AI^2 : Training a big data machine to defend

Kalyan Veeramachaneni
CSAIL, MIT Cambridge, MA

Ignacio Arnaldo
PatternEx, San Jose, CA

Alfredo Cuesta-Infante, Vamsi Korrapati, Costas Bassias, Ke Li
PatternEx, San Jose, CA

Abstract

We present an analyst-in-the-loop security system, where analyst intuition is put together with state-of-the-art machine learning to build an end-to-end active learning system. The system has four key features: a big data behavioral analytics platform, an ensemble of outlier detection methods, a mechanism to obtain feedback from security analysts, and a supervised learning module. When these four components are run in conjunction on a daily basis and are compared to an unsupervised outlier detection method, detection rate improves by an average of 3.41×, and false positives are reduced fivefold. We validate our system with a real-world data set consisting of 3.6 billion log lines. These results show that our system is capable of learning to defend against unseen attacks.

1 Introduction

Today, information security solutions generally fall into two categories: *analyst*-driven, or *unsupervised machine learning*-driven. Analyst-driven solutions rely on rules determined by fraud and security experts, and usually lead to high rates of undetected attacks (*false negatives*), as well as delays between attack detection and implementation of preventative countermeasures. Moreover, bad actors often figure out current rules, and design newer attacks that can sidestep detection.

Using *unsupervised machine learning* to detect rare or anomalous patterns can improve detection of new attacks. However, it may also trigger more *false positive* alarms and alerts, which can themselves require substantial investigative efforts before they are dismissed. Such false alarms can cause *alarm fatigue* and *distrust*, and over time, can cause reversion to *analyst-driven* solutions, with their attendant weaknesses.

We identified three major challenges facing the information security industry, each of which could be addressed by machine learning solutions:

Lack of labeled data: Many enterprises lack labeled examples from previous attacks, undercutting the ability to use supervised learning models.

Constantly evolving attacks: Even when supervised learning models are possible, attackers constantly change their behaviors, making said models irrelevant.

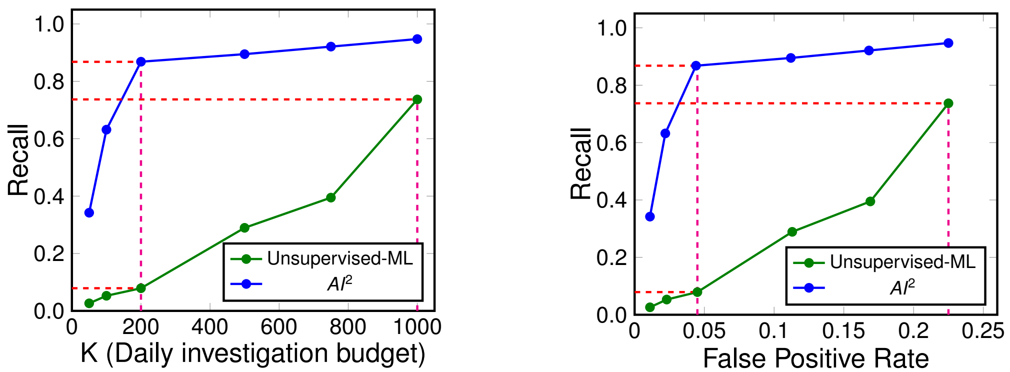
Limited investigative time and budget: Relying on analysts to investigate attacks is costly and time-consuming.

A solution that properly addresses these challenges must use analysts' time effectively, detect new and evolving attacks in their early stages, reduce response times between detection and attack prevention, and have an extremely low false positive rate. We present a solution that combines analysts' experience and intuition with state-of-the-art machine learning techniques to provide an end-to-end, artificially intelligent solution. We call this system AI^2. AI^2 learns and automatically creates models that, when executed on new data, produce predictions as intelligent as those deduced by human analysts. Backed by big data infrastructure, we achieve this in close to real time.

Our contributions through this paper are as follows:

1. Developed an *Active Model Synthesis* approach, which:
   (a) computes the behaviors of different entities within a raw big data set,
   (b) presents the analyst with an *extremely* small set of events ($k <<< N$), generated by an unsupervised, machine learning-based outlier detection system,
   (c) collects analyst feedback (*labels*) about these events,
   (d) learns supervised models using the feedback,
   (e) uses these supervised models in conjunction with the unsupervised models to predict attacks, and
   (f) continuously repeats steps (a) - (e).
2. Designed multivariate methods that are capable of modeling the joint behaviors of mixed variable types (numeric and discrete ordinal). These methods include density-based, matrix decomposition-based, and replicator neural networks.
3. Demonstrated performance of the AI^2 system by monitoring a web-scale platform that generated millions of log lines per day over a period of 3 months, for a total of 3.6 billion log lines.

Summary of results: In Figure 1, we present a snapshot of our system's progress after 12 weeks of use. With 3 months' worth of data, and with awareness of attacks, we evaluate



(a) Recall versus daily investigation budget

(b) Recall versus false positive rate

Figure 1: Recall versus bandwidth and recall versus false positive rate of the Active Model Synthesisand unsupervised outlier analysis after 3 months of deployment.

whether our solution can improve attack detection rates (*recall*) while reducing the number of alerts shown to the analyst ("*daily investigation budget*" k).

Using analyst time effectively: The AI^2 system achieves a detection rate of 86.8% even at an extremely low daily investigative budget of $k = 200$ events. This represents more than tenfold improvement[1] over the unsupervised outlier detection approach rate, which is 7.9%. Fixing the daily investigation budget at 200 keeps the false positive rate at 4.4%.

Reducing false-positives by a factor 5: If we allow for an higher daily investigative budget (for example, up to 1000), the unsupervised outlier detection based method can still only achieve a 73.7% detection rate, and the false positive rate is $> 22\%$. AI^2 achieves $> 86\%$ for a false positive rate of 4.4% a reduction by factor of 5.

On our choice of the title "Training a big data machine to defend": We define a big data system or machine as a software infrastructure that is able to ingest data in real time, compute and generate quantities that can then be analyzed, either by data scientists or a machine learning system. A machine learning substrate that sits on top of this system can analyze the data and automatically produce outliers. We provide a system that collects and incorporates analyst feedback, generates, and uses these models continuously without any involvement from its original developers - that is us. Thus, we are able to deliver a fully automatic system that could be trained by analysts.

In Section 2, we present an overview of the system, and the challenges encountered while building it. Section 3 summarizes related work in this area and Section 4 describes the data analyzed by our platform. In Section 5 we present the big data platform for behavioral analytics. Section 6 presents the outlier detection system. With the two key components in place, Section 7 presents the active model synthesis framework. Section 8 presents the experimental setup and the results achieved. Section 9 presents our key findings and conclusions.

2 AI^2

In this paper, we present an end-to-end system that learns over time thanks to feedback from security analysts. Figure 2 presents a schematic of our system, which is made up of the following components:

- Big data processing system: A platform that can quantify the behaviors (a.k.a *features*) of different entities, and compute them from raw data. With high-volume, high-velocity data, this first component requires processing at a challenging scale. We describe this system and what it accomplishes in Section 5
- Outlier detection system: This system learns a descriptive model of those features extracted from the data *via* unsupervised learning, using one of three methods: density, matrix decomposition, or replicator neural networks. To achieve confidence and robustness when detecting rare and extreme events, we fuse multiple scores into a final score that indicates how far a certain entity or event is from others. These methods are described in detail in Section 6.
- Feedback mechanism and continuous learning: This component incorporates analyst input through a user interface. It shows the top k outlier events or entities, and asks the analyst to deduce whether or not they are *malicious*. This feedback is then fed into the supervised learning module. The value of k and the feedback frequency (e.g. *daily* or *weekly*) are both decided by the end user.
- Supervised learning module: Given the analyst's feedback, the supervised learning module learns a model that predicts whether a new incoming event is *normal* or *malicious*. As more feedback is gathered, the model is constantly refined.

3 Related Work

Our work exploits ideas from a wide range of fields, including outlier analysis, ensemble learning, active learning, information security, behavioral analytics, and big data computing.

Outlier analysis methods have been reviewed in Hodge and Austin [2004], Chandola *et al.* [2009] and Aggarwal [2013a]. Our platform integrates outlier detection methods based on

[1] This result corresponds to the 12th and last week of deployment while the 3.41× improvement claimed in the abstract is the average improvement over the 12 weeks.

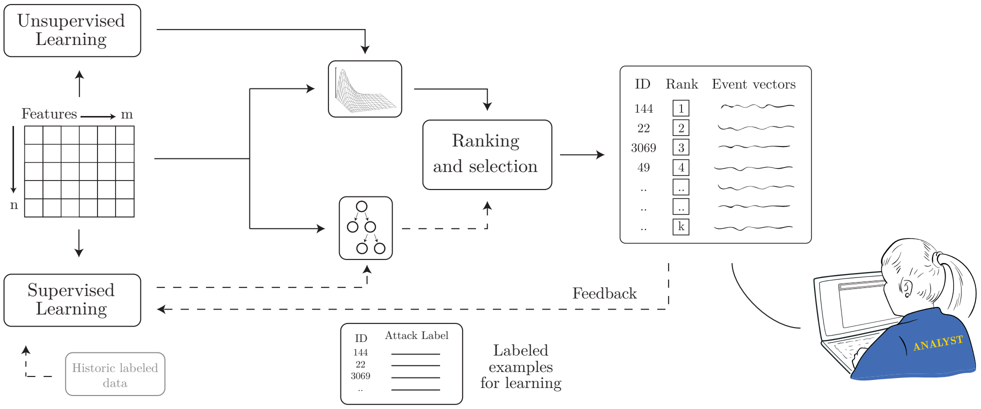


Figure 2: **AI^2.** Features describing the entities in the data set are computed at regular intervals from the raw data. An unsupervised learning module learns a model that is able to identify extreme and rare events in data. The rare events are ranked based on a predefined metric, and are shown to the analyst, who labels them as 'normal' or as pertaining to a particular type of attack. These "labels" are provided to the supervised learning module, which produces a model that can then predict, from features, whether there will be attacks in the following days.

Principal Component Analysis (Shyu *et al.* [2003]), neural networks (Hawkins *et al.* [2002]; Scholz and Vigário [2002]; Scholz *et al.* [2008]), and statistical models.

Ensemble learning can enhance the robustness of outlier analysis (Schubert *et al.* [2012]; Aggarwal [2013b]; Zimek *et al.* [2014]). This approach has received attention only recently, due to two main constraints: first, it is difficult to interpret and compare outlier scores retrieved with different methods (Gao and Tan [2006]; Kriegel *et al.* [2011]), and second, it is difficult to weight confidence in different methods, since there is no ground truth that can be used for learning. In fact, most works assume a semi-supervised setting, where a set of labels is initially available (Micenková *et al.* [2014]).

The active learning framework (Seung *et al.* [1992]) has been applied to the outlier analysis task in the past (Pelleg and Moore [2004]; Abe *et al.* [2006]). In these works, the most ambiguous examples are shown to the user for labeling. This approach is in line with the uncertainty sampling method introduced in Lewis and Catlett [1994].

Behavioral predictive analytics have shown promising results for network intrusion (Yen [2011]) and internal threat detection (Senator *et al.* [2013]). However, to the best of our knowledge, we present the first big data security system capable of detecting threats in real time, and of collecting analysts' feedback to improve detection rates over time.

4 Data characteristics

In this section, we present the typical characteristics of the data ingested by our platform (also summarized in Table 1).

Data sources and applications: Our platform processes both web logs and firewall logs. In a typical enterprise system, these logs are delivered in real, streaming time from widely distributed data sources. Web log analysis is aimed at the detection of web attacks, while mining firewall logs allows us to prevent data ex-filtration in enterprise setups.

Application	Web attacks prevention	Data exfiltration prevention
Source	Web logs	Firewall logs
Log lines (per day)	20M-200M	20M-200M
Unique entities (per day)	100K-10M	10K-5M
Malicious entities (per day)	0-100K	0-500
Concurrent active entities (per minute)	50-50K	50-10K

Table 1: Summary of the characteristics of the data ingested by our platform. The provided ranges are defined according to the values seen at the different enterprise setups where our platform has been deployed.

Data dimensions and unique entities: The computational effort associated with analyzing data can be reasonably estimated by the *data size* and the *number of unique entities*. The first refers to the volume of the raw data, and is generally reported in the form of size metrics (GB, TB) and/or number of log lines (for instance, a midsized enterprise platform easily generates tens of millions of log lines on a daily basis). The second is specific to behavioral analytics, and corresponds to the number of unique entities (IP addresses, users, sessions, etc) analyzed on a daily basis.

The data set used in this paper corresponds to three months' worth of logs, generated by an enterprise platform. This platform records millions of log lines per day, each corresponding to a specific user interaction, and has hundreds of thousands of daily active users. Table 1 presents the typical ranges we see in our current use cases.

Malicious activity prevalence: Under normal circumstances, malicious activities are extremely rare in an enter-

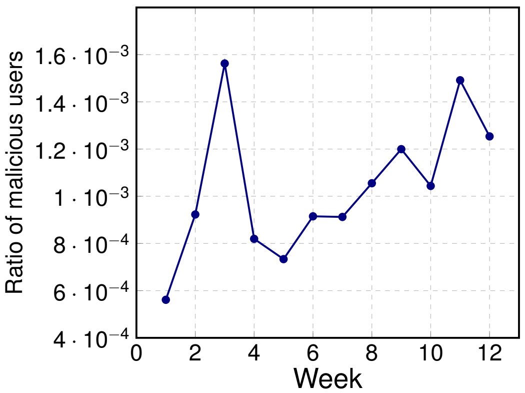


Figure 3: Weekly ratio of reported malicious users to the total number of active users.

prise setup. Attack cases represent a minor fraction of total events (generally $< 0.1\%$). To illustrate this fact, Figure 3 shows the ratio of reported malicious users to the total number of active users in the studied dataset. Three additional observations are worth remarking on:

- This dearth of malicious activity results in extreme class imbalance when learning a supervised model, and increases the difficulty of the detection process.
- It is safe to assume that not all malicious activities are systematically reported, either because their incident responses were inconclusive, or because they were not detected in the first place. This introduces noise into the data, since unreported attacks will be considered legitimate activity.
- Attack vectors can take a wide variety of shapes. Even when malicious activities are reported, we are not always aware of the specific vectors involved. Thus ,it is important to develop robust defense strategies that are capable of detecting as many attacks as possible.

ok

5 BigData platform for behavioral analytics

Our approach rests on the computation of behavioral descriptors for different entities, such as IP addresses, users, or sessions. These entities can be independent or connected; for instance, the same IP address may be associated with two or more users.

5.1 Behavioral signatures

A typical attack has a *behavioral signature*, which comprises the series of steps involved in committing it. The information necessary to quantify these signatures is buried deep in the raw data, and is often delivered as logs. These quantitative values can be specified by security experts, and generally correspond to indicators an expert would use to investigate an attack. Also known as *variables* or *features* in the field of machine learning, they are usually extracted on a per-entity, per-time-segment basis. Using the platform, we calculate a total of 24 variables per user per day. Take, for example, the *per-user* behavioral features shown in Figure 4. A platform capable of computing behavioral features in real time is itself valuable, since it allows analysts to monitor applications more dynamically.

5.2 Design requirements

A big data system for real-time behavioral analytics on web-scale applications must meet the following criteria:

1. Capable of analyzing the behavior of 10+ million entities on a daily basis.
2. Capable of updating and retrieving the behavioral signatures of active entities, on demand and in real time. The platform needs to be able to retrieve behavioral signatures for up to 50 thousand entities at once.

In the following section, we describe the key aspects that enable our big data system to process logs from many data sources, extract entities, transform raw logs into features, and keep these features up-to-date in real time. The system is designed to horizontally scale, in order to address billions of log lines per day.

5.3 From raw logs to behaviors in real time

To calculate behavioral features for one user over a particular time segment, one must isolate all relevant historic log lines and perform the aggregations that feature definition demands—for example, aggregating the money this user spent during that time segment. This process must be repeated for all the active users, populating the entity-feature matrix as shown on the right hand side of Figure 4. Such computations are challenging because of high volume, distributed storage of data, and the need to aggregate over historical data to compute the feature. We address this challenge by breaking the extraction of features into two processes: *Activity Tracking* and *Activity Aggregation*.

Activity Tracking: As the system absorbs the log stream generated by the platform, it identifies the entities involved in each log line (e.g. IP address, user, etc.) and updates the corresponding *activity records*. These *activity records* are calculated and stored according to two guidelines:

1. A very short temporal window. In our experiments, the temporal window over which these activity records are computed and stored is one minute. This way, we can compute behavioral features for different time intervals - 30 minutes, 1 hr, 12 hrs and 24 hrs. This allows flexibility in analysis.
2. A design streamlined toward efficient retrieval of the user data necessary for feature computation. Note that, depending on the definition of the feature, aggregating activity records for a larger time window can require anything from simple counters to complex data structures.

To elaborate on this second guideline, we show 5 categories of behavioral features, and discuss appropriate structures for efficient data retrieval and aggregation for each category:

- Counts, averages, and standard deviations: these three metrics can be derived from simple counters. For example: *the number of successful logins over the last 24 hours*.
- Indicators (or *boolean variables*): Aggregating indicators is also straightforward and requires no additional data structures. For example: *whether at least one address verification failed over the last 24 hours*.

- Relational features: these features are calculated using data at the intersection of two entities. For example: *the maximum outlier score given to an IP address from which the user has accessed the website*. To compute these features efficiently, we build graphs that represent relations between entities in the system.
- Temporal behaviors: these variables capture the time elapsed between two or more events, and therefore must be analyzed in chronological order. For example: *the minimum time from login to checkout*). Computing these features requires timestamping all the relevant events (in this case, logins and checkouts), and comparing the time elapsed between consecutive events.
- Unique values: This kind of feature cannot be computed with counters, since duplicated values must be kept track of. We use a dictionary to maintain a set of unique values of the feature, and update it every time new user activity is analyzed. For example: *number of different locations from which a user has accessed the website over the last 24 hours*.

Activity aggregation: Computing behavioral features over an interval of time requires two steps:

1. Retrieve all activity records that fall within the given interval. Note that, for the purposes of this study, we consider behavioral descriptors that have been aggregated over 24 hours and end at the time of the last user activity. This can be graphically represented as a rolling 24-hour window for feature computation.
2. Aggregate minute-by-minute activity records as the feature demands. Again, this aggregation step depends on the feature type. In the simplest case, *counters*, one must merely add all the minute-by-minute values together. The more complex case of *unique values* requires retrieving the unique values of the super set formed by the minute-by-minute sets.

Performance considerations: Because the temporal scope of our activity records is 1 minute, we can aggregate records and compute features for flexible time intervals. However, this strategy can result in poor performance. For instance, in the worst case, to compute features over a 24-hour window, we need to retrieve and aggregate 24×60 (minute records)=1440 records. This process has to repeat for millions of entity instances.

To improve retrieval and aggregation performance, we maintain activity records with different, overlapping time scopes. In particular, we maintain records on:

- a minute-by-minute basis (starting on the dot),
- an hourly basis (starting on the dot),
- a daily basis (starting at midnight), and
- a weekly basis (starting Sunday at midnight).

This way, if we need to compute features for long intervals, our record retrieval and aggregation requirements remain bounded and satisfy real-time requirements. For example, with this strategy, the computation of features over the previous 24 hours requires the retrieval and aggregation of no more than 23(hour records)+60(minute records)=83 records.

6 Outlier detection methods

The use of outlier analysis is motivated by the observation that attacks are rare and exhibit distinctive behavior. We combine three unsupervised outlier detection techniques:

6.1 Matrix Decomposition-based outlier analysis

Key idea: Outlier detection methods based on matrix decomposition use Principal Component Analysis to find cases that violate the correlation structure of the main bulk of the data (Shyu *et al.* [2003]). To detect these rare cases, PCA-based methods analyze the projection from original variables to the principal components' space, followed by the inverse projection (or *reconstruction*) from principal components to the original variables (see Figure 5). If only the first principal components (the components that explain most of the variance in the data) are used for projection and reconstruction, we ensure that the reconstruction error will be low for the majority of the examples, while remaining high for outliers. This is because the first principal components explain the variance of normal cases, while last principal components explain outlier variance (Aggarwal [2013a]).

Let X be a p-dimensional dataset. Its covariance matrix Σ can be decomposed as: $\Sigma = P \times D \times P^T$, where P is an orthonormal matrix where the columns are the eigenvectors of Σ, and D is the diagonal matrix containing the corresponding eigenvalues $\lambda_1 \dots \lambda_p$. Graphically, an eigenvector can be seen as a line in 2D space, or a plane in higher-dimensionality spaces, while its corresponding eigenvalue indicates how much the data is stretched in that direction.

Note that, at this stage, it is common practice to sort the columns of the eigenvector matrix P and eigenvalue matrix D in order of decreasing eigenvalues. In other words, the eigenvectors and their corresponding eigenvalues are sorted in decreasing order of significance (the first eigenvector accounts for the most variance, the second for the second-most, etc.).

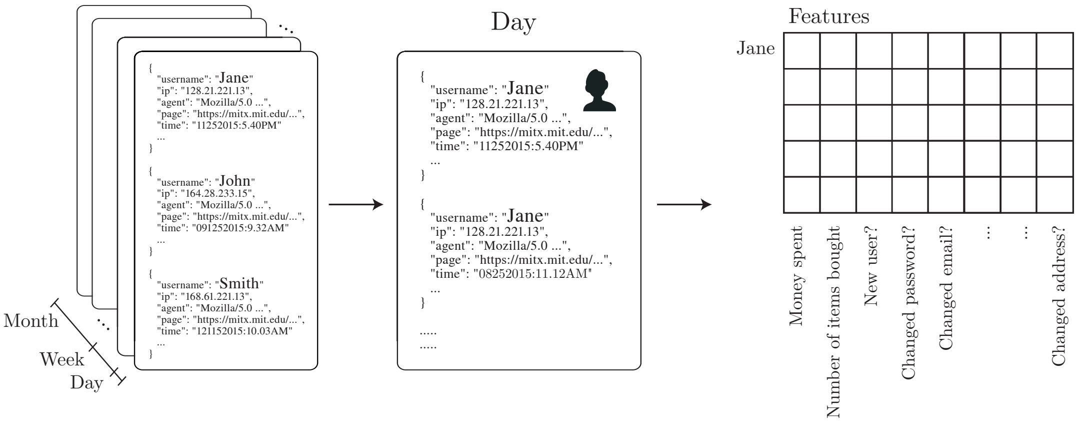
The projection of the dataset into the principal component space is given by $Y = XP$. Note that this projection can be performed with a reduced number of principal components. Let Y^j be the projected dataset using the top j principal components: $Y^j = X \times P^j$. In the same way, the reverse projection (from principal component space to original space) is given by $R^j = (P^j \times (Y^j)^T)^T$, where R^j is the reconstructed dataset using the top j principal components. This process is schematically depicted in Figure 5.

We define the outlier score of point $X_i = [x_{i1} \dots x_{ip}]$ as:

$$score(X_i) = \sum_{j=1}^{p} (|X_i - R_i^j|) \times ev(j) \qquad (1)$$

$$ev(j) = \frac{\sum\limits_{k=1}^{j} \lambda_k}{\sum\limits_{k=1}^{p} \lambda_k} \qquad (2)$$

Note that $ev(j)$ represents the percentage of variance explained with the top j principal components. As mentioned above, eigenvalues are sorted in decreasing order of significance; therefore, ev(j) will be monotonically increasing. This



```
{
  "username": "Jane"
  "ip": "128.21.221.13",
  "agent": "Mozilla/5.0 ...",
  "page": "https://mitx.mit.edu/...",
  "time": "11252015:5.40PM"
  ...
}

{
  "username": "John"
  "ip": "164.28.233.15",
  "agent": "Mozilla/5.0 ...",
  "page": "https://mitx.mit.edu/...",
  "time": "091252015:9.32AM"
  ...
}

{
  "username": "Smith"
  "ip": "168.61.221.13",
  "agent": "Mozilla/5.0 ...",
  "page": "https://mitx.mit.edu/...",
  "time": "121152015:10.03AM"
  ...
}
```

```
{
  "username": "Jane"
  "ip": "128.21.221.13",
  "agent": "Mozilla/5.0 ...",
  "page": "https://mitx.mit.edu/...",
  "time": "11252015:5.40PM"
  ...
}

{
  "username": "Jane"
  "ip": "128.21.221.13",
  "agent": "Mozilla/5.0 ...",
  "page": "https://mitx.mit.edu/...",
  "time": "08252015:11.12AM"
  ...
}

.....
.....
```

Figure 4: Extracting behavioral descriptors from big data. Our platform extracts information per entity from large raw log files. The result is a vector of indicators that describe the behavior of an entity over a predefined period of time.

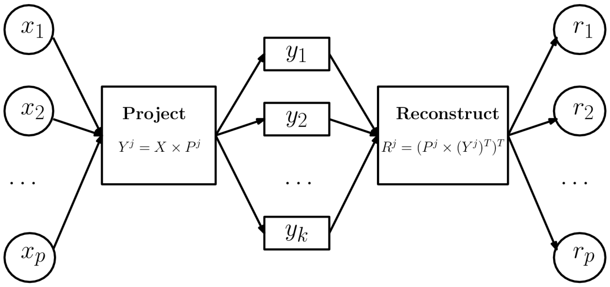


Figure 5: Matrix Decomposition outlier detection method: the original variables are projected into the top j principal components space. The dataset is reconstructed with the inverse projection, and the reconstruction error is used as the outlier score.

means that, the higher is j, the most variance will be accounted for within the components from 1 to j. With this outlier score definition, large deviations in the top principal components are not heavily weighted, while deviations in the last principal components are. This way, outliers present large deviations in the last principal components, and thus will receive high scores.

6.2 Replicator Neural Networks

Key idea: This method is similar to the previous one, in the sense that it also relies on a compression-reconstruction analysis. However, in this case, we train a multi-layer neural network to compress and reconstruct the data in such a way that the bulk of the data is reconstructed accurately, but outliers are not. This way, the reconstruction error can be directly translated into an outlier score.

Replicator Neural Networks (RNNs), or autoencoders, are multi-layer feed-forward neural networks. The input and output layers have the same number of nodes, while intermediate layers are composed of a reduced number of nodes. As depicted in Figure 6, we consider RNNs that are composed of three hidden layers. The first and third hidden layers count $p/2$ neurons, while the second, central layer is composed of $p/4$ neurons, where p is the dimensionality of the data. The tan-sigmoid transfer function is used as an activation function across the network.

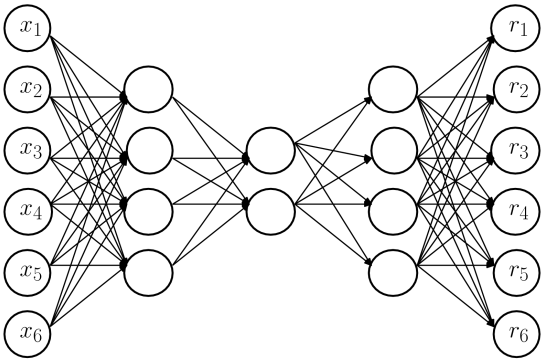


Figure 6: RNN composed of three intermediate layers. The original variables are compressed via non-linear transformation in the first layers and then decompressed to reconstruct the inputs. The reconstruction error is used as outlier score.

The network is trained to learn identity-mapping from inputs to outputs. The mapping from inputs to intermediate layers compresses the data. The data is then decompressed to reconstruct the inputs, mapping from intermediate layers to outputs. This reconstruction is lossy— that is, introduces an error, and the training process is aimed at minimizing it. The reconstruction error for the the i-th example is given by:

$$e_i = \sum_{j=1}^{p} (x_{ij} - r_{ij})^2 \tag{3}$$

where the input vector x and output vector r are both p-dimensional. Given a trained RNN, the reconstruction error is used as the outlier score: test instances incurring a high reconstruction error are considered outliers.

6.3 Density-based outlier analysis

Key idea: Next, we incorporate a technique that fits a multivariate model to the data. This results in a joint probability distribution that can be used to detect rare events, because test instances which fall within a low-density region of the distri-

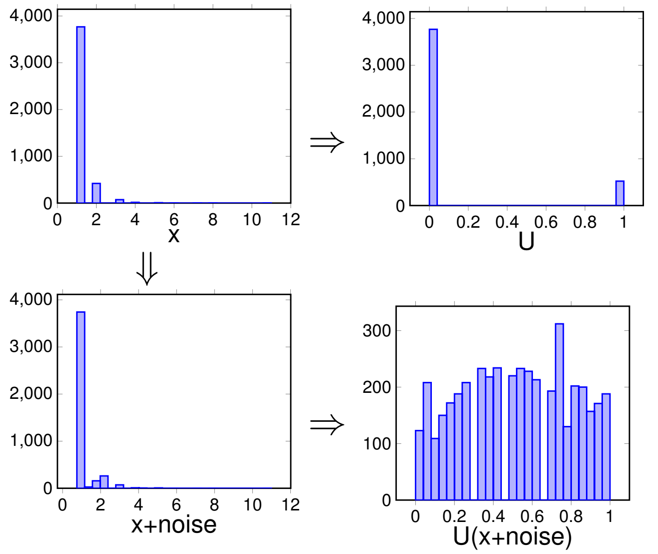


Figure 7: A discrete valued variable will not have a *uniform* density for its *cdf* values (top right). However, if tiny amount of additive white Gaussian noise is added to the variable (essentially making it continuous), the density of its *cdf* values look closer to *uniform* (bottom right). We use this trick to be able to model discrete variables using copulas.

bution are considered outliers. The outlier score is simply the probability density of a point in the multidimensional space.

To build a multivariate model from marginal distributions which are not all Gaussian, we exploit *copula* functions. A copula framework provides a means of inference after modeling a multivariate joint probability distribution from training data. Because copula frameworks are less well known than other forms of estimation, we will now briefly review copula theory. We will then describe how we construct the individual non-parametric distributions that make up a copula, and how we then couple them to form a multivariate density function.

A copula function $C(u_1, \ldots u_m; \theta)$ with parameter θ is a joint probability distribution of m continuous random variables, each of them uniformly distributed in $[0, 1]$. According to Sklar's theorem, any copula function that takes probability distributions $F_i(x_i)$ as its arguments defines a valid joint distribution with marginals $F_i(x_i)$. Thus, we are able to construct a joint distribution function for $x_1 \ldots x_m$ with arbitrary marginals as

$$F(x_1 \ldots x_m) = C(F_1(x_1) \ldots F_m(x_m); \theta). \qquad (4)$$

The joint probability density function (PDF) is obtained by taking the m^{th} order derivative of eqn. (4)

$$\begin{aligned} f(x_1 \ldots x_m) = \\ &\frac{\partial^m}{\partial x_1 \ldots \partial x_m} C(F_1(x_1) \ldots F_m(x_m); \theta) \\ &= \prod_{i=1}^{m} f_i(x_i) \cdot c(F_1(x_1) \ldots F_m(x_m); \theta) \qquad (5) \end{aligned}$$

where $c(.)$ is the *copula* density.

Gaussian copula: A multivariate Gaussian copula forms a statistical model for our variables given by

$$C_G(u_1 \ldots u_m; \Sigma) = F_G(\Phi^{-1}(u_1) \ldots \Phi^{-1}(u_m); \Sigma) \qquad (6)$$

where F_G is the CDF of multivariate normal with zero mean vector and Σ as covariance, and Φ^{-1} is the inverse of the standard normal.

Estimation of parameters: Let $\Psi = \{\Sigma, \psi_i\}_{i=1\ldots m}$ be the parameters of a joint probability distribution constructed with a copula and m marginals, being ψ_i the parameter of marginal i^{th}.

Given N *i.i.d* observations of the variables $\mathbf{x} = (x_{11}, \ldots, x_{mN})$, the log-likelihood function is:

$$L(\mathbf{x}; \Psi) = \sum_{l=1}^{N} \log \left\{ \left(\prod_{i=1}^{m} f(x_{il}; \psi_i) \right) \right. \\ \left. c(F(x_1) \ldots F(x_m); \Sigma) \right\} \qquad (7)$$

Parameters Ψ are estimated via maximum log-likelihood (Iyengar [2011]):

$$\hat{\Psi} = \arg\max_{\Psi} \sum_{l=1}^{N} \log \left\{ \left(\prod_{i=1}^{m} f(x_{il}; \psi_i) \right) \right. \\ \left. c(F(x_1) \ldots F(x_m); \Sigma) \right\} \qquad (8)$$

Estimation of $F_i(x_i)$: The first step in modeling copula density is to model the individual distributions for each of our features, x_i. We model each feature using a non-parametric kernel density-based method, described by:

$$f_\sigma(x_i^j) = \frac{1}{n\sigma} \sum_{j=1}^{n} K\left(\frac{x_i^j - \mu}{\sigma} \right) \qquad (9)$$

where $K(.)$ is a Gaussian kernel with the bandwidth parameter σ. Using this method together with our features, we encounter two problems.

First, most of the features produce extremely skewed distributions, making it hard to set the bandwidth for the Gaussian kernel. We set bandwidth parameter using *Scott's* rule of thumb.

Second, some of our variables are discrete ordinal. For copula functions to be useful, the probability density of $u_i = F(x_i)$ should be uniform, and for discrete-valued variables this condition is not met. In Figure 7, we demonstrate this using one of our features. The top left plot in the figure shows histogram for an original feature x_i. The histogram on the right is for u_i, which is the *cdf* values for the feature values. As we can see the histogram for u_i is not uniform.

To overcome this problem, we add additive white Gaussian noise to x_i. This simple transformation gives us a continuous-valued feature, given by x_i^c. In our formulation, we add noise to each feature value given by:

$$x_i^c = x_i + \eta(0, n_p) \qquad (10)$$

where n_p is a variance of the Gaussian distribution η used to add noise. This value is determined by evaluating $n_p = \frac{P_s}{SNR}$, where SNR is the desired signal-to-noise ratio. P_s is the signal power, estimated based on the distribution of all

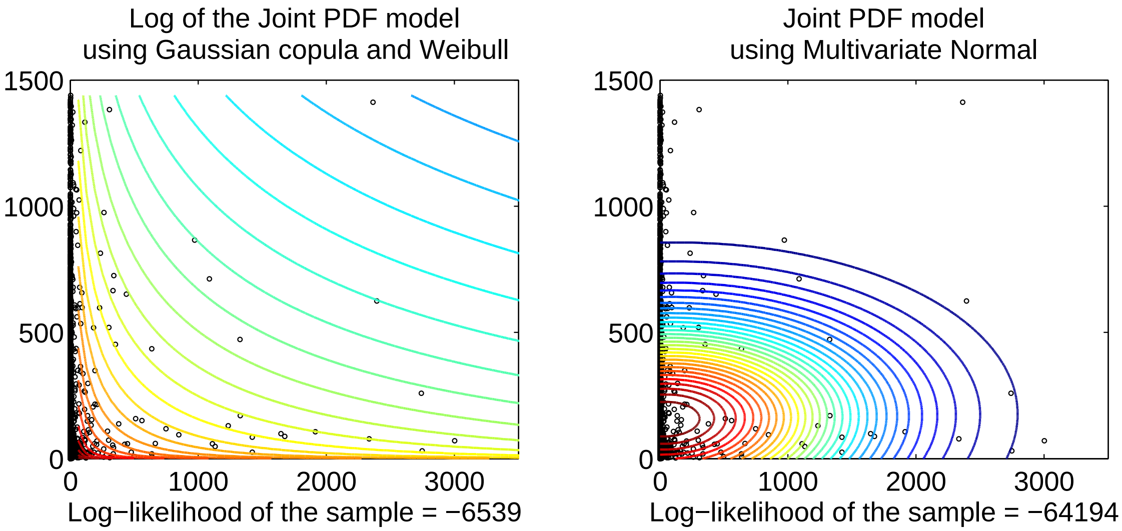


Figure 8: The left panel shows the contour lines of the log PDF of a joint Gaussian copula probability density model with Weibull marginals. The right panel shows the contour lines of a bi-variate normal fitted to the data. Qualitatively, it is clear that the copula model is better; as is evident from the contour lines spanning the entire data. Quantitatively, the log-likelihood of the multivariate normal model is one order of magnitude smaller at -64194 when compared to -6539 achieved by Copula based model.

values for the feature x_i [2]. For most of our features, the SNR value is set to 20. The bottom left plot in Figure 7 shows the histogram for the transformed variable x_i^c and the plot on the right shows the histogram for u_i^c. This looks closer to *uniform*.

Why Copulas?: In Figure 8 we demonstrate the efficacy of Copulas in modeling a bi-variate distribution. We took two features, plotted a scatter plot, modeled the features using a Gaussian copula with Weibull marginals and overlaid the contours for the density function. The plot on the left shows the result. On right we see the contours for a bi-variate Gaussian fitted to this data. We can see qualitatively that the joint Copula density function fits the data better. For quantitative comparisons, we evaluated the log-likelihood value to evaluate the fit. The Copula fits the data better by an order of magnitude.

6.4 Outlier score interpretation

The three outlier detection methods presented in the previous section assign a score that indicates each example's incongruity. Therefore, it is possible to rank all the test examples according to the score given by an individual detector. In the same way, one can select the top-k examples, or define a threshold to determine when the score is significant enough to consider the example as an outlier. Three main issues arise from such strategies:

1. Selecting the top-k examples can lead to false positives, because highly-ranked examples will not necessarily have a high outlier score. Since ranking is determined by comparing examples to each other and not by their absolute score, this scenario may occur if the data has few outliers.
2. Thresholding techniques are difficult to implement because scores are not easily interpretable. For instance, joint probability density values differ by more than 50 orders of magnitude (from 10^{-60} to 10^{-2}).
3. Because combining scores from different methods is not straightforward, it is also difficult to exploit the robustness of multi-algorithm outlier detection ensembles. Not only can the range of values returned by different methods be completely different, these methods can also result in opposite categorizations; in some cases, such as the matrix decomposition-based method, outliers receive the highest scores, while in other cases, such as probability density estimation methods, normal data points receive higher scores.

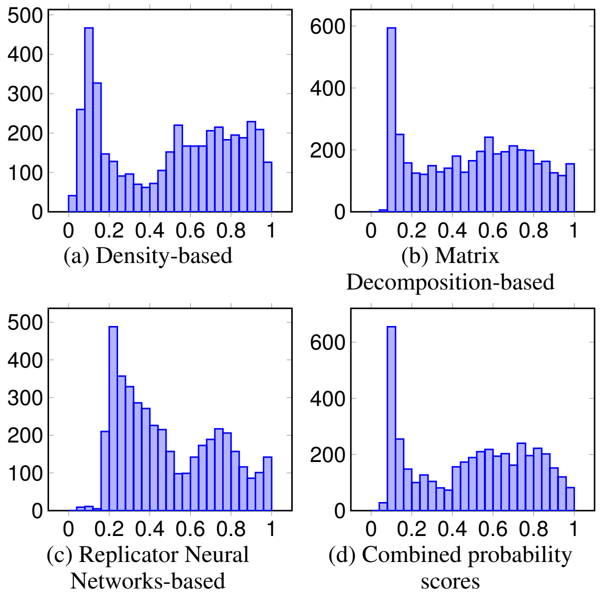
One solution for overcoming these limitations while still taking advantage of two or more detectors is to combine ranks instead of scores Zimek *et al.* [2014]. However, highly ranked examples will still be considered outliers, regardless of whether their absolute outlier scores are high or not. As a result, this strategy can result in a high false positive rate.

Another solution is to project all scores into the same space, ideally interpretable as probabilities. We adopt this last strategy; however, as we explain in the following subsection, it comes with its own challenges.

6.5 Transforming outlier scores into probabilities

We model matrix decomposition-based outlier scores with a Weibull distribution, which is flexible, and can model a wide variety of shapes. For a given score S, the outlier probability corresponds to the cumulative density function evaluated in S: $F(S) = P(X \leq S)$. The exact same technique can be applied to the replicator neural networks scores.

[2] In future we estimate the signal power value for each individual value of the feature x_i separately allowing a more customized noise value



(a) Density-based

(b) Matrix Decomposition-based

(c) Replicator Neural Networks-based

(d) Combined probability scores

Figure 9: For one-day's worth of data, these plots show the histograms for the outlier scores from the three methods and the histogram of the combined score. These scores for each method are after the series of transformations performed on their raw score.

Joint probability densities require an additional step, because the scores span different orders of magnitude. As a result, most of the information is lost. To mitigate this loss, we first compute the negative logarithm of the scores, and then shift the distribution to have positive support. Once this transformation is performed, we model the transformed scores with a Weibull distribution and determine the outlier probability for each example.

6.6 Outlier detection ensembles

Multi-algorithm ensembles are combinations of predictions from different machine learning models. This strategy improves robustness by compensating for the individual biases of models in the ensemble. In this case, we average outlier probabilities obtained separately by each of the methods. Each example must be highly scored by all methods in order to be highly ranked and shown to the end user.

7 Active Model Synthesis

This system is meant to continually identify new and evolving attacks with the help of an analyst, and to use these identifications to synthesize new models that can predict attacks *without* the analyst, using behavioral descriptors. For this, we designed a closed-loop system that entwines analyst intuition with machine intelligence.

We present an outline of the Active Model Synthesisframework in Figure 10. The algorithm has three phases–TRAINING, DEPLOYMENT and FEEDBACK COLLECTION/UPDATING–and cycles through these phases daily. The entity-feature matrix and the labeled data serve as the algorithm's inputs. In an everyday workflow, the system trains unsupervised and supervised models, applies these models to that day's incoming data, identifies k entities as extreme events or attacks, and brings them and their data to the analysts' attention. The analysts then use an interface to sort through these rare events and pick out which could truly be attacks. Finally, we use the analysts' deductions to build a new predictive model for the next day.

The key advantages of this system are:

- *Overcomes limited analyst bandwidth*: The number of events an analyst can feasibly examine is a tiny fraction of the overall event volume, about 10^{-5} %. To select these events, we rely on the accurate, robust and multi-method outlier detection system presented in Section 6. We update our models daily and use them the next day, as presented in Figure 10.
- *Overcomes weaknesses of unsupervised learning*: One of the key intuitions driving our system is that an event's rarity (or its status as an outlier) does not constitute *maliciousness*, and that an event's score does not capture the intent behind it. If we consider all top k events as malicious, then a simple threshold-based detector would be enough to diagnose them. A non-linear model enables us to imitate analysts' subjective assessment of the events.
- *Actively adapts and synthesizes new models*: Analyst feedback delivers labeled data on a daily basis. This increases the training data available to the system, allowing us to change models on a daily basis. This setup captures the cascading effect of the human-machine interaction: the more attacks the predictive system detects, the more feedback it will receive from the analysts; this feedback, in turn, will improve the accuracy of future predictions. Therefore, as time progresses and the systems absorb the analysts' feedback, we expect to see clear improvement in the detection rate. In addition, we allow the analysts to sort the attacks into multiple categories, enabling us to build custom models for different attacks.

8 Experimental setup

To validate our platform, we experimented with a real-world data set, with reported attacks introduced in Section 4. The experiments performed were designed to show how the analyst's feedback improved the threat-detection process.

8.1 Types of attacks

To illustrate the variety of threats that may compromise enterprise platforms, we describe the behaviors involved in three representative attacks.

- *Account takeover attacks*: Account takeover attacks generally consist of two steps. First, attackers will try to access a given website using many user/password pairs from a reduced number of IP addresses. At this stage, the bad actors will figure out which user credentials are active, but will perform few or no checkouts. Note that this step can be detected by looking at elevated numbers of login attempts originating from the same IP; however, strictly speaking, no fraud has yet been committed.

```
APPLY MODELS: Given the entity-feature matrix $\mathbf{M}_t$ at time $t$, deploy and execute the models $\mathbf{U}_{t-1}$ and $\mathbf{S}_{t-1}$:
  STEP 1A: Apply the unsupervised model $\mathbf{U}_{t-1}$ to $\mathbf{M}_t$ and generate scores given by $\mathbf{P}$.
  STEP 1B: Apply the supervised model $\mathbf{S}_t$ to $\mathbf{M}_t$ and generate scores given by $\mathbf{Z}$.

SELECT $k$ ENTITIES: Given analyst bandwidth $k$, scores from unsupervised model, $\mathbf{P}$, and the supervised model $\mathbf{Z}$:
  STEP 2A: Select top $\frac{k}{2}$ entities based on the score $\mathbf{P}$.
  STEP 2B: Select top $\frac{k}{2}$ entities based on the score $\mathbf{Z}$.

COLLECT FEEDBACK AND UPDATE: Show the $k$ entities to the analyst and collect feedback:
  STEP 3: For each of the $k$ entities collect analyst defined labels and add $D_k$ it to the labeled training data $D_t = D_k \cup D_{t-1}$
TRAINING: Given the entity-feature matrix $\mathbf{M}_t$ at time $t$, and labeled data $\mathbf{D}_t$:
  STEP 4A: Train an unsupervised model $\mathbf{U}_t$ using $\mathbf{M}_t$. The unsupervised training includes multiple modeling techniques
      and an ensembling method described in Section 6.
  STEP 4B: If labeled data, $\mathbf{D}_t$, is available, train a supervised model $\mathbf{S}_t$ using a random forest classifier.
```

Figure 10: ACTIVE MODEL SYNTHESIS algorithm

At a later time, the same or a different bad actor will access the site with stolen ”validated” credentials, and perform transactions using the credit cards associated with these accounts. In this case, to avoid raising suspicions, attackers will generally use a single user per IP address.

- *New account fraud*: In a new account fraud, a bad actor gains access to a stolen credit card and creates a new account using the credit card owner’s personal information. Once the account is created, the bad actor performs transactions with the stolen credit card.
- *Terms of service abuse*: This category covers fraudulent violations of the terms of service agreement. Frauds of this sort have very distinct signatures. Two simple examples are the abusive use of promotional codes, or deleting the web browser’s cookies to participate more times than allowed in an online voting platform.

8.2 Analyzing the impact of the analysts’ feedback

We compare the fraud detection rate obtained with a purely unsupervised outlier analysis approach to the Active Model Synthesis strategy explained in Section 7.

In some scenarios, we may have access to labeled data from the past, even before the rare event detection system is deployed. We call these labels *historic* labels. We introduce an additional parameter, $d \in \{0, 28\}$ to represent the number of days for which we have (albeit incomplete or noisy) labeled examples. For each strategy, we report the total number of detected attacks on a monthly basis, the recall, and the area under the receiver operating characteristic curve (AUC) of the deployed classifier.

Figure 11 shows the detection rates achieved with user-based features, where the analyst has a fixed daily bandwidth of $k = 100$ incident investigations. The following observations are worth noting:

- The Active Model Synthesis setups beat fully unsupervised outlier detection by a large margin. Over the 12 weeks of the simulation, the outlier detection approach caught a total of 42 attacks, while the Active Model Synthesis setups with d=0 and d=28 detected 143 and 211 attacks respectively, out of a total of 318 attacks successfully linked to individual users.
- The detection rate of the Active Model Synthesis setups with d=0 and d=28 increases over time, reaching 0.500 and 0.604 respectively at the 12^{th} and final week of the simulation.
- The performance of the classifiers at the end of the 12^{th} week is almost identical between the three Active Model Synthesis setups. In the case of d=0, the AUC of the classifier in the final week reaches 0.940, while the setup considering d=28 reaches 0.946.

Based on these results, we present the following key findings:

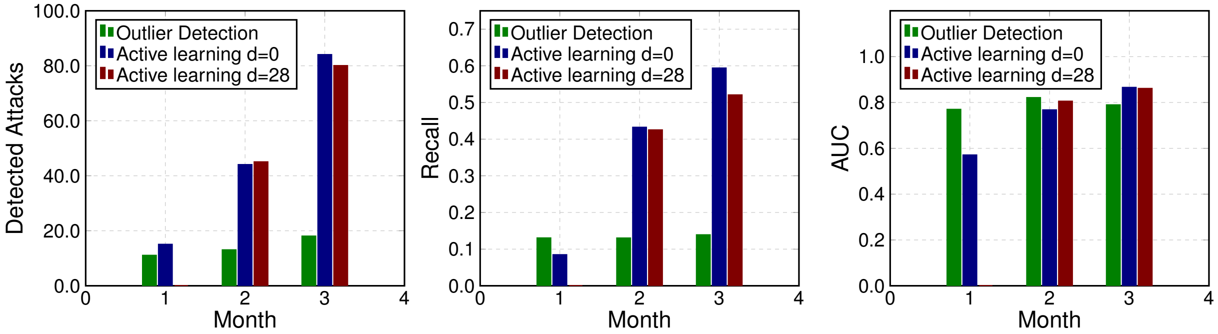
- Over the course of three months, our system with d=0, with no initial labeled examples, increased the attack detection rate by $3.41\times$, compared to state-of-the-art unsupervised outlier detection.
- Our platform reduces the number of false positives with respect to state-of-the-art unsupervised outlier analysis. As shown in Figure 1, once the system is trained, we achieve a recall of 0.868 with k (analyst bandwidth) set to 200, whereas the unsupervised-ML approach achieves 0.737, even when the analyst is shown 1000 entities a day. This observation indicates a simultaneous increase of the attack detection rate and a fivefold false positive reduction; therefore, our system improves the analyst’s efficiency and mitigates alarm fatigue issues.
- The system learns to defend against unseen attacks and can be bootstrapped without labeled data. Given enough interactions with the analyst, the system reaches a performance similar to that obtained when historic attack examples are available.

9 Conclusion

We present an end-to-end system that combines analyst intelligence with state-of-the-art machine learning techniques to detect new attacks and reduce the time elapsed between attack detection and successful prevention. The system presents four key features: a big data behavioral analytics platform,

	k=50				k=100				k=200				k=500				k=750				k=1000			
	OD		AL		OD		AL		OD		AL		OD		AL		OD		AL		OD		AL	
week	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr	FPr	TPr
1	0.013	0.133	0.013	0.133	0.026	0.133	0.026	0.133	0.051	0.133	0.051	0.133	0.129	0.333	0.129	0.333	0.193	0.400	0.193	0.400	0.257	0.400	0.257	0.467
2	0.013	0.174	0.013	0.130	0.027	0.217	0.027	0.304	0.054	0.391	0.054	0.348	0.135	0.652	0.135	0.783	0.203	0.826	0.203	0.870	0.271	0.913	0.271	0.913
3	0.020	0.037	0.020	0.000	0.039	0.074	0.039	0.111	0.079	0.259	0.079	0.222	0.197	0.630	0.197	0.630	0.296	0.630	0.296	0.852	0.395	0.815	0.394	0.963
4	0.013	0.000	0.013	0.048	0.026	0.095	0.026	0.143	0.053	0.190	0.053	0.429	0.133	0.286	0.132	0.667	0.199	0.524	0.199	0.714	0.265	0.714	0.265	0.857
5	0.012	0.048	0.012	0.048	0.024	0.048	0.024	0.286	0.048	0.048	0.047	0.476	0.119	0.429	0.119	0.714	0.179	0.524	0.179	0.810	0.238	0.619	0.238	0.857
6	0.013	0.042	0.013	0.000	0.026	0.083	0.026	0.292	0.052	0.167	0.052	0.458	0.130	0.250	0.130	0.625	0.195	0.375	0.195	0.667	0.260	0.458	0.260	0.625
7	0.012	0.160	0.012	0.120	0.025	0.240	0.024	0.520	0.050	0.320	0.049	0.640	0.124	0.560	0.124	0.720	0.186	0.640	0.186	0.800	0.249	0.800	0.249	0.880
8	0.013	0.074	0.013	0.296	0.027	0.148	0.026	0.667	0.053	0.333	0.053	0.815	0.134	0.444	0.133	1.000	0.200	0.741	0.200	1.000	0.267	0.815	0.267	0.963
9	0.013	0.226	0.013	0.581	0.026	0.290	0.026	0.677	0.053	0.387	0.052	0.806	0.132	0.677	0.132	0.871	0.198	0.742	0.198	0.871	0.265	0.742	0.265	0.871
10	0.013	0.100	0.012	0.300	0.025	0.100	0.025	0.633	0.051	0.200	0.050	0.767	0.127	0.500	0.126	0.867	0.190	0.733	0.190	0.933	0.253	0.767	0.253	0.933
11	0.014	0.083	0.014	0.361	0.028	0.111	0.028	0.556	0.056	0.167	0.056	0.694	0.141	0.306	0.141	0.722	0.212	0.444	0.212	0.806	0.283	0.639	0.282	0.806
12	0.011	0.026	0.011	0.342	0.023	0.053	0.022	0.632	0.045	0.079	0.044	0.868	0.113	0.289	0.112	0.895	0.169	0.395	0.168	0.921	0.225	0.737	0.225	0.947

Table 2: False positive rate (FPr) and true positive rate (TPr) of the Active Model Synthesisand unsupervised outlier detection over the 12 weeks of deployment. We report the FPr and TPr of the compared approaches for different daily investigation budgets ($k \in 50, 100, 200, 500, 1000$)



(a) Detected Attacks (b) Recall (c) AUC

Figure 11: User-based analysis: we report the number of detected attacks, recall rate, and AUC of the three compared approaches: outlier detection, Active Model Synthesiswith $d = 0$, and Active Model Synthesiswith $d = 28$

an ensemble of outlier detection methods, a mechanism for obtaining feedback from security analysts, and a supervised learning module. We validate our platform with a real-world data set consisting of 3.6 billion log lines. The results show that the system learns to defend against unseen attacks: as time progresses and feedback is collected, the detection rate shows an increasing trend, improving by $3.41\times$ with respect to a state-of-the-art unsupervised anomaly detector, and reducing false positives by more than $5\times$.

References

Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 504–509, 2006.

Charu C. Aggarwal. *Outlier Analysis*. Springer, 2013.

Charu C. Aggarwal. Outlier ensembles: Position paper. *SIGKDD Explor. Newsl.*, 14(2):49–58, April 2013.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, July 2009.

Jing Gao and Pang-Ning Tan. Converting output scores from outlier detection algorithms into probability estimates. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 212–221, Washington, DC, USA, 2006. IEEE Computer Society.

Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier detection using replicator neural networks. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, *Data Warehousing and Knowledge Discovery*, volume 2454 of *Lecture Notes in Computer Science*, pages 170–180. Springer Berlin Heidelberg, 2002.

Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, October 2004.

S.G. Iyengar. Decision-making with heterogeneous sensors-a copula based approach. *PhD Dissertation*, 2011.

Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Interpreting and unifying outlier scores. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pages 13–24, 2011.

David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the eleventh international conference on machine learning*, pages 148–156, 1994.

Barbora Micenková, Brian McWilliams, and Ira Assent. Learning outlier ensembles: The best of both worlds–supervised and unsupervised. In *KDD'14 Workshops: Outlier Detection and Description (ODD^2)*, 2014.

Dan Pelleg and Andrew W. Moore. Active learning for anomaly and rare-category detection. In *Advances in Neural Information Processing Systems 17*, pages 1073–1080, 2004.

Matthias Scholz and Ricardo Vigário. Nonlinear PCA: a new hierarchical approach. In *Proceedings of the 10th European Symposium on Artificial Neural Networks (ESANN)*, pages 439–444, 2002.

Matthias Scholz, Martin Fraunholz, and Joachim Selbig. Nonlinear principal component analysis: Neural network models and applications. In AlexanderN. Gorban, Balázs Kégl, DonaldC. Wunsch, and AndreiY. Zinovyev, editors, *Principal Manifolds for Data Visualization and Dimension Reduction*, volume 58 of *Lecture Notes in Computational Science and Enginee*, pages 44–67. Springer Berlin Heidelberg, 2008.

Erich Schubert, Remigius Wojdanowski, Arthur Zimek, and Hans-Peter Kriegel. On evaluation of outlier rankings and outlier scores. In *Proceedings of the Twelfth SIAM International Conference on Data Mining, Anaheim, California, USA, April 26-28, 2012.*, pages 1047–1058, 2012.

Ted E. Senator, Henry G. Goldberg, Alex Memory, William T. Young, Brad Rees, Robert Pierce, Daniel Huang, Matthew Reardon, David A. Bader, Edmond Chow, Irfan Essa, Joshua Jones, Vinay Bettadapura, Duen Horng Chau, Oded Green, Oguz Kaya, Anita Zakrzewska, Erica Briscoe, Rudolph IV L. Mappus, Robert McColl, Lora Weiss, Thomas G. Dietterich, Alan Fern, Weng-Keen Wong, Shubhomoy Das, Andrew Emmott, Jed Irvine, Jay-Yoon Lee, Danai Koutra, Christos Faloutsos, Daniel Corkill, Lisa Friedland, Amanda Gentzel, and David Jensen. Detecting insider threats in a real corporate database of computer usage activity. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1393–1401, New York, NY, USA, 2013. ACM.

H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, New York, NY, USA, 1992. ACM.

Mei-ling Shyu, Shu ching Chen, Kanoksri Sarinnapakorn, and Liwu Chang. A novel anomaly detection scheme based on principal component classifier. In *in Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop, in conjunction with the Third IEEE International Conference on Data Mining (ICDM'03*, pages 172–179, 2003.

Ting-Fang Yen. *Detecting stealthy malware using behavioral features in network traffic*. PhD thesis, Carnegie Mellon University, 2011.

Arthur Zimek, Ricardo J.G.B. Campello, and Jörg Sander. Ensembles for unsupervised outlier detection: Challenges and research questions a position paper. *SIGKDD Explor. Newsl.*, 15(1):11–22, March 2014.