User and Entity Behavior Analytics for Enterprise Security

Madhu Shashanka*
Charles Schwab
madhu.shashanka@schwab.com

Min-Yi Shen, Jisheng Wang
Niara, Inc.
{*minyishen, jisheng*}*@niara.com*

***Abstract*—This paper presents an overview of an intelligence platform we have built to address threat hunting and incident investigation use-cases in the cyber security domain. Specifically, we focus on User and Entity Behavior Analytics (UEBA) modules that track and monitor behaviors of users, IP addresses and devices in an enterprise. Anomalous behavior is automatically detected using machine learning algorithms based on Singular Values Decomposition (SVD). Such anomalous behavior indicative of potentially malicious activity is alerted to analysts with relevant contextual information for further investigation and action. We provide a detailed description of the models, algorithms and implementation underlying the module and demonstrate the functionality with empirical examples.**

***Keywords*-Anomaly Detection; User and Entity Behavior Analytics; Singular Value Decomposition (SVD); Mahalanobis Distance**

I. Introduction

A key problem in enterprise security is to detect compromised user accounts and insiders within the company who may have malicious intent (rogue users). The variety of scenarios in which this can take place and huge variability in the characteristics of networking environments across companies makes this problem very complex. However, assuming that the actions of a compromised or rogue user are inherently different from his or her everyday job responsibilities makes this problem somewhat simpler to tackle. If each user's actions are tracked over time and against actions of other similar users, one can develop a *baseline* profile of the user's behavior and any deviations from this behavior can be flagged as potential *anomalies* that warrant further investigation. In this paper, we describe the User and Entity Behavior Analytics module of the Niara Security Analytics Platform. This module applies machine learning algorithms on diverse data sources such as network packets and logs to identify anomalous behavior of users, IP addresses and devices within an enterprise network.

There is a long history of applying anomaly detection techniques and machine learning approaches to problems in computer security. One of the earliest works can be traced back to 1987 [3]. Despite a large body of work in academia since then [2] trying to apply anomaly detection to security problems, very little has found its way to the industry. There is renewed interest in industry about user behavior analysis but there is a lot of skepticism among security practitioners and very few real-world deployments leverage machine learning [8], [4], [12], [9].

There are several reasons for this disconnect. [4] provides an excellent summary of incorrect assumptions that are commonly made about the problem domain (attacks and malicious activity are rare and anomalous), training data (attack-free data is available and the norm, simulations are representative and traffic is static) and operational usability (false alarms $> 1\%$ are acceptable, definition of malicious is universal and users can interpret anomalies). [12] identifies further characteristics they say makes security domain not well-aligned with the requirements of machine learning: very high cost of errors, lack of training data, "semantic gap" between results and their operational interpretation, enormous variability in input data, and difficulties in conducting sound evaluation.

These observations resonate with our experience as well. It is very important to carefully consider the right use-cases and have a well-defined scope. The most important factor is making sure one hasn't made incorrect assumptions about the data and designing the right features for the problem. Anomaly detection can only serve as a starting point to help users identify events of potential significance. It is crucial to provide operational context on why an event was flagged as anomalous and other supporting information. The goal is to make security analysts more efficient and effective and not just provide a new source of alerts that are not actionable. Towards this end, we believe we have built a system that provides rich supporting context and data, all the way down to the network packets, to help analysts identify and understand events of importance to the enterprise.

The paper is organized as follows. We first introduce User and Entity Behavior Analytics (UEBA) in Section II and set up the problem. Section III presents a detailed description of the core algorithms underlying the UEBA module. We then provide an overview of the Niara platform and describe details of the implementation and deployment in Section IV. We present experiments and results from a real-world dataset in Section V, and conclude with final remarks in Section VI.

* Author was with Niara, Inc. when this work was performed.

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

II. User and Entity Behavior Analytics

Behavioral analytics and anomaly detection are very broad terms. In this section, we set up the problem with a specific server access behavior use-case as a running example. There are several other types of UEBA use-cases in the product - eg., anomalies related to new values not seen in the baseline, anomalies related to geolocation etc. that are supported by appropriate machine learning algorithms underneath. But due to space constraints, we will focus on this particular use-case.

Consider an important server within an enterprise that needs to be monitored for actions from compromised accounts or rogue users. We monitor the access patterns of each user accessing the server. The entities we are interested in are all users who connect to the server. To detect an anomaly, we first need to define a baseline against which to compare. We consider two scenarios:

- Historical baseline: we evaluate a user's behavior against his or her own behavior over time in the past.
- Peer baseline: we evaluate the user's behavior against the behavior of all peers.

But first, we have to define what we mean by user *behavior*. This involves identifying the time-granularity for analysis (hourly, daily, weekly etc.) and identifying a set of *features* to characterize the pattern of access within each time-period for each user. In this example, we chose to monitor daily user access behavior and below are a subset of features that are computed daily for each user-server pair:

- timestamp of first access of the day,
- timestamp of last access for the day,
- duration between last and first access,
- sum total of durations of all eflows of the day,
- number of eflows during the day,
- total upload bytes, and
- total download bytes.

Data collected over time across all the above features serves as input to the anomaly detection algorithm. For historical baseline, data vectors for several days in history for a particular user-server pair is used as baseline data $\mathbf{X}$. Data for the same user-server pair for the test day serves as test data vector $\mathbf{x}$. For peer baseline, data vectors for all users (or users within a particular group) accessing a server on a particular day is used as baseline $\mathbf{X}$ for that day and each vector within $\mathbf{X}$ is scored against that baseline.

III. Algorithm Overview

Our approach is based on the concept of *Mahalanobis distance* [7]. We first describe the concept and detail how we adapt it to create a flexible anomaly detection algorithm that can be used in real-world scenarios to produce explainable results.

A. Mahalanobis Distance

Given a set of observations of a variable, what characterizes an anomaly? Intuitively, it is reasonable to think of an anomaly as an *unlikely* observation - a low probability event. One can look at the empirical distribution of the variable and infer the probability for any observation. If we assume that the variable has a normal distribution as a first approximation, low probability events occur at the tails of the distribution, far away from the mean value. In other words, the farther out an observation is from the mean value at the center of the distribution, the lesser the probability. Intuitively, finding such low probability observations is equivalent to finding *outliers* that are farther away in distance from values that are common. The distance from the mean can then be used an indicator of the extent or magnitude of anomaly. This distance, when expressed in terms of the standard deviation instead of absolute units, gives the *z-score* which shows how many standard deviations away an observation is from the mean value. This allows one to compare anomalies in variables that have different distributions.

Mahalanobis distance is a multi-dimensional generalization of the z-score. When there are multiple variables in each observation, the Mahalanobis distance shows how many standard deviations away an observation is from the mean value of all observations. It is unit-less and scale-invariant. Given an observation vector of N variables $\mathbf{x} = \{x_1, \ldots, x_N\}$ and a set of observations $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ with mean vector $\boldsymbol{\mu} = \{\mu_1, \ldots, \mu_N\}$ and covariance matrix $\boldsymbol{\Sigma}$, the Mahalanobis distance is given by

$$\sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})}. \tag{1}$$

Mahalanobis Distance in Practice: In real-world scenarios, true values of the mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ are not available and one has to use empirical estimates from the observations. To compute the distance, notice from equation 1 that the sample covariance matrix $\boldsymbol{\Sigma}$ cannot be ill-conditioned or singular, otherwise the inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$ cannot be computed accurately. See [5] for a detailed discussion on the numerical stability of these calculations.

To get around these constraints, we take an alternative approach that obviates the need for inverse covariance matrix computation. Let the set of observations be given by $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_K\}$ with mean vector $\boldsymbol{\mu}$ and a vector $\boldsymbol{\sigma}$ of standard deviations along each dimension. We first normalize all variables to have 0 mean and unit variance to generate

$$\bar{\mathbf{X}} = \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_K\} \text{ where } \bar{\mathbf{x}}_i = (\mathbf{x}_i - \boldsymbol{\mu}) \ominus \boldsymbol{\sigma} \tag{2}$$

where $\ominus$ indicates element-wise division. For notational convenience, let us represent this z-score transformation as

$$\bar{\mathbf{x}}_i = zsc(\mathbf{x}_i, \mathbf{X}), \tag{3}$$

where the first argument indicates the vector being transformed and the second argument indicates the data whose mean and variance information is used in the transformation. We then transform the data using Singular Values Decomposition (SVD [13]) as

$$\mathbf{U}\mathbf{S}\mathbf{V}^T = \bar{\mathbf{X}} \tag{4}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and $\mathbf{S}$ is a matrix containing singular values as its diagonal elements. The columns of $\mathbf{U}$ indicate orthogonal directions in decreasing order of variance corresponding to decreasing magnitudes of singular values in $\mathbf{S}$. In other words, the columns of $\mathbf{U}$ represent axes of a new coordinate system for the data. This can help in two ways:

- If the matrix $\mathbf{X}$ is singular i.e. one or more of the singular values are zero, then we take only those components from $\mathbf{U}$ that correspond to non-zero singular values and compute the projection. Let r denote the number of components chosen and let s_i be the i-th singular value. Then,

$$r = \sum_{\text{all } j} I(s_j > 0), \text{ where } I(\text{True}) = 1, I(\text{False}) = 0. \tag{5}$$

- If the matrix is ill-conditioned i.e. one or more of the singular values have extremely low magnitudes, one can pick a threshold - say 95% or 99% - and choose only as many components from $\mathbf{U}$ such that the sum of squares of corresponding singular values (as a percentage of the total sum) is greater than the threshold. If the threshold is given by t, $0 < t < 1$ then

$$r = \underset{i}{\operatorname{argmin}}\, i, \text{ such that } (\sum_{j=1}^{i} s_j^2 / \sum_{\text{all } j} s_j^2) \geq t. \tag{6}$$

We can now take the first r columns of $\mathbf{U}$ - let us denote it by $\bar{\mathbf{U}}$ - as the coordinate system we want to work in. Given an observation vector $\mathbf{x}$, we first preprocess it the same way the training data $\mathbf{X}$ was preprocessed - by subtracting the sample mean and dividing by the sample standard deviation. We take the preprocessed vector $\bar{\mathbf{x}}$ and work with its projection $\mathbf{y}$ given by $\mathbf{y} = \bar{\mathbf{U}}^T\bar{\mathbf{x}}$. The *Mahalanobis* distance of $\mathbf{x}$ from observations in matrix $\mathbf{X}$ can then be calculated as

$$\mathbf{y}^T\bar{\mathbf{S}}^{-2}\mathbf{y} \tag{7}$$

where $\bar{\mathbf{S}}$ is a diagonal matrix of the first r singular values from $\mathbf{S}$.

To summarize, all the steps in the algorithm are shown in table I.

Table I
ALGORITHM

Step 1.	Inputs: Baseline matrix $\mathbf{X}$, test vector $\mathbf{x}$
Step 2.	z-score $\mathbf{X}$ and preprocess $\mathbf{x}$: $\bar{\mathbf{X}} \leftarrow zsc(\mathbf{X}, \mathbf{X})$, $\bar{\mathbf{x}} \leftarrow zsc(\mathbf{x}, \mathbf{X})$
Step 3.	SVD: $\bar{\mathbf{X}} \rightarrow \mathbf{U}\mathbf{S}\mathbf{V}^T$
Step 4.	Number of components: compute r from Eqn. 5 or 6
Step 5.	Reduce to first r components: $\mathbf{U} \rightarrow \bar{\mathbf{U}}$, $\mathbf{S} \rightarrow \bar{\mathbf{S}}$
Step 6.	Project to SVD space: $\bar{\mathbf{U}}^T\bar{\mathbf{x}} \rightarrow \mathbf{y}$
Step 7.	Compute distance: $\mathbf{y}^T\bar{\mathbf{S}}^{-2}\mathbf{y}$.

B. Enhanced Mahalanobis Distance

We extend the approach described above in several ways to make it more flexible and appropriate for our use-cases.

1) One Sided Deviations: A main requirement for most security use-cases is to find cases where the deviations are one-sided. For example, in the case of monitoring download activity from a sensitive internal server, one might not really care if somebody downloads *less* than what is normal but would want to know if the download magnitude is really large. But Mahalanobis distance will flag deviations from normal activity equally on both the low and high sides.

We extend the approach to provide an optional parameter for each of the N variables to specify if deviations have to be ignored in the positive or negative direction from the mean. To be more precise, the user specifies an optional vector of length N containing entries -1, 0 or 1. If the i-th entry is -1, values of the i-th variable that are less than the mean will not contribute towards the Mahalanobis distance computation; if the value is +1, values greater than the mean will not be considered. If the value is 0, values both above and below the mean will be considered.

Let $\mathbf{v}$ denote this vector of parameters with v_i being the i-th entry. Referring back to Table I, we introduce an additional step before projecting the preprocessed test data vector onto the SVD space in Step 6. Let $\bar{x}_i$ refer to the i-th entry of the preprocessed vector $\bar{\mathbf{x}}$. We modify $\bar{\mathbf{x}}$ as follows:

$$\bar{x}_i = 0 \text{ if } \bar{x}_i v_i > 0 \qquad \forall \quad i, \quad 1 \leq i \leq N. \tag{8}$$

In this equation, we check if a variable has deviation in a direction that is not of interest. We then change that entry to the mean value, which for a z-scored variable is equal to 0. We then follow steps 6 and 7 to compute the distance.

2) Variable Weighting: Another common requirement in some use-cases is the ability to provide different weights to different variables. For example, if we want to monitor users for extended hours of server access but do not care

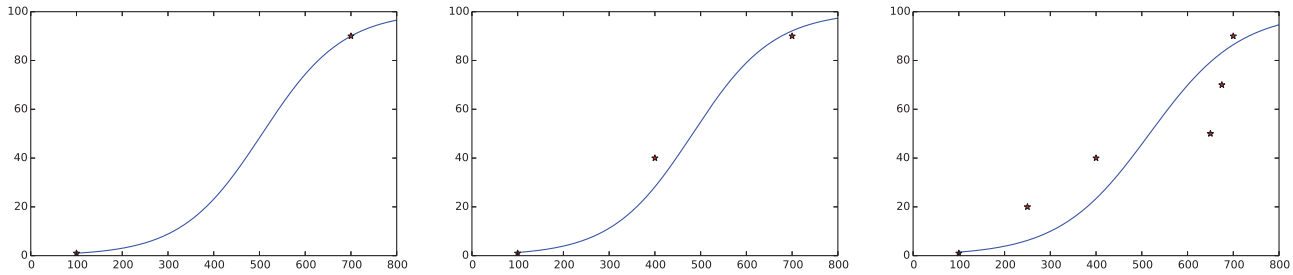

Figure 1. Illustration of mapping Mahalanobis distance to a confidence score between 0 and 100. In each panel, the blue curve indicates the sigmoid fit on data points denoted by red stars. The Mahalanobis distance values are on the x-axis and confidence scores are on the y-axis. Writing the data points as a set of (x, y) tuples, the data points are [(200, 1), (700, 90)] in the left panel, [(100, 1), (400, 40), (700, 90)] in the middle panel, and [(100, 1), (250, 20), (400, 40), (650, 50), (680, 75), (700, 90)]. The data-points can be weighted differently as well. In the third panel, the point (250, 20) has a weight of 0.4 and (650, 50) has a weight of 0.2 as compared to all the other data points with weights of 1.0.

much about their download activity, this extension provides the ability to have time variables contribute more to the Mahalanobis distance when compared to the byte-related variables such as download and upload activity.

Let $\mathbf{w}$ be the vector of weights where w_i is the weight for the i-th variable. The idea is to increase the variance of the i-th variable by a factor w_i. To implement this, refer back to Step 2 in Table I. After $\mathbf{x}$ is processed to obtain $\bar{\mathbf{x}}$, we scale the entries as follows

$$\bar{\mathbf{x}} = \bar{\mathbf{x}} \odot \mathbf{w} \tag{9}$$

where $\odot$ is element-wise multiplication.

3) Robustness to Outliers: Computing the Mahalanobis distance using equation 1 has an important practical shortcoming. Since the true mean $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are typically unknown, in practice one uses empirical estimates from the dataset. However, the estimate of the covariance matrix is very sensitive to the presence of outliers in the data. There are heuristic approaches that aim to find a subset of points in the dataset that are "pure" and exclude the outliers. It works iteratively to find points whose covariance matrix has the smallest determinant [10], [11].

Because of this and other reasons mentioned previously, we take an alternative approach using SVD to compute the distance. However, SVD is also sensitive to outliers and the decomposition obtained from equation 4 can be skewed because of the presence of outliers in $\bar{\mathbf{X}}$.

We take inspiration from recent work on robust versions of Principal Component Analysis (PCA, [1]). PCA, closely related to SVD, is a commonly used technique for dimensionality reduction and data analysis. Given a matrix of noisy measurements (images, sensor data etc.), PCA is used to find a low-rank matrix that is used as a proxy for the true values of the quantities being measured. This works well under the assumption that the noise is small and the noise errors follow an independent and identically distributed normal distribution. This is not the case in many practical situations and Robust PCA or RPCA [1] was introduced to handle cases where assumption on the noise matrix is not satisfied. The intuition of RPCA is simple. Given a measurement matrix $\mathbf{X}$, RPCA aims to identify additive components

$$\mathbf{X} \approx \mathbf{A} + \mathbf{E} \tag{10}$$

where $\mathbf{A}$ is a low-rank matrix and $\mathbf{E}$ is a sparse matrix. $\mathbf{A}$ is interpreted as the true values and $\mathbf{E}$ is the error matrix. Several algorithms have been proposed and the idea is to cast the problem of separating the low-rank data component from the sparse noise component into a convex optimization problem. We use the *inexact augmented lagrange multiplier* algorithm [6], an iterative method that typically converges in less than 100 iterations.

Coming back to the problem of outliers skewing SVD, the goal - similar to goal of the minimum covraince determinant estimator - is to identify a subset of points in the dataset that does not contain any outliers. Referring back to Table I, we introduce an additional step before performing SVD in Step 3. Let ρ be the number of points we would like in the subset, this should be greater than half the number of total points in the entire dataset. We first perform RPCA on the preprocessed data matrix,

$$\bar{\mathbf{X}} \approx \hat{\mathbf{X}} + \mathbf{E}. \tag{11}$$

Compute the vector $\mathbf{err}$ to be the sum of absolute values of entries in $\mathbf{E}$, i.e. $err_i = \|\mathbf{e}_i\|_1$, $1 \leq i \leq K$ corresponding to the error of the i-th observation vector. We would like to keep only the first ρ vectors with the lowest errors. Let err be the ρ-th smallest value in $\mathbf{err}$. We retain $\bar{\mathbf{x}}_i$ in $\bar{\mathbf{X}}$ if $err_i \leq err$, otherwise we remove the vector from $\bar{\mathbf{X}}$. After obtaining the reduced dataset in $\bar{\mathbf{X}}$, we perform Step 2 again before proceeding with the SVD.

4) Explainable Results: An important aspect of any anomaly detection scheme is to not only identify the anomaly but also provide information on why the detected points are anomalous. In our case, we show how much each variable contributes to the anomaly score.

Given the preprocessed test vector $\bar{\mathbf{x}} = \{\bar{x}_1, \ldots, \bar{x}_N\}$, the contribution of variable j is given by

$$c_j = \bar{x}_j^2 / \sum_{i=1}^{N} \bar{x}_i^2. \quad (12)$$

5) Mapping the Distance to a Score: Mahalanobis distance is not bounded from above, the computed distances can be arbitrarily large. However, one of the product requirements is to generate a confidence score for each anomaly that is bounded in the interval $[0, 100]$. We map the mahalanobis distance into a confidence score by using a sigmoid function. If m is the distance, the final score is given by

$$score = \frac{100}{1 + e^{-k(m-m_0)}}, \quad (13)$$

where k is the steepness of the curve and m_0 is the distance for which the score is 50.

Fixing values for the parameters k and m_0 will specify the exact mapping from the Mahalanobis distance to a confidence score. We compute the parameters empirically by using a linear fit given desired $score$ for at least two different values of m. We fit the equation below:

$$s = -k \times m + k \times m_0, where \quad s = \log \frac{100 - score}{score}. \quad (14)$$

It can be easily verified that the value for k is given by the negative slope of the fit and the value for m_o is given by $intercept/k$. Figure III-B shows example results of this process.

6) Learning from User Feedback: We have also implemented an optional feature where the algorithm can take feedback from users. For a given test vector, a user can provide feedback that the computed anomaly score is too high (in case of a false positive) or too low.

Handling false positives can be easily implemented without making changes to the algorithm. In the algorithm as mentioned in section III-B3, we make sure the training set does not contain any outliers or anomalies. When a user marks a test vector that is scored as an anomaly by the algorithm, it means that the test vector can be considered part of the baseline. In other words, we can use this test vector marked as a false positive as part of the training data. The idea is to maintain a list of false positive data and include them during training to make sure similar data vectors don't raise anomalies in the future. However, this approach cannot extend to the case where a user thinks the generated anomaly score is too low.

To adapt to both kinds of feedback (anomaly score too high or too low), we look at the contribution from each variable to the score, and increase or decrease the weights of each variable in proportion to the corresponding contributions. For example, if a user marks a test vector as a false positive with a score that is too high, the intuition is to look at the variables that contribute the most and reduce their weights. Similar vectors in future sessions will have lower scores because of the lowered weights on the high-contribution variables.

Let δ be a tunable parameter between 0 and 1 that indicates to what extent we want the weights changed based on a single feedback. Let f indicate user's feedback - it takes value +1 if the user thinks the score is too low and -1 if the user thinks the score is too high. We modify the weight vector $\mathbf{w}$ as follows,

$$\mathbf{w}_{new} = \mathbf{w} + f\delta\mathbf{c}. \quad (15)$$

To make sure the weights don't grow too high or too low, it is a good idea to have limits on the maximum and minimum values.

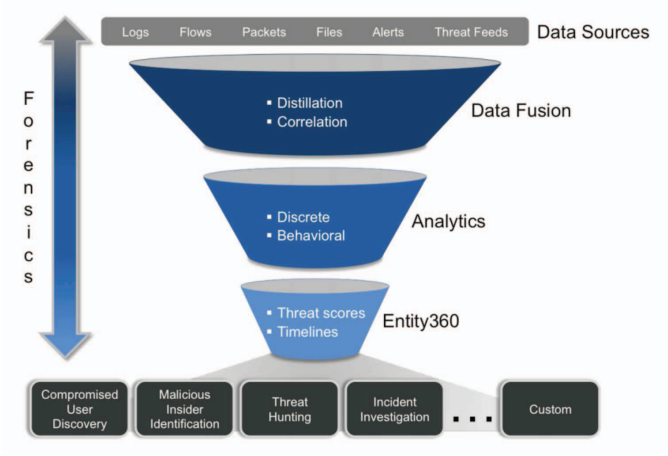

Figure 2. Overview of the Niara Security Analytics Platform

IV. Solution Overview and Implementation

In this section, we provide an overview of the Niara platform and describe details of the UEBA module architecture and implementation.

A. Niara Security Analytics Platform

The Niara Platform is an enterprise security analytics solution aimed at threat hunting, incident investigation, insider threat detection and related use-cases. The goal is to provide security analysts automated detection of attacks that have bypassed perimeter defenses and are on the inside. Activities such as command and control, internal reconnaissance, lateral spread, privilege escalation and exfiltration indicative of potentially malicious activity are detected automatically. Figure 2 provides an overview of the conceptual architecture of the platform.

Entity360: The core idea underlying the platform is the concept of an *Entity360*. All information related to a particular entity - a user, an IP address, or a device - from a variety of data sources are brought together to generate a comprehensive risk profile in the entity360. It provides a coherent visual representation of all the enriched security information that is associated with an entity. It is meant to provide one-click access to information that security analysts would otherwise spend hours or days searching across multiple data silos and assembling as part of any investigation and response effort. Synthesizing entity360's is a multi-step process - the first step is bringing information from a variety of data sources together in a *data fusion* step, and a variety of *analytics* modules are then applied to extract specific security insights.

Data Fusion: The platform can ingest raw data from a variety of network and security data sources (eg., packets, flows, logs, files, alerts, threat feeds). During data fusion, raw data is *correlated* to make it more meaningful (eg., associating IP addresses with users) and distilled into summaries that provide rich context (eg., authentication and device usage histories, port-protocol relationships). The platform is built on a robust big data architecture that can handle varying velocities in the arrival of data from different sources at scale.

Analytics: Analytics modules use machine learning models - both supervised and unsupervised - to detect and identify events of importance to the security analyst. These modules contribute to an entity's overall risk score which is tracked over time. There are several analytics modules in the platform but they can be broadly grouped into two categories - *behavioral* analytics and what we have termed as *discrete* analytics. Discrete analytics modules are mostly based on supervised machine learning algorithms and are geared towards detecting and identifying the *known unknowns*. Examples include identifying malware infections, suspicious executable files etc. On the other hand, behavioral analytics modules are mostly based on unsupervised machine learning algorithms to address the more complex problem of detecting the *unknown unknowns* such as malicious or compromised insiders.

Events and Alerts: When an analytics module successfully detects malicious activity or an anomaly, it triggers an *event*. Each event is associated with one or more entities and comes with two scores - (i) a *severity* score that is defined by the security analyst to indicate the business context and importance, and (ii) a *confidence* score generated by the module that can be thought of as a probability that the module is correct in its detection. Values for both scores are in the range from 0 to 100. For example, an analyst may choose to have a *severity* of 100 for all behavioral events of the CEO of the company while choosing to set it to a low score of 40 for events related to a non-critical server that does not connect to any other machines. Similarly, when a module is sure about a malware detection, the generated *confidence* score will be closer to 100 and when it is not certain, it may still trigger an event but will a low score. Events with both *severity* and *confidence* scores greater than 60 are elevated as *alerts*. All events and alerts for an entity are brought together in its *entity360* and the severity and confidence scores are combined to generate a unified alert threat score.

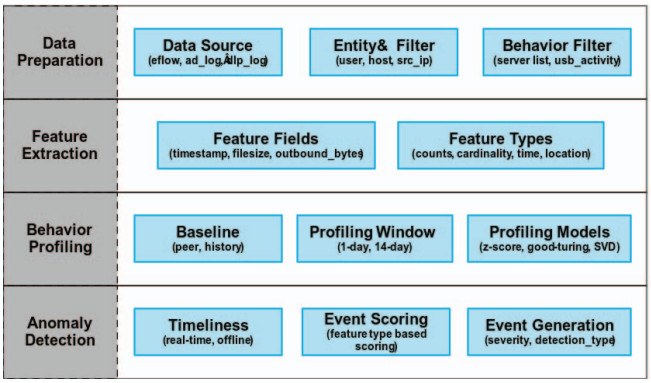

Figure 3. Generic Architecture for UEBA

B. UEBA Implementation

Figure 3 summarizes the architecture of the UEBA module. The architecture supports other types of UEBA use-cases but we will focus on server-access behavior anomaly detection based on SVD. An analyst will first have to configure the use-case by specifying (a) the type of entities to focus on - users, IP addresses or devices, (b) the IP address(es) or hostname(s) of the internal server of interest, (c) the baselines to be used, and optionally (d) any other filters on data. An example use-case could be to monitor accesses to the internal finance server (server of interest) by users (entity) within the finance group (additional data filter). In addition, the analyst will define the features to be used, we presented example features in Section II. We omit details of the use-case configuration workflow as it is out of scope for this paper.

The entire workflow can be broken down into four distinct phases. We will describe each one of them below.

1) **Data Preparation** - In the first step, the workflow obtains relevant data from all the data sources. It applies all the defined filters, groups data by identified entities and prepares data for the next feature extraction stage.
2) **Feature Extraction** - In this step, data is obtsained from all the relevant fields, grouped by each entity per day, and the configured features are computed and stored.

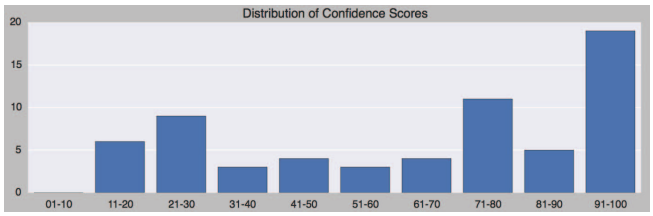

Figure 4. Distribution of Confidence Scores. The x-axis denotes bins of confidence scores and y-axis shows the number of events with confidence scores in the corresponding bins.

3) **Behavior Profiling** - This is the step where for each entity, the extracted features are grouped into configured baselines and the machine learning model (SVD) is applied to generate a *behavior profile* for that particular entity.
4) **Anomaly Detection** - In the final step, the test feature values are scored against the behavior profile and an event is generated with an associated confidence score.

The entire workflow runs once every day on Apache Spark.

V. Experiments and Results

The UEBA module has been deployed at multiple customer sites as part of the Niara platform. In this section, we present results on a real-world dataset of network traffic collected within the Niara internal network over a span of a 3 months from Nov 2015 through Jan 2016. The entire dataset comprises 1,315,895,522 (1.3 Billion) raw data records where each record corresponds to a network layer-4 conversation.

We present results on a specific server access behavior use-case. We tracked accesses to our internal Jenkins server by specifying the server's IP and port (8080). During the 3-month period, there were 362,791 conversations in total originating from various internal IP addresses to the Jenkins server. We configured the UEBA module to detect anomalies for this use-case using both peer and historical baselines. There were 64 *events* that were generated out of which 39 were *alerts* with a confidence score greater than 60. Table II shows the number of events and alerts by baseline type, and Figure 4 shows the distribution of confidence scores.

Table II
Number of Events and Alerts by Baseline Type

	Events	Alerts
History	12	03
Peer	52	36

Figure V shows an event and an alert generated for the same admin user accessing the Jenkins server on Jan 14th. This example demonstrates the value of scoring an entitity's behavior against both historical and peer baselines. While based on the activity of peers, the admin's actions look anomalous but based on the historical behavior, it was beningn behavior. The anomaly scores generated are based on values of all features but the primary feature contributor is "download bytes." Figure V shows two alerts where the primary feature contributors were related to the time of access.

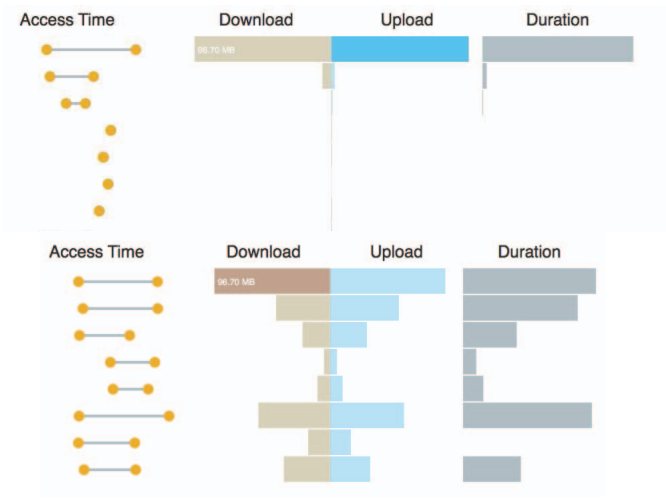

Figure 5. The two panels above show visualizations of features for an alert (top, confidence score 96) and an event (bottom, confidence score 16) generated for the same user on Jan 14th. In both panels, the first row shows features of the user that was alerted on that day while the remaining rows show features of the baselines. In the top panel, peers were used as a baseline while the user's own historical features were used in the bottom panel. Notice that compared to peers, the behavior was anomalous while compared to the user's own past behavior, Jan 14th was a normal day. The user was an admin of the Jenkins server and the deviation in behavior compared to peers was expected. The "download bytes" feature is the most contributing feature for this anomaly.

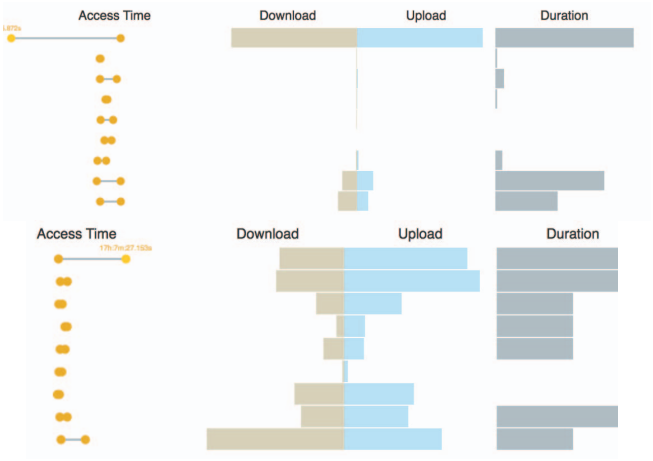

Figure 6. The two panels above show alerts for an admin (top, confidence score 98, peer baseline) and a user (bottom, confidence score 76, historical baseline), where the most contributing feature was the first or last time of access during the day.

Discussion

There are several issues that one needs to pay particular attention to during deployment. Below, we describe a few of them in no particular order.

Size of training set: For results to be meaningful, the absolute minimum number of records necessary for training should be atleast the number of features used. But we are also limited by practical deployment constraints where we cannot wait for weeks to gather training datasets. With seven features in our current implementation, we do not generate any anomaly score until we have atleast 7 training records. For historical anomalies, we start with the previous 7 days and use upto the previous 14 days of data for training. For the current implementation of peer anomalies, there is no upper bound on the number of records in the training set.

Impact of variance in training data: If values for a particular feature are too consistent in the training data, the variance will be very small and small deviations can result in extremely high anomaly scores. *The anomaly score value should always be looked at in the context of the training data*. For example, if an employee begins the day at the same exact time every day and comes in a minute late on the scoring day, it will raise an anomaly with a very high score. A corollary is that the units used to express features can be very important. During tuning and experimentation, we had initially expressed first and last access times in minutes. In certain cases, this artificially resulted in low variance in the training dataset since we had lost the fidelity at the level of seconds and generated a few false positive events. We then started using seconds to express these two features.

Feature Weighting: Extreme care should be taken while assiging different weights to different features. Unless the use-case demands it, all features should be given equal weights. In our examples, we provided higher weights to the ”time-of-access” features compared to the ”bytes” features. The values for first and last access times of the day are always between 0 and 24 hours as opposed to download or upload bytes which exhibit much higher variances. As a result, high variances in the byte related features could overwhelm deviations in access times that were of interest to our internal security analysts. We cannot disclose the weight values as the information is proprietary.

VI. Conclusions

In this paper, we presented the User and Entity Behavior Analytics (UEBA) module of the Niara Security Analytics Platform. We provided an overview of the solution and presented details of the SVD-based algorithms to detect anomalies of interest to security analysts. We described the architecture and implementation details and showed the effectiveness of the solution with example empirical results on real-world data. The solution is currently deployed at multiple customer sites and has provided increased visibililty and insight to security teams about their networks and users. In terms of follow-on work, we have generalized the deployment architecture to enable customers to self configure new use-cases on arbitraray logs of their choice. We have also enhanced the module with more machine learning models and validated them on real-world datasets. We will report these developments in future papers.

VII. Acknowledgments

We would like to thank Prof. Ankur Teredesai from the University of Washington for reviewing the draft and providing valuable feedback.

References

[1] E. Candes, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Jrnl. of the ACM*, 58(3), 2011.

[2] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, Sep 2009.

[3] D. Denning. An intrusion detection model. *IEEE Trans. on Software Engg.*, 13(2), 1987.

[4] C. Gates and C. Taylor. Challenging the anomaly detection paradigm: A provocative discussion. In *Proceedings of the 2006 Workshop on New Security Paradigms*, NSPW ’06, pages 21–29, New York, NY, USA, 2007. ACM.

[5] A. Ker. Stability of the mahalanobis distance: A technical note. Technical Report CS-RR-10-20, Oxford University Computing Laboratory, 2010.

[6] Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. Technical Report UILU-ENG-09-2214, UIUC, 2010.

[7] P. Mahalanobis. On the Generalised Distance in Statistics. In *Proceedings of the National Institute of Science, India*, volume 2, pages 49–55, 1936.

[8] A. Pinto. Secure because math: A deep-dive on machine learning based monitoring. In *Black Hat Briefings USA*, 2014.

[9] K. Rieck. Computer security and machine learning: Worst enemies or best friends? In *SysSec Workshop (SysSec), 2011 First*, pages 107–110. IEEE, 2011.

[10] P. Rousseeuw. Least median of squares regression. *Jrnl. of Am. Stat. Assn.*, 1984.

[11] P. Rousseeuw and K. Van Driessen. A fast algorithm for the minimum covariance determinant estimator. In *Technometrics*, volume 41, Aug 1999.

[12] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 305–316. IEEE, 2010.

[13] G. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 35-4:551–566, Dec 1993.