historical system vitals and monitoring events. Policies with acceptable balance of SLA impact and reduction move to deployment while others can be passed to SAs for further evaluation.

**Monitoring Policy Deployment** performs the deployment of new monitoring policies by close interaction with the monitoring infrastructure. Urgency of deployment is assessed base on server profiles, and used for scheduling policy deployment.

The reminder of this section is focused on select elements of Polygraph implementation.

## 3.1 Policy Model

In general, an alert policy is defined by a user-specified predicate over component KPIs or context parameters. The policy may also include threshold conditions on the event reoccurrence in order to delay alert generation. Polygraph prototype works with three types of policy: [BASIC] IF A; [AND] IF A AND B; and [OR] IF A OR B. Here, A and B are predicate units consisting of one KPI reference and related threshold value. Our methods apply to more complex predicate expressions by conversion to conjunctive normal form.

In the Polygraph prototype, we consider only threshold-based alert policies, which have predicates that refer only to KPIs and threshold values. The analyzed alert traces show that most false-alerts result from threshold-based policies. Sample policies for each type:

- IF ($System.VirtualMemoryUtilization > 90$)
- IF ($NT_Physical\_Disk.Disk\_Time > 80$) AND ($NT\_Physical\_Disk.Disk\_Time <= 90$)
- IF ($SMP\_CPU.CPU\_Status$ = 'off-line') OR ($SMP\_CPU.Avg\_CPU\_Busy\_15 > 95$)

## 3.2 Learning Alert Patterns

Alerts are identified by analysis of incident ticket details. The alerts-related tickets are automatically generated, and are characterized by a text pattern that may include details about the policy, monitoring event (KPIs and values). Polygraph includes methods for detection of text patterns for alert identification. For each alert, it extracts KPI values, alert policy references, and managed component references. Further, text analysis of incident resolution description helps classify an alert as true alert or false alert.

Given a scope defined by a target alert policy $P$ and a server or group of servers $H$, Polygraph scans the related historical content to identify alerts and constructs the *true set*, comprising the identified true alerts, and *false set*, comprising the false alerts. A sample approach to capture alert pattern is based on the KPI value patterns, i.e., the ranges of KPI values for each of the true and false sets. Polygraph includes also methods for discovery of time patterns, discussed next.

## 3.3 Policy Generation with Value Patterns

An alert value pattern is described by the *true value range*, the range of KPI values that corresponds to alerts in the true set. Without loss of generality, we assume a monotonic behavior for KPI values; if a value corresponds to a true alert, than all alerts for higher KPI values are true alerts.

**Proposition 1** *For a given alert policy $P$ consisting of one KPI parameter $p$ and its corresponding threshold $\theta$, let $T$ be the true set of $P$, and $t = min(T)$. If $t > \theta$, $P$ modified to include the threshold $t$ will not generate any false negatives for the given dataset.*

Proposition 1 describes how to tune the threshold value for a BASIC alert policy. If the smallest KPI value $t$ for true alerts is bigger than the original threshold, then

the new threshold value is set to $t$ and no true alerts are missed. False alerts related to KPI values between the old and new threshold are eliminated while those for KPI values above the new threshold remain. As an example, given a policy governing CPU threshold, if all true alerts happen for thresholds greater than or equal to 95%, we can safely raise the original threshold of 90% to 95%. If a false alert occurs for CPU load of 98%, it is not eliminated by the new threshold setting.

### 3.4 Policy Generation with Time Patterns

To further improve the false alert reduction, Polygraph takes into account the time patterns. Periodic patterns of jobs, like daily, weekly, or even monthly, can significantly affect resource consumption and trigger alerts, false or true. The method comprises the finding of periodic patterns based on learning set and extrapolation of these patterns in the analysis to remaining historical content. In order to limit the risk of missing true alerts by extrapolation, we focus on the periodicity of true alerts, rather than on false alerts.

Given a scope of policy and server(s), periodicity analysis starts with the related true set of the policy and produces a set of periodic time intervals, called *true time ranges*, during which all of the occurring alerts are considered true alerts. Alerts that occur outside the true time ranges are considered false alerts. Periodicity analysis requires specification of a threshold for the width of a true range to mine a set of these true ranges for each alert policy. For example, suppose host $H$ has three true events at 3pm, 4pm, and 8pm. Given a true time range threshold of 1 hour, the analysis results in two true time ranges (2pm-5pm) and (7pm-9pm). Smaller thresholds lead to more false alert detection, but increase the risk of missing true alerts.

### 3.5 Server-Based Policy Tuning

The typical approach to defining alert policy in service delivery infrastructures is to use the same best-practices policies for all servers with similar installed software and workloads. While this approach yields acceptable alert accuracy during early server lifetime, it will no longer efficient later in server lifetime, across capacity changes (e.g., memory or hard disk upgrades), dynamic re-clustering for workload balancing, and other events. For example, for a server with increase of hard disk capacity from 100GB to 10TB, an alert threshold of 90% utilization will generate alerts when 1TB of disk space is still available.

To address these limitations, Polygraph takes the approach of tuning alert policy thresholds for each server and our experiments show significant benefits. Polygraph does not exclude the tuning of alerts for groups of servers with similar configurations and workloads, which we plan to integrate in the future.

### 3.6 New Policy Evaluation

In the evaluation phase, Polygraph scans the set of alerts for the current scope (i.e., union of true and false sets) and compares the KPI values against the new policy threshold. The new counters for new true and false alerts are compared with the old counters in order to assess the risk (true alert misses) and benefit (false alert reduction). Installation-specific thresholds on the impact metrics are used to determine if the new policy (1) is highly beneficial with very low impact and should be automatically deployment,(2) has borderline benefits and risks and should be forwarded for SA analysis, or (3) has limited benefits and higher risks, and should be discarded.

## 4 Evaluation

Our empirical results are based on large and detailed datasets collected from globally distributed production environments serving real clients. We collected 30-day datasets with around 60K events. We divide the datasets of system performance metrics, alerts, and events into six parts (5 days for each) based on their occurred time: older data are to be used for learning purpose, and recent data are for test purpose. To show the effects of training data size on our alert threshold adjustment schemes, we use the first five datasets to make five differently sized training data (datasets of 5, 10, 15, 20, and 25 days) and the last part as test data. By default, the true time range threshold is 1 hour.

The experiments in this section compare various methods for new policy generation with respect the effect of the automatically generated policies to eliminate false alerts. Namely, the experiments present the ratio of false alerts for which the generated policies do not trigger alerts. We refer to the not-triggered false alerts as 'detected false alerts'. The larger the shares of detected false alert, the larger the reduction enabled by Polygraph framework.

### 4.1 Data Characteristics

In the target service delivery environment, alert policy specification comprises several fields including the target server and predicate descriptions. The threshold values have never been changed since initial deployed (which is typical in most environments).

System KPI values are collected at one-minute intervals and are aggregated over different time windows such as 15 minute, 1 hour, 1 day, and further.

We use two widely deployed alert policies, say $P_1$ and $P_2$, in our experiments to show the effectiveness of the Polygraph framework. Table 1 shows their characteristics, including total alert count, share of total alert trace, and distribution across true and false alerts; actual policy

Table 1: Characteristics of select alert policies

| Alert Policy | Count | Ratio(%) | True Alerts | False Alerts |
|---|---|---|---|---|
| $P_1$ | 23355 | 40.48 | 1026 (4.39%) | 22329 (95.61%) |
| $P_2$ | 3344 | 5.80 | 1526 (45.63%) | 1884 (56.34%) |

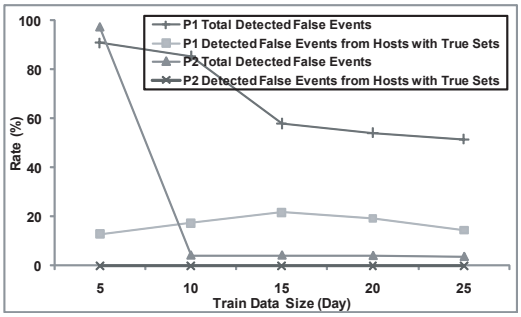


Figure 2: Host-based false alert detection with varying learning set size

expressions are not disclosed for privacy issues. $P_1$ has a large volume of alerts, of which 95% are false alerts. $P_1$ is a good example of how Polygraph can automatically and effectively tune an alert policy threshold. $P_2$ illustrates the case when Polygraph needs expert SA reviews to prevent abuse of automation.

## 4.2 Basic Threshold Adjustment

The basic threshold adjustment method comprises of policy generation using value patterns and including all servers, i.e., the entire data set is used for assessment of the true set. Both $P_1$ and $P_2$ alert policies do not have any gain when applying this method, even if the current policy thresholds are not optimal. This is because servers with same alert policy are not similar with respect to related datasets, as shown by experiments below.

## 4.3 Host-Based Adjustment

This experiment compares the basic threshold adjustment (i.e., analysis across all servers) with host-based adjustment (i.e., server-based analysis) including only the servers whose training data includes true alerts. Figure 2 illustrates false alert detection rates of $P_1$ and $P_2$ as the training set increases. Note that the difference between the plots for each policy indicates the false alert detection rate for servers whose training data do not have any true events. In general, we observe that more false alerts are detected as training set gets smaller. $P_1$ shows a high rate of false alert detection where as $P_2$ shows a low detection rate. For $P_2$, we see that at least 10 days of training data is needed for reliable performance. The spike of $P_2$ is due to the large number of servers with no true events in the training set.

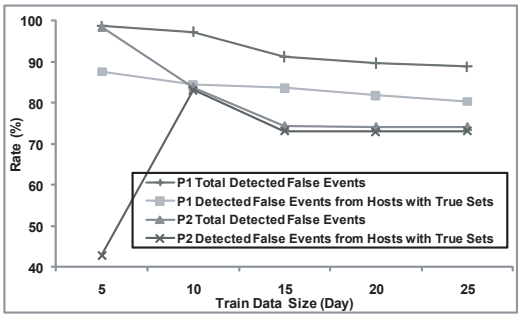


Figure 3: Host and time-based false alert detection with varying learning set size

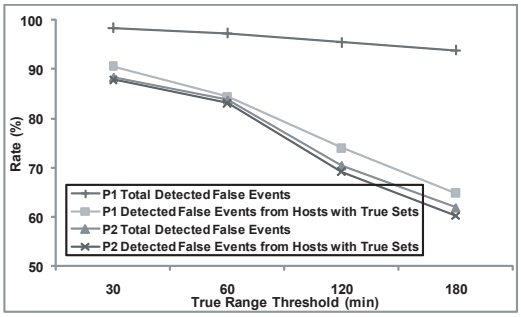


Figure 4: Host and time-based false alert detection with varying true time threshold

## 4.4 Host and Time-Based Adjustment

This experiment compares the basic adjustment method with a method using the time-based pattern for false alert detection and selection of the servers that experience true alerts. Figure 3 shows false alert detection rates of $P_1$ and $P_2$ for the two methods when increasing the training set. In general, the host-and-time-based schema shows higher false alert detection than the time-based schema (see Figure 2). Similarly, the host-and-time-based scheme shows higher false alert detection rates than the host-based scheme. Based on the experiments described above, $P_1$ can be safely tuned by Polygraph with no human interaction, but $P_2$ needs to be shown to the system administrator before deployment.

## 4.5 Impact of True Time Range Threshold

This experiment evaluates the impact of the *true time range threshold* on the rate of false-alert detection when applied to host-and-time-based scheme. The experiment uses the 10-day training dataset and varies the threshold value from 30 to 180 minutes.

The results are illustrated in Figure 4. As the threshold value increases and true time ranges of $P_1$ and $P_2$ get larger, the rate of false-alert detection decreases.

## 4.6 Discussion

Based on our experiments, we identify extensions that can improve the effectiveness of Polygraph in reducing

false-alerts by automated tuning of alert policies.

**Leverage operational data for tuning alert policy.** The analysis of monitoring event data demonstrates the need to leverage operational data in the tuning of alert policy. For instance, Polygraph integrates operational data that describes when the scheduled maintenance activities are expected to generate significant workload on the managed servers, such as anti-virus scans and backups. Our analysis shows that 20.3% of a customer's alerts are due to virus scan that caused higher CPU usage than the normal state. In order to eliminate these false alerts, one has to exploit operational data and include in the new policies predicates related to the execution context. Another relevant type of operational data includes SLA specifications and attainment statistics. One can exploit SLA information to delay generation of alerts when workload varies significantly but yet the SLAs are unlikely to be missed.

**Emphasize more recent history.** When long event history is available, such as in typical production environments, false-alert detection is likely to exhibit poor quality if all data samples receive equal weight in the analysis. This is because the detector misses to recognize the most recent trends in the occurrence of false alerts. In order to improve decision quality, a weighted scheme can be employed to emphasize recent input. Weights should be carefully chosen such that the discard of old content does not cause the missing of true alerts.

**Scalable policy deployment.** Polygraph can generate new policies for a server profile that matches a very large number of servers. In order to avoid the disruption of the monitoring infrastructure, the policy deployment should be staged. The staging order and timeline is based on the assessment of server-specific risks/costs due to delaying the policy deployment.

**Impact of change operations.** The behavior of a managed system changes over time due to infrastructure changes for the server itself (e.g., hardware, software) or the environment in which the server is running (e.g., workload). As a result, monitoring policy becomes outdated. Integration of service management data that describes change operations, such as new patch/software installation, memory expansion, and subnet change, should be used to trigger analysis for policy tuning and determine the relevant historical content to consider.

**Leverage server similarity.** Our experiments reveal the potential benefit of grouping similar servers in alert policy tuning. This is helpful in cases when the training dataset collected on an individual server does not have sufficient data points for rare events; grouping similar servers provides a better training dataset, hence better policy tuning. A sample similarity criterion includes the servers in the same web-application cluster, which all have the same server configuration and the same workload characteristics. For more general cases, Polygraph must use a server similarity measure that integrates server resource profiles and workload characteristics.

## 5 Conclusion

This paper introduces Polygraph, a framework for automated reduction of false alerts in large scale IT infrastructures based on an active-learning approach. Polygraph mines historical incident ticket content to learn from SAs about which alerts are false and to correlate this information with other service management content, such as system vitals and server similarities, to modify threshold-based alert policies and eliminate false alerts. In experiments with real-life traces from a large and diverse service delivery environment, Polygraph performs very well in reducing false alerts while not missing true alerts. Polygraph achieves this by combining host and time-based tuning of alert policies.

In future work, we plan to extend the Polygraph model for automated generation of new policy by including conditions on execution context and methods for automatic identification of durations when maintenance processes generate temporary spikes in resource consumption. Furthermore, for improved quality of false-alert reduction, we plan the development of methods that emphasize recent history.

## References


[1] BODIK, P., GOLDSZMIDT, M., FOX, A., WOODARD, D. B., AND ANDERSEN, H. Fingerprinting the datacenter: automated classification of performance crises. In *EuroSys* (2010).

[2] D. BREIGAND, E. H., AND SHEHORY, O. Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management. In *ICAC* (2005).

[3] DRUMMOND, C., AND HOLTE, R. Learning to Live with False Alarms. In *Proceedings of the KDD Data Mining Methods for Anomaly Detection Workshop* (2005).

[4] HEWLETT-PACKARD COMPANY. HP ServiceCenter software. `http://www.openview.hp.com/products/ovsc/index.html` (2011).

[5] IBM CORPORATION. Tivoli Monitoring. `http://www-01.ibm.com/software/tivoli/products/monitor/` (2010).

[6] IBM CORPORATION. Tivoli Software. `http://www-01.ibm.com/software/tivoli/` (2010).

[7] KIM, S., CHENG, W., GUO, S., LUAN, L., ROSU, D., AND BOSE, A. Polygraph: System for dynamic reduction of false alerts in large-scale it service delivery environments. *IBM Research Technical Report* (Apr. 2011).

[8] N. JAN, S. LIN, S. T., AND LIN, N. A decision support system for constructing an alert classification model . *Expert Systems with Applications Vol. 36, No. 8* (Oct. 2009).

[9] SOLEIMANI, M., AND GHORBANI, A. Critical Episode Mining in Intrusion Detection Alerts. In *CNSR* (2008).