A Cloud-Based Anomaly Detection for IoT Big Data

Omri Soceanu$^{1(\text{E})}$, Allon Adir$^{1(\text{E})}$, Ehud Aharoni$^{1(\text{E})}$, Lev Greenberg$^{1(\text{E})}$, and Habtamu Abie$^{2(\text{E})}$

$^1$ IBM Research - Haifa, Haifa University Campus, 3490002 Mount Carmel, Haifa, Israel
{Omri.Soceanu,adir,aehud,levg}@il.ibm.com

$^2$ Norwegian Computing Center, adir, P.O. Box 114, 0314 Blindern, Oslo, Norway
habtamu.abie@nr.no

Abstract. Security of IoT systems is a growing concern with rising risks and damages due to successful attacks. Breaches are inevitable, attacks have become more sophisticated, and securing critical infrastructure has become a greater challenge. Anomaly detection is an established approach for detecting security attacks, without relying on predefined rules or signatures of potential attacks. However, existing outlier detection techniques require adaptation if they are to be applied in a Big Data cloud context. We describe a novel outlier detection solution, which is currently being used by hundreds of customers with highly variable data scales. We describe our work in adapting this technology to handle IoT on a Big Data cloud setting. Specifically, we focus on efficient outlier analysis and managing large numbers of alerts using automatically controlled alert budgets.

Keywords: Cyber-security · Anomaly detection · Outliers · IoT big data · Cloud · Critical infrastructure

1 Introduction

IoT security is of the utmost importance. The impact of a security breach can go far beyond the attacked device. Attacks can range from a privacy breach and the exposure of sensitive and personal information, to a life threatening attack on an automotive system, or a distributed bot-nets attack with major financial ramifications. A breach can also lead to diminished customer trust or even to customer turnover.

Attacks have also become wider and more sophisticated. A survey of IoT-enabled cyber attacks [15] assessed the increasing attack paths to critical infrastructures and services for all application domains. DDoS attacks are on the rise, getting more complex and increasing in power with the average attack in 2020 using more than 1 Gbps of data with most attacks now lasting 30 min to an hour [8].
Breaches are inevitable and securing critical infrastructure has become a greater challenge. Moreover, many attacks cannot be stopped by standard policy-based or rule-based security systems. In cases of accounts stolen from privileged users or internal attacks, firewalls, access control levels, and rule-based management are not sufficient. There is a real need for a different approach that detects malicious activities beyond the abilities of rule-based systems.

Anomaly detection is one such established approach, and it has already been used in a wide range of security settings. This approach unites a rich family of techniques [10] and doesn’t rely on predefined rules or signatures of potential attacks. Instead it learns the typical behavior of the monitored system and triggers an alert when abnormal behavior is detected.

1.1 Challenges and Our Contribution

Compared to rule-based and signature-based approaches, anomaly detection is a more complex process that includes advanced analytics for training and analysis. This in turn imposes higher demands on disk space, memory, and CPU resources. Even in organizations that recognize the importance of data security, typically only a small portion of the overall computational resources can be allocated for security analytics.

The growing data volumes of industrial use cases make the performance and scalability requirements for this kind of analytics increasingly difficult to meet. This has become a critical factor in the acceptance of anomaly detection solutions. The need to adapt anomaly detection techniques to large scalable systems is especially vital for IoT platforms. Here, large numbers of highly diverse devices, from different domains and behavioral aspects, need to be analyzed to detect anomalies. IoT platforms often include many devices with high communication frequencies. The size of the IoT platform, and thus the amount of data to be analyzed, can also grow or change over time. This raises the requirements of the anomaly detection platform to automatically scale, while preserving performance and resilience, in the face of a dynamically growing IoT system.

Another challenge in anomaly detection is the number of false positives, i.e., cases when an alert is triggered, but no attack is actually taking place. Even a false positive rate of one per mille (0.1%) would make an anomaly detection system impractical in a large scale setup. It would produce too many alerts to be handled by the limited resources of security operators. Such a high rate of alerts would cause the security operators to become less sensitive and might result in their missing an alert triggered for a real attack.

In this work, we present a novel scalable platform for anomaly detection. Our main contributions are:

- An industrially proven generic and extendable anomaly detection architecture that can be easily customized for different domains.
- A number of techniques that enable the significant reduction of computational resources and disk space requirements, geared for scalable anomaly detection.
- A novel approach to control a number of generated alerts within a predefined budget, while preserving the most significant outliers by learning historical outlier distribution.
- A scalable cloud-based architecture that can analyze the Big Data produced by large IoT systems.

1.2 Related Work

The main approaches of anomaly detection use statistical methods, machine learning, and data mining techniques. A wide range of anomaly detection techniques is reviewed in [10]. One of the most well-studied applications of anomaly detection in the security domain is network intrusion detection. Bhuyan et al. [6] provides a comprehensive survey on anomaly-based network intrusion detection techniques. However their usage for data protection is limited. Deorankar et al. [9] survey anomaly detection techniques based on Machine Learning for IOT cyber-security. A number of intrusion detection techniques, datasets and challenges are surveyed in [11]. IoT anomaly detection has been suggested at the device, router, and cloud levels. Butun et al. [7] survey different anomaly methods and their applicability to IoT and cloud applications. Yu et al. [17] proposed unsupervised contextual anomaly detection method in IoT through wireless sensor networks. Yan et al. [16] use a classification algorithm based on mini-batch gradient descent with an adaptive learning rate and momentum to detect network anomalies in IIoT. Liu et al. [13] and Arrington et al. [4] use artificial immunity to detect anomalous behavior of IoT devices at the router level. Mehnaz and Bertino [14] designed a privacy-preserving real-time anomaly detection using edge computing that detects point, contextual, and collective anomalies in streaming large scale sensor data while preserving the privacy of the data. A scalable, cloud-based model to provide a privacy preserving anomaly detection service for quality assured decision-making in smart cities described in [2].

Scalability is a well known challenge for anomaly detection. Koufakou [12] proposes the Attribute Value Frequency (AVF) algorithm as an efficient approach for anomaly detection of categorical data. AVF is compared to a number of Frequent Itemset Mining algorithms based on the Apriori algorithm [1] and shows much better run-time without losing accuracy. Another efficient technique is described by Bertino et al. [5]. Here the Naive Bayes Classifier method is used to obtain linear scalability for the training phase and very fast analysis. Moreover, Bertino et al. compare the use of three different levels of information granularity: coarse, medium-grained, and fine. They show that on real data the achieved precision and accuracy of outlier detection is very similar for all the levels, but the execution times are a few orders of magnitude faster for coarse information granularity. We chose a multi-level approach to improve scalability. At the algorithmic level, we pervasively use exponential smoothing. At the architecture level, deploying the system in IBM’s cloud environment Apache Spark™ for a quick and easy storage and performance scale-up.
2 Cloud-Based Architecture

The architecture we use to deploy the outlier detection system on the cloud is depicted in Fig. 1. The main input to the system is an Apache Kafka™ stream of event descriptions, where each event is described in terms of specific features. For example, Table 1 shows part of a stream of events, each corresponding to some communication between a particular device and the IoT management system being used. The features include some meta data describing the communication, such as the time, device owner (Client), device ID, action type (e.g., send, connect, or disconnect), and whether the action failed or not. In the case of a send action - the sent payload can also be analyzed. Thus, the system can detect anomalies related to meta-data of the message, such as the software used to send the message (SW), and the message size and format (which can be automatically computed). We can also analyze semantic features of the message, such as temperature readings ($T$).

| Time | Client | Device | Action | Fail | Payload |
|------|--------|--------|--------|------|---------|
| 07:06 | C1     | Cam1   | Send   | 0    | { $T=20$, SW=S1...} |
| 07:07 | C2     | Cam2   | Send   | 0    | { $T=19$, SW=S2...} |
| 07:07 | C2     | Cam5   | Connect| 1    |         |
| 07:10 | C1     | Cam1   | Send   | 0    | { $T=56$, SW=S1...} |
| 07:13 | C1     | Cam4   | Connect| 1    |         |

Our outlier detection system defines a specific interface for the event stream. The event stream arrives on a dedicated Kafka stream and in a particular format, which includes an identification of the device involved in the event and the set of event features in JSON form. The outlier detection system is thus general purpose in the sense that it can analyze any event stream that conforms with this interface. The IoT devices communicate using messages of different forms and purposes arriving on multiple MQTT topics. Messages are then forwarded to a central Message Hub cloud service from which they arrive at the input bridge shown in Fig. 1. The input bridge converts all these variously formed messages into the specific form defined by the interface. Beyond simple formatting, the input bridge can also compute features from the incoming messages. For example, it might compute the payload size, determine the message format (as features to analyze), or extract other feature from the payload data.

The outlier analysis takes place on a cloud environment, the same cloud environment used. The outlier detection runs on a Spark service. Using Spark makes the solution very scalable because the resources needed for the analysis (memory, disk, CPU, communication) can be adjusted if the analyzed IoT system...
grows (or shrinks) without any manual changes required in the code or in other parts of the system.

The outlier detection includes two main units running in a pipeline: aggregation and scoring (see Fig. 1). The aggregation unit takes the input event stream described above and converts it into a stream of features aggregated for pre-defined time windows (termed analysis periods). For example, it could compute the total payload size sent by a device during the last 15 min. When the analysis period ends, the aggregator streams all the aggregations to the scoring unit. In the example just given, the aggregated input feature is the payload size, the aggregation type is summation, and the analysis period is 15 min. This specific setting is defined in a configuration kept on a Cloud storage service (Object Storage shown in Fig. 1). The configuration typically defines several aggregations; some are general purpose (e.g., the number of failures, or communication frequency) and some are specific to the IoT domain.

The scoring unit receives the stream of aggregations and uses these to update behavioral models of the analyzed entities. The entities being modeled are defined for every specific domain; they typically include all the devices, device types, and clients in the system. We model the devices so that we can detect anomalous device behavior. We model the device types so that we can detect an attack on an entire class of devices, for example via malware that attacks some particular firmware. Modeling clients can help detect a broad attack where many devices exhibit minor changes in behavior that only appear abnormal when examined together in aggregate.

The scoring unit can now compare the incoming aggregations with the behavioral models of the entities involved. If the difference is large enough, an alert is sent on the output alerts Kafka stream. The alert is sent along with enough information to justify and explain the reason for the alert. This includes the observed anomalous feature along with the corresponding statistical information from the model (e.g., the historic average and standard deviation). An alerts-consumer now reads this stream of alerts and can store or display these in a dedicated GUI.

The data continuously streams along the system units, from the devices sending their events to Watson IoT, to the output alerts stream. Thus, data is almost never at rest, except for the short terms during the windowed aggregation and for the summaries kept in the models for longer periods. This means we do not have to keep the very large quantities of data that can be expected in IoT system logs. That said, the system only knows the summary information in the model and does not have access to the entire precise historical data. If such historical data is needed (e.g. for detailed post-mortem attack analysis), then the history should be backed up via a separate facility.
2.1 Anomaly Detection Unit

As described above, the system models the typical behavior of certain modeled entities such as devices, device-types, clients, end-users, and geographic locations. This allows the system to later detect deviations from the normal modeled behavior. We model these behaviors using various features that are aggregated during analysis periods of fixed length. For example, the model may note the average and variance of the number of failures the entity has per analysis period.

The events streaming into the system are described in terms of features, some of which are used to identify the modeled entity involved in the event (e.g., the Device and Client features in Table 1). Other features are aggregated over time periods for analysis purposes. These are termed aggregated features (e.g., the payload size, and failure counts). An event meta-type is a tuple of event features names. Consider the example features SW, Action of the event stream of Table 1, and the computed feature of the message format - Format. The defined event meta-types may include, for example, <SW, Format>, <SW, Action>, and <Action>. An event type is an instantiation of an event meta-type that assigns specific values to the features named in the event meta-type. For example, <SW = S1, Format = JSON> is an event-type of the event meta-type <SW, Format>. Every specific event in the event stream may correspond to a number of event types, each relating to different event features. For example, the first record in Table 1 corresponds to (at least) the event types <SW = SW1, Format = JSON>, <SW = SW1, Action = Send>, and
The system uses a fixed collection of event meta-types for every modeled entity type. So, for example, the system may define the event meta-types \<$SW, Format>$, \<$SW, Action>$, and \<$Action>$ for device entities, and the event meta-types \<$device>$, \<$SW, Action>$ for client entities. For every event, the system creates a list of modeled entities and corresponding event types that match the event. For example, for the first record in Table 1, the system identifies the modeled entity \<$Device = Cam1>$ and corresponding event types \<$SW = SW1, Format = JSON>$, \<$SW = SW1, Action = Send>$, and \<$Action = Send>$.

Note that the feature \textit{Device} sometimes serves to identify a modeled entity and sometimes serves as part of an event type. Every analysis period (e.g., 15 min), the aggregation unit (see Fig. 1) creates an \textbf{event type summary} for every modeled entity. The summary specifies the number of events of each event-type that occurred with the entity. The summary also includes aggregations of features for every event type used by the entity - such as the total number of failures, or maximum payload size. In our example, the system would calculate that during the period 7:00–7:15, the modeled entity \<$Client = C1>$ had 2 events of the event type \<$Device = Cam4>$, aggregated with 2 fails. It also had 2 events of the event type \<$SW = SW1, Action = Send>$ aggregated with no fails. Thus, the aggregation unit aggregates the aggregated features for every event type per modeled entity in the event type summary. It can also compute the total aggregation of the aggregated features for every entity, regardless of the event types involved. Thus, we can configure the aggregator to compute the total number of fails, and average or maximum payload size for every entity.

The aggregation unit runs on a Spark service as a collection of distributed processes that perform the aggregations jointly in a map-reduce approach. Even in very large IoT systems, where Big Data scales of traffic information needs to be analyzed, the system can still perform the aggregation with reasonable performance and can be simply scaled by adding more resources to the Spark cluster. The system was designed to keep a minimal amount of data during its operation. Once any piece of the data is streamed into the system and analyzed, it can be dropped. The temporary event summaries need only be stored, in the form of Spark RDDs (Resilient Distributed Dataset), for the duration of the analysis periods. Even the entity models that need to be kept for longer time spans while the entity remains active, keep relatively small amounts of summary data per entity. This is possible because the averages in the models are all smoothing averages; these can be updated incrementally after every period of activity. See Sect. 3.

### 2.2 Anomaly Scoring

The anomaly scoring is carried out concurrently by multiple Spark processes. Every \textbf{analysis process} is responsible for analyzing the activities of several modeled entities during the last analysis period. An analysis process considers all the event types and corresponding aggregated features from the event type
summary of the analyzed modeled entity. It compares the behavior observed in the last analysis period to the entity’s normal behavior as modeled. It then updates the behavioral model of the entity. This is carried out by a collection of anomaly scorers, each scorer responsible for analyzing, modeling, and checking a particular behavioral aspect and producing a score. The scores represent the risk or probability of an attack and are therefore given as numeric values between 0 and 1. A higher value corresponds to a higher probability of an attack.

For example, the volume scorer looks for an unusually high number of events of some event type during the analyzed period. For this purpose, for every modeled entity, the scorer maintains some volume related statistics for every event type (e.g., $<SW = SW1, Action = Connect>$). This includes the average volume observed so far, the average of the volume squares (to compute the variance), the top volume scores with representatives from recent weeks, and the number of samples that were used for computing the statistics i.e., the number of events the modeled entity had of this event type in the past. All the averages and counts here are computed using smoothing average techniques (see Sect. 3) and are computed in a logarithmic scale. Using logarithmic scales is a conventional approach when analyzing volumes, since their order of magnitude is typically more interesting than the precise volume.

These statistics are, in effect, the volume-related part of the model for each entity. The volume scorer is responsible for updating this part of the model by referring to the aggregated volume in the event type summary of the entity. But before doing so, the scorer would give a volume anomaly score to the event type by comparing the count in the event type summary to the related statistics in the model. For example, it could compute a risk based on the Z-score from the newly observed volume and from the modeled average and variance; it would then “moderate” the score by considering the number of samples and the maximal values observed so far. A combined volume score is then computed for the modeled entity by taking the top scoring event types and computing some weighted average of their scores.

The volume-related part of the model includes statistics for individual event types alongside general statistics of the modeled entities. Most other scorers require more compact models that only include general statistics of the modeled entities. For example, a new activity types scorer models the typical number of new behaviors that a modeled entity (client or device) exhibits every period. A variability scorer tracks the general variability of a device’s activity. If a device normally exhibits small variability (i.e., repeats having more or less the same diversity of activities) and suddenly behaves with much greater variability, this could indicate an account hijacking.

Alert generation is the final step in the analysis process for a period. Here, the multiple scores coming from the various scorers are weighted and combined for every modeled entity. If the resulting score is above some threshold (automatically set by the budgeting scheme - see Sect. 4), an alert is produced and displayed in the system GUI. The GUI also gives the reasons for the alert in a user friendly display of the related statistics, and allows the end user to further
investigate the root causes for the alert (e.g., by displaying all the related device activities).

2.3 Generic Architecture

The system is generic in the sense that it can easily be configured to find anomalies in completely different domains, in logs of different types of systems, and for different purposes. As mentioned above, beyond the IoT use case, the tool is currently part of an IBM product where it reports possible insider attacks by analyzing the logs of user database activity. The tool was also configured to serve other use cases, including detecting anomalies in logs of storage devices to predict pending device failures, and to detect anomalies in the logs of a cloud-based object store.

Configuring the tool for a specific domain includes the following steps:

**Model Design:** The model consists of entities and corresponding event meta-types. These are defined in terms of event features that are available or that can be computed. This step is carried out by experts in the domain and depends on the particular use case for the anomaly detection. In our example above, the experts decided to model the behavior of devices, device types, and clients. When selecting meta-types, the choice of the meta-type $<SW, Action>$ was made so that abnormal volumes of different types of actions by some software can be detected. The choice of the meta-type $<Action>$ was made so that abnormal usage volumes for specific action can be detected, regardless of the software involved.

**Preparing the Event Stream:** The IoT system should forward the events to be analyzed to the dedicated Kafka topic used by the input-bridge. The input-bridge must be set up to collect or compute the chosen event features from the input event stream before forwarding them to the outlier detection units (see Fig. 1).

**Domain Specific Scorers:** During system configuration, one can select relevant scorers from a library of available generic scorers. The overall volume and failure scorers described above are generic in the sense that they are relevant to many IoT domains. However, one can also add new scorers that are relevant only to the specific domain being configured. In a camera IoT for example, a scorer can be added to detect anomalies in the number of automatic movement operations that the camera reports in its message payloads.

3 Exponential Smoothing Averages

The behavioral models we use for anomaly detection rely on computing the average and standard deviation of various properties, e.g., the number of operations of a particular type that a particular device performs per hour. These statistics can then be used to detect anomalies. If in the current hour, the number of operations of this type is far from the average by a large enough number of standard deviations, an alert is issued.
Say \( V_1, V_2, V_3, \ldots \) is a stream of observed values, where \( V_i \) is observed at the \( i \)’th period. One way to compute the average at time \( i \) is simply as the average of values observed until this point in time, \( (V_1 + V_2 + \ldots + V_i)/i \). This has the advantage of being easy to update incrementally, without the need to store the values observed prior to period \( i \). However, it has the disadvantage that it gives equal weight to all observed values, old and new. In reality, a device’s behavior is gradually changing, so older values should have a decreased impact on the current estimate of the average.

Another approach is to store a sliding window of the last \( k \) observed values, \( V_{i-k+1}, V_{i-k+1}, \ldots, V_i \), and compute the average over this window. This has several disadvantages. One disadvantage relates to the way old values are treated. Older values within the window are weighted the same as the newer values in the window. When a value drops off the window, it abruptly gets completely forgotten and no longer has any effect on the current estimate of the average. Another disadvantage is that performance issues may arise. Assuming we are tracking a large number of properties and use large windows, storing this data may require a large space, and maintaining it as new data arrives may be slow.

Since our system must work on Big Data, efficiency considerations are crucial. Therefore, we chose a third approach: exponential smoothing averages, referred to as smoothing averages below. Denoting the smoothing average at time \( i \) by \( S_i \), the basic formula is as follows.

\[
S_i = V_i \alpha + S_{i-1}(1 - \alpha),
\]

where \( \alpha \) is some predefined constant. A possible method for choosing \( \alpha \) is to select the window size that will have a total influence weight of 0.5 on the smoothing average result. For a window of \( n \) periods, this is \( \alpha = 1 - 0.5^{1/n} \).

A model composed of a set of smoothing averages has several advantages: much smaller model size, efficient model updates and accounting for both recent and historical data.

### 3.1 Smoothing Average Behavior

We demonstrate the behavior of the smoothing average scheme over specific test cases. We created the test cases by generating independent and identically distributed samples of normal distributions with a standard deviation of 1 and with means that were chosen to simulate various changes in behavior, as detailed below. One additional test case was taken from real data gathered on a running instance of our system at a customer site. In all test cases, we show the smoothing average approach with initialization and gap handling. The \( \alpha \) is configured such that the weight of 25 time periods will be equal to 0.5 (“smoothing average” in the figures). We compare this to a simple sliding window scheme of 50 periods (“sliding window” in the figures).
Figure 2 shows the reaction to an abrupt change in the input. Both approaches are similar, except the smoothing average takes a bit longer to fully adapt to the change. In Fig. 3a, we show the same thing, except there’s also a gap of inactivity between the old (high) values and the new (low) ones. Since it gives lower weights to older values, here the smoothing average adapts to the change more quickly.

Figure 3b shows real data giving the number of different unique operations per hour performed by a user. During most of the sampled period there are no gaps and only moderate changes. As a result, the two measures behave similarly, but employing the smoothing average is more efficient computationally. Some differences do exist and can be explained. During hours 130–150 the sliding window average increases with no apparent justification. This happens because the very low measures of hours 80–90 are being removed from the window. The smoothing average more intuitively tracks the data and actually decreases during hours 130–150, by giving higher weights to the new measures.

4 Alert Budgeting

Outlier detection systems are characterized by the dominance of legitimate behaviors in the training data. Malicious examples are typically missing from the training data or are too few to be representative. The commonly used approach in such a setup is to train models that describe the typical behavior of entities, including typical deviations from the average behavior. The trained models are then used to score the entities’ current behaviors according to their deviations from the typical historical behavior.

In an ideal world, the malicious behaviors’ deviations would be much larger than the deviations of legitimate behaviors. Thus, one could just look for large
enough deviations from the historical data and expect to detect all malicious events, without producing any false positives for legitimate behaviors. In practice, it is not exceptional that some legitimate behaviors exhibit a high level of abnormality. Meanwhile, sophisticated attacks try to mimic legitimate behaviors to escape detection by outlier detection systems. One of the important questions is how to choose the threshold above which a behavioral deviation should trigger an alert to the security operator. If the threshold is set too low, the rate of false alerts is too high and exceeds the capacity of the operators. On the other hand, if it is set too high, the chance of missing malicious behaviors significantly increases.

There are a number of existing approaches for the selection of a threshold. The most basic approach requires security operators to manually select the threshold. The problem with this approach is that it is not clear how to come up with a good threshold. A good threshold can differ in different domains and may also change over time. It can lead to too many alerts emitted at some periods, and to no alerts at all even when high risk events occur. Another approach allows you to define a limited budget for the number of alerts emitted per a predefined interval, such as 48 alerts per day. In this case, waiting 24 h and reporting 48 top alerts introduces a reporting delay that is impractical. On the other hand, reducing the delay by reporting alerts each hour (in this case 2 alerts per hour) is also not optimal. The top scoring events typically have low scores for most of the hours, but the system would still trigger 2 alerts per hour. This makes operators less sensitive to the alerts. In addition, hours with many events with high scores will be limited to 2 alerts, so some high risk events might be missed. A number of works [3] propose various optimization methods for thresholds selection (see Sect. 1.2).

In this work, we developed a novel approach that defines an alert budget \( B \) as the rate of alerts that operators decide they can handle. Given the alert budget, the system periodically re-adjusts the threshold automatically based on historical scores, so the average rate of emitted alerts matches the predefined alert budget.
The alert budget is designed to be satisfied on average and not for every period, so one can expect some fluctuations in the number of alerts per period. To constrain the variation of the number of triggered alerts, we introduce a \textbf{max alert number} parameter $K$. The max alert number is a “hard” constraint, so if for a specific period there are more than $K$ events with scores above the threshold, only the top $K$ events with the highest scores are reported. The max alert number should be set to a large enough value to allow a reasonable level of alert fluctuations, but not too large to avoid a situation where a few non-typical hours (e.g., system upgrade) produce too many alerts and, as a result, raise the threshold too high. A possible way to select $K$ is $K = \max(10, 10Bd)$, where $d$ is the duration of a single period. This restricts one period from producing more than 10 times more alerts than the average per period, or more than 10 alerts for the case that $B < 1$.

The alert budgeting approach has two important aspects for anomaly detection in a Big Data setup. First, it allows us to define and automatically control the average rate of alerts produced by the anomaly detection system. So even for very high rates of monitored events, only a predefined rate of alerts will be produced. The target alert rates should be selected according to the human resources available. Second, as we show below, our approach uses a fixed amount of disk space that doesn’t depend on the volume of monitored data and takes only a small portion of the processing time.

### 4.1 High-Level Budgeting Flow

The high-level flow for alert budgeting is depicted in Fig. 4. Given the alert budget $B$, the flow starts from scores that are periodically produced by the scorers. For each period and for each scorer, the top $K$ scores are inserted into the top scores tables, where $K$ is the max alert number parameter.

The top scores table keeps a sliding window of $t$ historical periods. The size $t$ of the sliding window determines how fast alert budgeting reacts to the changes in scores. For example, when a one-day sliding window is used, the system is significantly influenced by a local change of a few hours duration, which makes the system unstable. On the other hand, a one-year sliding window makes the system react very slowly to the changes, so it might take a few months for the system to start reacting to a change. For our industrial use case, we used a one-week sliding window; that gave us the right balance between the agility and stability of the system.

Based on the historical top scores, the \textbf{threshold calculation} step computes \textbf{scorers thresholds}. The score thresholds are computed in such a way that the number of historical events above the threshold matches the target alert budget. If the top scores of new periods have a similar distribution to the historical top scores, the average number of future events above the threshold will also match the target alert budget. However, if the score distribution changes, (e.g., higher scores appear more frequently) the historical scores gradually accumulate in the new distribution and the score thresholds will change accordingly.
The score thresholds are used in the **scores normalization** step to normalize different scores to the same scale. Only scores from previous periods are used to compute the current period thresholds; in this way the thresholds can be calculated before starting the analysis. This allows scores to be normalized in a distributed way, without having to wait for the processing of all the events of the current period.

The **final score** is computed based on the normalized scores and is used to determine whether an alert should be reported.

![Alert budgeting](image)

**Fig. 4.** Alert budgeting

### 4.2 Scalability

While the alert budget is not guaranteed to produce the same number of alerts for each period, on average it provides the required rate of the alerts. This allows the planned allocation of the resources needed to investigate the alerts and enables security operators to effectively monitor systems with high volumes of activities.

Using the alert budget scheme doesn’t require keeping a full record of historical events; only a very limited portion is required. The critical parameters here are the number of scorers ($N$), the length of the sliding window of historical periods ($t$), and the number of top scores held for each scorer $K$ that equals the max alert number parameter. The total number of records $R$ required to be stored is $tKN$. 
For example, for 5 scorers ($N = 5$), a sliding window of historical periods equal to one week ($t = 168$) and a max alert number $K = 10$, the resulting total number of records is $R = 8400$ and doesn’t depend on the total volume of activities.

5 Results

To evaluate budgeting quality we used real production data collected over 3 months (from an undisclosed European telecom company). Traffic from 7949 internal-system users was recorded and analyzed. We fixed the anomaly budget goal to $B = 2$ alerts per day, and the history training sliding window length to $t = 7$. Next, we compared the budgeting quality for a variety of budget-factor update frequencies. Thus, we are able to compare our proposed automatic, hourly threshold update frequency to a manual update at different frequencies. This test can be used to assess how frequently a human operator would have to change the budget factor in order to match the performance of an automatic system. The performance is measured as the difference between the actual alert rate and the target budget goal line. To this end we compare $B$ with the number of cumulative alerts up to each day, divided by the cumulative number of days.

Figure 5 plots for every date the number of alerts during a 7-day sliding window. It can be observed that for a 30-days update frequency, the 14 alerts per week goal ($B = 2$ alerts per day) is hardly met. Specifically, there is a strong under-budget trend from mid-September till mid-October. At the beginning of November, the 7-day and 30-day update lines are much higher than the goal line, while towards mid-October they are both much lower than the goal line. These drastic fluctuations result from the specific day when the factor is calculated. Since both the 30-day and 7-day factors are static for a very long time, calculating the budget factor at a peak or a low point of a monthly cycle has a lasting effect. In contrast the 1-hour update frequency is much more dynamic and is consistently much closer to the 14 alerts per week goal line. It can quickly adjust, and so it is impacted much less intensely from calculating the budget factor at a local low point (as can be seen at the beginning of November) and for a shorter time frame. If fact, when we compute the mean absolute difference from the 14 weekly alerts goal, we find that the hourly update frequency has by far the lowest mean of 3.8 as compared to 10.7 for the 30 day frequency.

Finally, we used a configurable simulator to assess speed performance, measured as the number of records analyzed per second. This is dependent on the number of instances, the CPU, and memory specifications. We tested our system on a local machine with 16 GB RAM and an i7 Intel 8-core processor (2.7 GHz) and were able to reach 200K records per second. Deploying the system to the cloud, we observed a near-linear scale up in performance dependent on the number of executors. Speed almost doubled when we went from a single executor to 2 and again from 2 to 4. Thus, we have confirmed our assumption of scalability under the Spark framework.
6 Summary

We presented a novel, scalable, generic, and industrially-proven system that uses anomaly detection to identify security attacks on IoT cloud platforms. The system includes a novel approach that handles the alerts, offloading the cognitive burden of the human through semi-automaton. The system uses historical data to calibrate alert thresholds and achieve the target rate on average, while still properly handling periods where anomalous behavior is exceptionally high or low.

The scalability of the system is achieved using a distributed cloud-based scalable architecture (i.e., Spark and Kafka), and efficient modeling techniques, specifically smoothing averages with gaps. The system can be deployed in multiple settings using a configuration interface to define the modeled entities (the objects that are being tracked for anomalous behavior), and the event meta types (combination of features that define distinct events). The configuration interface further allows users to select the scoring algorithms used from a library, and to extend it with new ones.

Our future work plans include continuing and enhancing the system’s use in the IoT domain, and for additional fields beyond security. We have already had success in early experiments predicting device failures in storage systems, and we plan to further explore using the outlier system for quality-related use cases. For algorithm enhancement and improved precision, we plan to introduce the clustering of modeled entities based on similarity of behavior. The anomaly detection for an individual modeled entity will consider the particular entity as well as similar entities in its cluster. More future directions include addressing adaptivity and increased automation.
Acknowledgements. Part of this work has been carried out in the scope of the FINSEC project (contract number 786727), which is co-funded by the European Commission in the scope of its H2020 program. The authors gratefully acknowledge the contributions of the funding agency and of all the project partners.

References

1. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proceedings of 20th International Conference very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
2. Alabdulatif, A., Khalil, I., Kumarage, H., Zomaya, A.Y., Yi, X.: Privacy-preserving anomaly detection in the cloud for quality assured decision-making in smart cities. J. Parallel Distributed Comput. 127, 209–223 (2019)
3. Ali, M.Q., Al-Shaer, E., Khan, H., Khayam, S.A.: Automated anomaly detector adaptation using adaptive threshold tuning. ACM Trans. Inf. Syst. Secur. (TISSEC) 15(4), 17 (2013)
4. Arrington, B., Barnett, L., Rufus, R., Esterline, A.: Behavioral modeling intrusion detection system (bmids) using internet of things (iot) behavior-based anomaly detection via immunity-inspired algorithms. In: Computer Communication and Networks (ICCCN), 2016 25th International Conference on. pp. 1–6. IEEE (2016)
5. Bertino, E., Terzi, E., Kamra, A., Vakali, A.: Intrusion detection in rbac-administered databases. In: 21st Annual Computer Security Applications Conference, pp. 10. IEEE (2005)
6. Bhuyan, M., Bhattacharyya, D.K., Kalita, J.: Network Traffic Anomaly Detection and Prevention: Concepts, Techniques, and Tools (2017). https://doi.org/10.1007/978-3-319-65188-0
7. Butun, I., Kantarci, B., Erol-Kantarci, M.: Anomaly detection and privacy protection in cloud-centric internet of things. In: 2015 IEEE International Conference on Communication Workshop (ICCW), pp. 2610–2615. IEEE (2015)
8. Cook, S.: Ddos attack statistics and facts for 2018–2020 (2020). https://www.comparitech.com/blog/information-security/ddos-statistics-facts/
9. Deorankar, A.V., Thakare, S.S.: Survey on anomaly detection of (iot)-internet of things cyberattacks using machine learning. In: 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC), pp. 115–117. IEEE (2020)
10. Eltanbouly, S., Bashendy, M., AlNaimi, N., Chkirbene, Z., Erbad, A.: Machine learning techniques for network anomaly detection: A survey. In: 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), pp. 156–162. IEEE (2020)
11. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity 2(1), 1–22 (2019). https://doi.org/10.1186/s42400-019-0038-7
12. Koufakou, A., Ortiz, E.G., Georgiopoulos, M., Anagnostopoulos, G.C., Reynolds, K.M.: A scalable and efficient outlier detection strategy for categorical data. In: 19th IEEE International Conference on Tools with Artificial Intelligence, 2007. ICTAI 2007, vol. 2, pp. 210–217. IEEE (2007)
13. Liu, C.M., Chen, S.Y., Zhang, Y., Chen, R., Guo, K.L.: An iot anomaly detection model based on artificial immunity. In: Advanced Materials Research. vol. 424, pp. 625–628. Trans Tech Publications Ltd (2012)
14. Mehnaz, S., Bertino, E.: Privacy-preserving real-time anomaly detection using edge computing. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 469–480 (2020). https://doi.org/10.1109/ICDE48307.2020.00047
15. Stellios, I., Kotzanikolaou, P., Psarakis, M., Alcaraz, C., Lopez, J.: A survey of iot-enabled cyberattacks: assessing attack paths to critical infrastructures and services. IEEE Commun. Surv. Tut. 20(4), 3453–3495 (2018)
16. Yan, X., et al.: Trustworthy network anomaly detection based on an adaptive learning rate and momentum in iiot. IEEE Trans. Ind. Inf. 16(9), 6182–6192 (2020)
17. Yu, X., et al.: An adaptive method based on contextual anomaly detection in internet of things through wireless sensor networks. Int. J. Distr. Sensor Netw. 16(5), 1550147720920478 (2020). https://doi.org/10.1177/1550147720920478