# Detecting Insider Threats using RADISH, a System for Real-time Anomaly Detection in Heterogeneous Data Streams

Brock Böse*, Bhargav Avasarala*, Srikanta Tirthapura†, Yung-Yu Chung†, and Donald Steiner*,
*Northrop Grumman Corporation †Iowa State University, Dept. of Elec. and Computer Engg.

***Abstract*—We present a scalable system for high-throughput, real-time analysis of heterogeneous data streams. Our architecture enables incremental development of models for predictive analytics and anomaly detection as data arrives into the system. In contrast with batch data-processing systems, such as Hadoop, that can have high latency, our architecture allows for ingest and analysis of data on the fly, thereby detecting and responding to anomalous behavior in near real-time. This timeliness is important for applications such as insider threat, financial fraud, and network intrusions. We demonstrate an application of this system to the problem of detecting insider threats. Namely, the misuse of an organization's resources by users of the system, and present results of our experiments on a publicly available insider threat dataset.**

***Index Terms*—Anomaly Detection, Insider Threat, Streaming Analytics, Real-time Analytics**

## I. Introduction

AT THE dawn of the information age, organizations focused a majority of their resources protecting their assets from compromise by outside forces. A recent rash of high profile incidents has brought to light a fundamental truth that security experts have known since at least 44 BC, i.e. the greatest threat to an organization does not come from without, but from within. Insiders, operating within the domain of their normal activities and with the freedom necessary to efficiently accomplish their tasks, can bypass the elaborate defenses against external threats to steal critical secrets and/or damage critical resources. In the digital age the threat posed by the insider, be they malicious or merely unwitting, has increased manyfold.

A recent study by the Center for Strategic and International Studies (CSIS) [1] estimated the total cost of cyber espionage worldwide to be between $150 - $300 billion per year. The loss of intellectual property is an obvious deficit, but the full list of costs, both direct and indirect, illustrates the enormity of the risk posed by these threats. A 2012 report by the Zurich Insurance Group [2] identified six sources of direct loss resulting from an internal data breach, including *menu costs* associated with reconfiguring security features after a data breach, such as reissuing credit cards and changing user accounts, *forensic investigation costs* that can range from $200 to $1500 per hour, customer relations costs, credit monitoring costs, legal reparation costs, and costs associated with additional regulatory processes, such as investigations launched by federal or state authorities.

In addition to direct costs, there are indirect costs associated with an insider incident that, while harder to quantify, can be potentially more damaging. These include *reputation damage* that can result in loss of clients and a reduction in business from retained clients. As an example, the recent light shown on NSA activity by Snowden has caused significant costs for US IT providers, as their clients requested that data centers be moved outside the US as well as significant loss of business among foreign clients who no longer believe their data is safe [3]. Another indirect, yet real, cost results from *additional competition* due to compromised technology and/or business plans. As noted in the recent CSIS study [1], "the victim may not know the reason they were underbid, a negotiation went badly, or a contract was lost". A notion of the harm that is done can be gleaned from a recent theft of proprietary data from the oil and gas industry. In this case, the theft targeted "project-financing information with regard to oil and gas field bids and operations" [4].

The need to mitigate risk associated with the above costs has spurred the development of numerous tools that aim to prevent data exfiltration, identify and monitor high risk individuals within an organization, and so on. A review of the current work in this field, which we summarize in Section II, reveals that these efforts fall short in their protection, the flexibility and maintainability of their preventative measures, and/or the timeliness of their response.

Our approach to detecting malicious insider actions in an enterprise is based on automatically identifying unusual behavior within the stream of actions associated with different users within the enterprise's computational network. The distinguishing feature of our approach is the use of **streaming analytics** in learning typical patterns of behavior occurring in event streams and in monitoring the streams for deviations from these patterns. To enable streaming anomaly detection at scale, we built RADISH (Real-time Anomaly Detection In Streaming Heterogeneity), a system for rapidly detecting patterns and anomalies in streaming data.

RADISH ingests and analyzes heterogeneous streams of data in real-time to detect patterns that span different streams. In order to scale to large volume streams and provide high-throughput processing, RADISH integrates open source distributed processing frameworks that enable it to use the power of parallel computing on a cluster of computers. A key

NGIS-DSEA-14-01335
Approved for Public Release, #15-2206; Unlimited Distribution

technical ingredient in RADISH is the use of novel streaming machine learning and data mining methods that enable scalable real-time analytics. These methods build behavioral models of individual entities as well as aggregated behavior.

In this paper, we describe the overall architecture of RADISH, a general streaming anomaly detection system. We show how the RADISH system is particularly applicable to insider threat detection, and present initial results showing the effectiveness in detecting malicious insider actions within a synthetic dataset generated for the DARPA ADAMS program [5], [6].

The rest of the paper is organized as follows: in Section II, we present a brief review of current technologies for massive on-line data analytics and the state of the field of insider threat reduction. In Section III, we describe the RADISH design followed by details of the current implementation in Section IV. In Section V, we describe a system based on RADISH for detecting insider threats, followed by our evaluation on an independently developed synthetic dataset. We conclude with a discussion of the results of the experiments and future work in Section VI.

## II. Related Work

The RADISH system advances the state-of-the-art in streaming analytics and provides a novel approach to insider threat detection. In this section, we survey related work.

**Insider Threat Detection Systems.** Currently, most insider threat mitigation technologies, such as Wave [7], focus specifically on *data loss prevention*. These technologies use data encryption, automated remote backup, and document tagging to ensure that a malicious insider cannot deface, delete, or exfiltrate sensitive organizational information. This type of approach fails to address two key features of the malicious insider problem. First, the insider is usually operating within the constraints of their normal function, thus, access to and exfiltration of the sensitive data is not precluded. Second, document tagging is, in any event, difficult to maintain and easy to defeat. It's difficult to maintain since each proprietary document and all descendant documents to be protected must be manually identified as sensitive and tagged correspondingly. This process is easy to defeat, since most tagging mechanisms are based on hashing the document to a key and storing that key in a blacklist against which all outgoing documents are checked. This hash can often be circumvented by simply adding a single character to the document.

Technologies such as Raytheon's SureView [8] take a more holistic approach to mitigating insider threats by compiling and analyzing multiple sources of information on user behavior, but again suffer from drawbacks associated with gathering, storing, and analyzing massive amounts of data. SureView allows exhaustive auditing of privileged users, including a DVR-like playback of the user's actions. Recording at this level of fidelity requires a significant amount of storage. Thus, for a sizable organization, it is resource-prohibitive to store all the data from all possible users. Thus, the usual mode of operation is to identify those users who pose potential threats before they are monitored. Given that detection of the average insider incident occurs 32 months after the act [9], predetermination is not a luxury organizations currently possess. Other technologies, such as Palisade [10], Prelert [11], and Securonix [12], provide an adaptive approach to identifying malicious behavior by characterizing network and user behavior based on analyzing logs after the logs have been collected in a central repository. Thus, they can provide forensic evidence of how an attack occurred, but cannot stop an incident that takes only minutes to carry out. To prevent a determined insider, the security system must operate in real time[1].

In summary, the RADISH approach differs from existing commercial anomaly detection approaches in three critical respects. First, all analyses are performed on the stream of data as it is generated in the system, not on static data sets that are gathered after the fact. This is advantagous in that it eliminates the requirement to store prohibitively large amounts of data for later analysis, thereby allowing for the analysis of much larger collections of disparate data and enabling real-time alerts. Second, our analysis considers streams from many disparate sources rather than a single type of data. These sources, such as e-mails, browser history, and security logs, yield a greater depth to our system's characterization of employees, assets, and resources reducing the false positive rate while simultaneously increasing the detection rate. Finally, at the lowest level, our characterization of normal behavior is performed automatically and algorithmically. This has the effect of reducing expense and improving accuracy by eliminating costly and error-prone hand-coded rules for user behavior.

The insider threat problem has also been addressed in the academic literature. [13] uses the synthetic ADAMS dataset to build and evaluate a multi-domain anomaly detector for identifying suspicious users. Unlike the approach reported here, it does so by aggregating features per domain (e.g. HTTP, email, logon, etc) across all time, and computing anomaly scores for each user. Furthermore, each domain is first considered and modeled in isolation, and individual anomaly scores fused in order to compute a single score. However, if any set of individual models produce noisy anomaly scores, they may pollute the final score and decrease the accuracy and reliability of the system. In addition, using aggregate features will tend to have a smoothing effect and has the potential to mask anomalous behavior which occurs in bursts over a small time scale. We note that [13] does not compare the anomalous users identified by their system with the malicious users identified in the dataset. In contrast to the work described in [13], we focus on individual user sessions and compute an anomaly score for each, considering all domains at once; that is, the features we use per session contain information from all domains, and as a result, implicitly take correlations between cross-domain events into account. We note that our findings on this dataset did align with those reported in [13], and we show that the sessions we identify as anomalous correspond well with malicious activity.

**Distributed Systems and Streaming.** According to IBM,

[1]At the time of publication, vendors are starting to release systems based on streaming input, however, they still consume significant processing and storage resources and do not perform real-time analysis on all available data streams.

as of 2012, 2.5 exabytes ($2.5 \times 10^{18}$ bytes) of data are generated every day [14]. Internet-based companies (i.e. Google, Facebook, etc) generate enormous volumes of log data, corresponding to user activity-related events, such as log-ins, clicks, and search queries [15]. Mining this kind of data can provide valuable insights into user patterns and preferences, forming the basis for advertising and recommender systems. Early architectures designed to ingest and process this data involved first aggregating it into a single data store (e.g. HDFS) for offline consumption. While this paradigm is indeed effective for offline batch-processing of large volumes of data, it cannot be trivially modified to deal with streaming applications where decisions must be made quickly in response to incoming data. Hybrid approaches may use a large amount of data collected for a pre-specified period of time in order to train models offline that are then applied in real-time. These approaches may be viable in a few cases, such as when a recommender system is attempting to learn user preferences or when an intrusion detection system monitors network activity to establish patterns of normalcy. Such an architecture would still rely on storing the data, which may not be desirable due to space constraints. Streaming applications, which ingest and analyze the data as it arrives and dynamically update models, clearly cannot leverage such an architecture. Implied in the online requirement for streaming applications is the notion that data will likely be discarded after being seen and, any analytics will encounter the data only once. Real time applications, such as discovering trending topics in social media, determining current user sentiment, or discovering abnormal activity in a computer network are just a few examples of areas that require distributed stream processing architectures. Apache Storm [16] and Apache Spark Streaming [17] are technologies that provide distributed stream processing frameworks. When combined with popular messaging systems, such as Kafka [15], ActiveMQ [18], and ZeroMQ [19], the can achieve very high throughput and low latencies for even complex stream processing.

## III. System Design

RADISH identifies suspicious activity by simultaneously analyzing incoming data streams to learn patterns of normal behavior and, in the context of this learned behavior, search for anomalous activity that portends abnormal system behavior, such as a data breach or attack from within. RADISH is composed of two distinct processes; a learning process (RADISH-L) and an alerting process (RADISH-A) that run simultaneously and continuously.

RADISH-L analyzes the streams to automatically derive models representing patterns of normalcy that characterize the normal operation of a system. (The system may be an enterprise's computer network including computers, their applications, and communications between them, or might be a complex machine composed of interacting parts with sensors indicating the status of the parts.) In the case of insider threat detection, these patterns cover rules about the behavior of users as well as the usage behavior of resources such as workstations, servers, and/or files. RADISH-A matches incoming event streams against the patterns of normalcy derived by RADISH-L to detect anomalous system behavior.

The system architecture is depicted in Figure 1. The sensors originate events, where, in this paper, we define an event to be a tuple of attributes with a sensor observation timestamp. An array of sensors is typically deployed within an organization's infrastructure at a variety of places, such as user workstations, routers, firewalls, and servers. The Harmonizer collects the events, transforms them into a common format, performs enrichment and identity resolution, and time-orders the events within a stream. This enriched stream is published to both RADISH-A, which detects anomalies using a current set of models, and to RADISH-L, which uses the events to hone the next generation of models. The models are managed in a repository that allows RADISH-L to load, update, and save models that are retrieved by RADISH-A for detection. Finally, the entire system is monitored and controlled via the controller. We now present a more detailed view of each component.

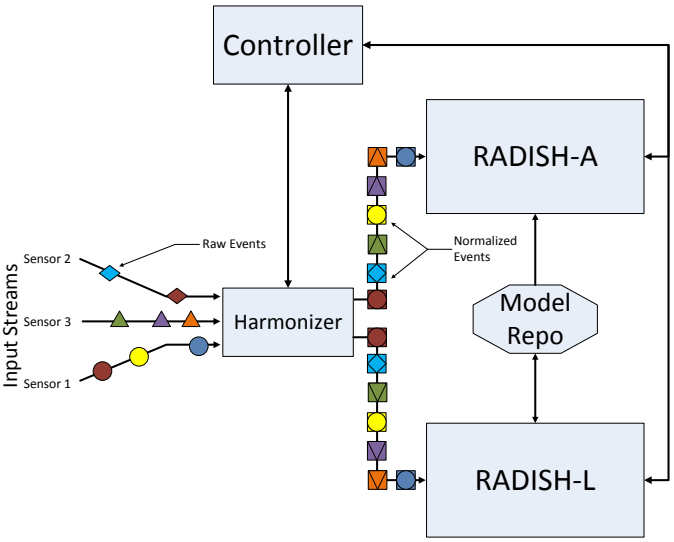



Fig. 1: The RADISH System Architecture Normalized streams are fed to both RADISH-L that generates models of typical user behavior and to RADISH-A that uses these models to detect anomalous behavior.

### A. Sensors

Sensors are devices and applications capable of relaying information on user, asset, or resource utilization to the Harmonizer. In principle, any information source that can communicate over a network can be included as an information stream into RADISH. Examples of sensors currently available include various log files such as HTTP logs, email logs, system call logs, Centrify command data [20], Guardium database access data [21], and Windows system event logs [22].

### B. Harmonizer

The Harmonizer, Figure 2, normalizes events generated by different sensor streams into a common data format in addition to time-ordering them. To that end, the Harmonizer

performs the functions of transforming data, performing identity resolution for objects in the stream, and providing any additional enrichment needed. For a large production system covering disparate locations in an organization, numerous Harmonizers could operate in parallel, each handling events from an appropriate sub-set of entities being monitored by the system.

When a new stream is introduced into RADISH, a plug-in must be implemented that provides the custom processing required to convert the sensor's events into the Harmonizer's common data forat. The Harmonizer provides services to the plug-in to aid in this process. The first is an identity resolution service that resolves any primary identifiers associated with the event (such as Unix login names, e-mail addresses, and mac addresses) into a common identifier (such as employee IDs and inventory IDs) that can be easily recognized by downstream components. The second service is enrichment that attaches metadata to the event that may be inconvenient or impossible to add at the sensor. For instance, data from a building access control system might be enriched with the IDs of devices that are associated with the people entering the building (such as their cell phone or work station IDs). This data, in general, is not available to access control systems, but is helpful in establishing correlations if anomalous behavior is associated with users' devices. The third service is a joining of streams, where multiple streams can be joined together according to a join criterion, resulting in one or more derived streams. For instance, streams corresponding to different activities by the same user, such as web browsing history, file access, and emails can be combined into a single stream containing all activities of the user. Of the tasks required of the Harmonizer, the first three (transforming events to a common format, event enrichment, and identity resolution) require no information exchange between the events and can be performed in an embarassingly parallel fashion. Time ordering of the events proves more problematic, but can be resolved by partitioning the stream based on time intervals, sorting each partition separately, and then recombining the sub-streams in order according to their interval.

### C. RADISH-L: Streaming Machine Learning

RADISH-L ingests events from the Harmonizer, extracts and aggregates relevant features from these events, and dynamically creates statistical models representing patterns of normalcy that are then utilized by RADISH-A. This module has to operate under near real-time constraints to rapidly provide accurate models at high data velocity without using large amounts of storage.

Typical streaming machine learning algorithms face several challenges. First, raw data must be converted to a representation that can be efficiently utilized by the specific algorithm. These representations are known in the machine learning community as *feature vectors*. For streaming data, multiple events in the stream are usually aggregated to construct a single feature vector. For instance, all the events from a single session of a user's interaction with a database can be aggregated to derive a single feature vector represention of the session.

For our insider threat detection system, anomalies can usually be explained as a series of abnormal actions, e.g. a user logs on outside of their usual working hours, connects an external device, such as a USB drive, to their workstation, and transfers a large amount of data. Thus, the system needs to maintain a state object, corresponding to a feature vector, that is continually updated as data arrives. The system also needs to maintain criteria that dictate when to pass the featuer vector to the trainer to update its internal parameters. An additional challenge in developing models on streaming data is that machine learning algorithms are typically designed for the case when the data is available all at once. It is non-trivial to modify the algorithms for the case when data arrives sequentially, requiring the models to be incrementally updated. Thus, our system requires specialized machine learning algorithms for dynamically processing streaming data.

In particular, we need algorithms that scale to large data sets and provide accurate results at high processing throughput. We consider two subtasks here – streaming feature vector generation and streaming machine learning. With current streaming platforms, such as Apache Spark [17] or Apache Storm [16], the key to scaling to high data rates is the ability to parallelize each subtask by (logically) dividing the data into multiple substreams, perhaps one per entity, or by partitioning the stream by key.

For instance, to detect unusual user actions signifying potential insider threats, we may want to derive per-user streams and construct feature vectors for each user through a sequential process that maintains state for the user's current session and then outputs a feature vector aggregating all events within that session. A session may be determined by the interval between user logon and logoff, or by other given measures, such as an regular time period (hour or day). Machine learning then occurs on a stream of such feature vectors. Further research will be conducted on which session definitions yield optimal results.

**Online model management and updates.** Trained models built from streaming data are passed to RADISH-A for detecting anomalies. For both research and production systems, we use a database to store serialized models for several reasons. First, keeping track of models trained in the past can give insight into how the nature of more recent data has changed and can be useful for detecting concept drift. Second, it allows trained models to be loaded and used immediately or further updated with the most recent data, allowing for greater flexibility in both testing and production environments.

### D. RADISH-A: Streaming Anomaly Detection

RADISH-A, Figure 3, uses the models ascertained during the learning phase to identify abnormalities in the event stream coming from the Harmonizer. Abnormalities, if determined to be of sufficient risk, are then elevated to an alert and reported to analysts or security operators via the controller.

In order to make the above determination, the RADISH-A utilizes a multi-tiered structure. At the lowest tier, normalized events from the Harmonizer are processed using specific rules determined from RADISH-L. We refer to such as "Tier

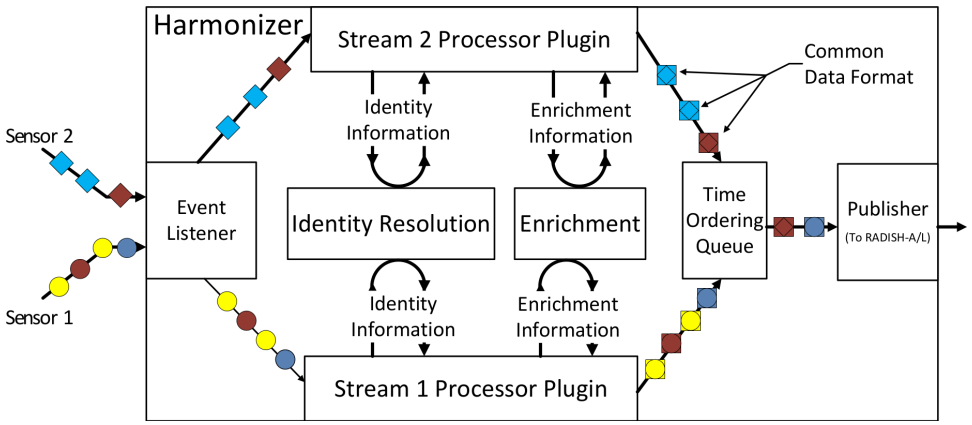



Fig. 2: Design of the stream harmonizer - The Harmonizer prepares the stream for further analysis by merging the data streams from the various sensors, performs identity resolution and enrichment, provides a common data format, and attempts to time order the events.

1" events; currently we consider different types of Tier 1 events, such as those related to credential fraud, inappropriate data access, or data exfiltration. The rules for detecting such events are specific to the user or entity involved. Multiple Tier 1 events are combined into Tier 2 events that focus on macro-level alerts that implement system-wide or role-specifc policies. Such alerts include the underlying Tier 1 events that contributed to the Tier 2 events. While we use only two tiers in the current implementation and architecture, additional tiers can easily be added.

We chose this multi-tier architecture for the RADISH system because it lends itself to a clean separation of logic between what constitutes suspicious activity and what conditions should warrant an alert. The benefit of this separation becomes apparent when considering the trade-off that naturally occurs when balancing Type I (false positive) and Type II (false negative) errors. In order to reduce Type I errors, a system's rules need to be made specific, capturing the proclivities of individual situations which may be out of the ordinary, but not malicious. This specificity tends to make the rules cumbersome, complicated, and brittle (i.e. we would have to look for a very specific set of events to fire an alert for a specific user), leading to an increase in Type II errors. The multi-tier system allows us to specify what is considered suspicious at the level of the individual user, asset, or resource, where we can take into account unique aspects of the object under consideration automatically via machine learning techniques. At the higher level, we can then define generalized policies as to which types of suspicious behaviors warrant an alert. This allows us, to a certain extent, to minimize both Type I and Type II errors.

Furthermore, the multi-tier system facilitates maintaining and augmenting the system while communicating the cause of alerts to the security analyst. As it turns out, the functions are linked in a very natural way. As a general rule, the complexity of a system has a direct impact on the difficulty of elucidating why a particular aspect of the system is set up the way it is and of changing an element to either fix or augment the system. For the complex event engine, the lower tier is purely dedicated to identifying what is suspicious activity and, as such, gives rise to relatively simple rules that are easy to follow (e.g. User X causes an inappropriate access event because they have accessed a database they usually don't use). Similarly, the second tier focuses purely on what constitutes an alert (e.g. User X caused an alert because they had 5 inappropriate access events in the last 30 minutes). When communicating this to the end user, the information is already grouped in a manner that facilitates and invites further exploration (from alert to originating suspicious events), so that they can determine the appropriate response action.

### E. Controller

Finally, the RADISH system includes a graphical user interface that is organized around the concept of an "analyst dashboard". The main window acts as a container for various component displays and controls that allow the security analyst to control the operation of the system and maintain situational awareness using data visualization. The data comes from two main sources, alerts from RADISH-A and sensor events from the Harmonizer. These are updated over time while a context of alerts and behaviors is built up for each user, and are displayed in a fashion that allows the analyst to see the correlated behaviors that lead to alerts.

## IV. Implementation

The described RADISH system has been implemented as a prototype in order to demonstrate the feasibility and efficacy of the chosen architecture. In this section, we provide details of our implementation.

### A. Runtime Environment

In order to verify that the proposed system can operate in a fully distributed manner, the system was run on a small cluster, Table I, comprising three Dell PowerEdge T105 Tower Servers networked through a D-Link DGS-2208 8 port switch. Each node in the cluster was configured with Java 1.8.0_20, zeromq-2.2.0, Apache-Hadoop 2.5.0 and Spark 0.9.1 compiled against Hadoop 2. All inter-process communication was effected using the zeroMQ socket library.

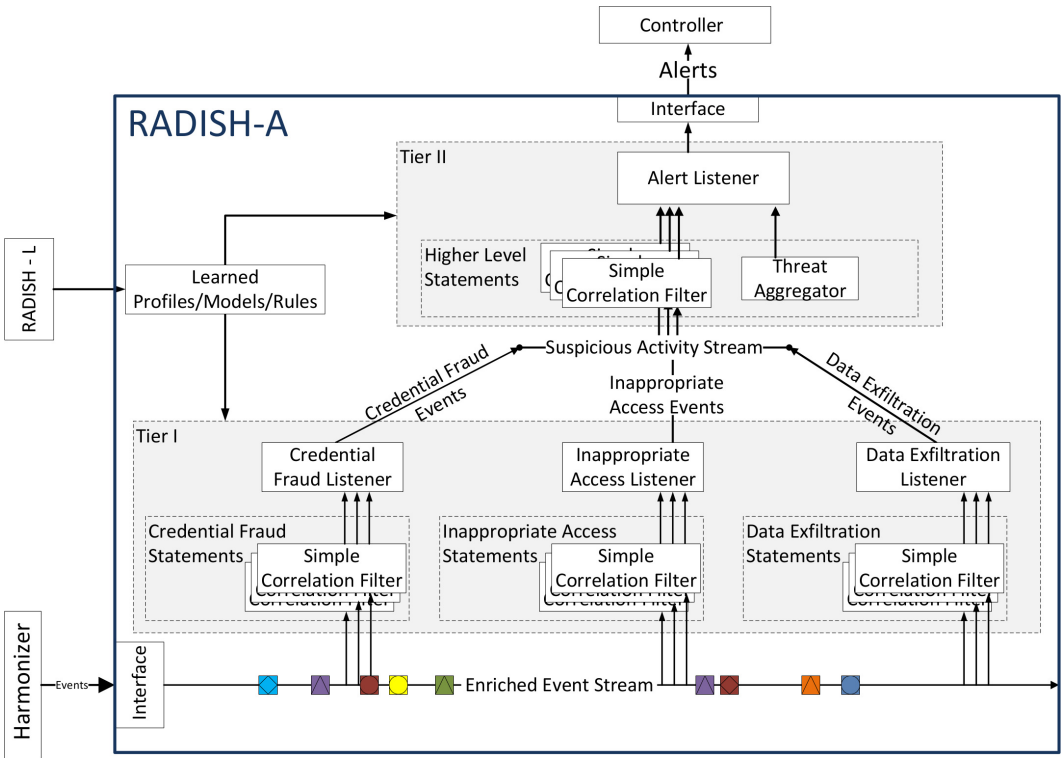



Fig. 3: Architectural framework of the RADISH-A system. The lower tier (Tier I) uses automatically trained models to identify anomalous activities in the stream. Tier II uses a combination of hand-coded rules and learned models to aggregate anomalous activities into alerts.

### *B. Component Implementation*

In this prototype implementation, the harmonizer and sensors are written in Scala. The sensors are assumed to generate data in CSV format; wrappers can easily be written to accommodate other streaming data formats. For experimentation, evaluation, and demonstration purposes, we augmented the harmonizer with the capability to read data from pre-determined log files in CSV format representing "pre-recorded" data. Since, for non-production purposes, we do not necessarily want the system run time to correspond exactly to the data event times, the harmonizer contains an internal clock that can be set to run at an arbitrary multiple of wall time. The harmonizer dispatches events to RADISH-A and RADISH-L according to the timestamp of the event. Enrichment and identity resolution are performed using in-memory maps that relate primary and universal identifiers. Inter-process communication between the Harmonizer and RADISH-A/L follows the Publish-Subscribe (pub/sub) design pattern. In this case, the harmonizer publishes to an "events" topic, to which the controller, RADISH-A, and RADISH-L subscribe.

A typical dashboard layout was created for the prototype using JavaFX selected since it is part of the current Java core language. It supports simple 2D graphics as well as more complex 3D visualizations and animations. Sensor data is presented on the right side of the UI and allows the analyst to see the correlated behaviors that led to alerts and also create a forensic timeline. Detailed information on each alert event is also logged and displayed in a console window. This detailed data can also be used to investigate the nature of alert events. Event and alert data can be obtained by subscribing to the harmonizer and alerts topic respectively. Commands are issued to other processes (harmonizer, RADISH-A, and RADISH-L) using a request-reply pattern [23].

The streaming feature vector generation and online modeling training components of RADISH-L are built using Apache's Spark Streaming engine. Spark Streaming leverages Spark's Resilient Distributed Dataset (RDD) data structure by treating a stream as a sequence of batch RDDs, known as discretized streams (DStreams). Each RDD is processed through a sequence of transformations such as map and reduce; our streaming feature generation and machine learning modules are written as a pipeline of such transformations.

RADISH-A has been implemented using ESPER, an open source Complex Event Processing (CEP) framework [24]. We chose to implement the prototype system using ESPER with an eye towards extending our models to perform more complex event monitoring tasks such as frequent episode mining. ESPER's API includes a straightforward regular expression-like syntax for identifying multi-stage events in a stream which will greatly facilitate capturing episodes of interest. All alerts generated by RADISH-A are published to the "alerts" topic that is monitored by the controller.

Separate frameworks were chosen for implementing RADISH-A and RADISH-L to address the differing needs of the learning and alerting process. Spark-Streaming allows for very fine grained control over the events in the stream which

facilitates model training and pattern discovery. ESPER's high level abstraction for specifying event patterns allows the rapid implementation and maintenance of alerting rules expressed in a straightforward and relatively easy to understand manner.

TABLE I: Spark Cluster Properties

| | Hydra-01 | Hydra-02 | Hydra-03 |
|---|---|---|---|
| HDFS Function | Data Node | Data Node | Data Node<br>Name Node |
| Spark Function | Slave Node | Slave Node | Slave Node<br>Master Node |
| Processor | Quad-Core AMD Opteron™ 1300 Series | Quad-Core AMD Opteron™ 1300 Series | Quad-Core AMD Opteron™ 1300 Series |
| Storage | 500 GB Main<br>2 TB HDFS | 150 GB Main<br>2 TB HDFS | 150 GB Main<br>500 GB HDFS |
| RAM | 8 GB | 8 GB | 8 GB |

## V. Insider Threat Detection Using RADISH

We describe the application of RADISH to the insider threat problem along with the corresponding experimental results in terms of accuracy and speed.

### A. Description of the Dataset

The DARPA ADAMS dataset [5] was synthesized by ExactData LLC using a dynamic data generator tool. It consists of a number of log files, including one file for Logon/Logoff events, one for events related to removable devices, one for HTTP accesses, one for email messages, and one for information in the LDAP directory. We replay the data as a continuous stream of events in timestamp order for real-time streaming analysis by RADISH in order to detect anomalous and possibly malicious behavior of system insiders in this system.

Ten sample synthetic datasets created by ExactData are available from the CERT Insider Threat Center at CMU's SEI. They are numbered according to the generator version that was used to create them. For this report, we used the r2 dataset which simulates the behavior of 1,000 employees recorded over 494 days, resulting in more than 430 million events (see Table II). Among more than 375 thousand sessions, 6 sessions are known to be malicious. This low number of malicious sessions represents a realistic scenario, where the objective is to find the proverbial needle in haystack while keeping the false positives to a minimum. We labeled a session as malicious, if the session contains one or more events that were part of a malicious activity.

### B. Data Processing Method

RADISH processes this data as follows (see Figure 4). The **sensors** consist of the different log files. Thus we have one sensor each for HTTP logs, Logon events, events on removable devices, and email messages.

TABLE II: Statistic of the CERT Insider Threat r2 dataset

| r2 | Count |
|---|---|
| Number of Users | 1,000 |
| Number of Days | 494 |
| Number of Events | 434,624,511 |
| Number of Sessions | 375,678 |
| Number of Malicious Sessions | 6 |

The **harmonizer** joins the data streams from all the sensors resulting in a composite stream of all system events in timestamp order. This composite stream is then partitioned into substreams consisting of a timestamp ordered sequence of events for each user (*per-user stream*). The harmonizer handles streams in real-time, typically without needing to buffer large amounts of data (larger buffers are needed when the different sensor sources are not synchronized in time).

The first step in RADISH-L is **streaming segmentation**, Performed at the level of a *user session*. Each per-user stream is further divided into a number of *sessions* consisting of all events within a single login session of a user. Each session is further treated as a single (complex) object that is the focus of further analysis.

The next step is **feature vector generation**. Each user session, which consists of a complex sequence of events, is converted into a feature vector that can be processed by a machine learning algorithm. For the ADAMS dataset, we use five attributes to represent a user session as a feature vector: the start hour, the duration (in minutes), the number of emails sent, the presence/absence of removable media (a binary attribute), and the number of HTTP accesses. These features were chosen to be easily interpretable and readily available in the dataset. Feature vectors of sessions are constructed and updated dynamically as more events arrive in the session. When a session is complete, the feature vector corresponding to the session is normalized so that the attribute along each dimension lies between 0 and 1. Note, this normalization step is important so that no single dimension dominates all others. The normalized vectors are then used for machine learning. It must be noted that for our particular application of anomaly detection, we adopt unsupervised machine learning algorithms to determine if any single feature vector deviates appreciably from its neighboring points. Thus, there is no a-priori labeling of each vector from which to learn a discriminative classifier. In general though, the processing framework described above can use both supervised and unsupervised methods.

Streaming versions of machine learning or data mining tasks such as classification, clustering, or anomaly detection require significantly different methods than those used by traditional batch techniques. We illustrate the issues involved in the context of designing a streaming version of the Unsupervised k-nearest neighbor (kNN) anomaly detector [25] that we have used in out current implementation. Going forward in this paper, unless otherwise indicated, kNN will always refer to the streaming unsupervised version of kNN. Likewise, Point will always refer to a user session. In the kNN anomaly detector, when a new point arrives, we compute the $k$th smallest distance to the points observed so far and use this as the anomaly score of the point. If the anomaly score exceeds a threshold $\tau$, then the point is regarded as anomalous. The first few points are used to learn typical behavior of the user session. Among the anomalous sessions that are found, particular attention will be paid to ones that access both files and removable media devices, since these will help identify suspicious sessions. One potential issue with this approach

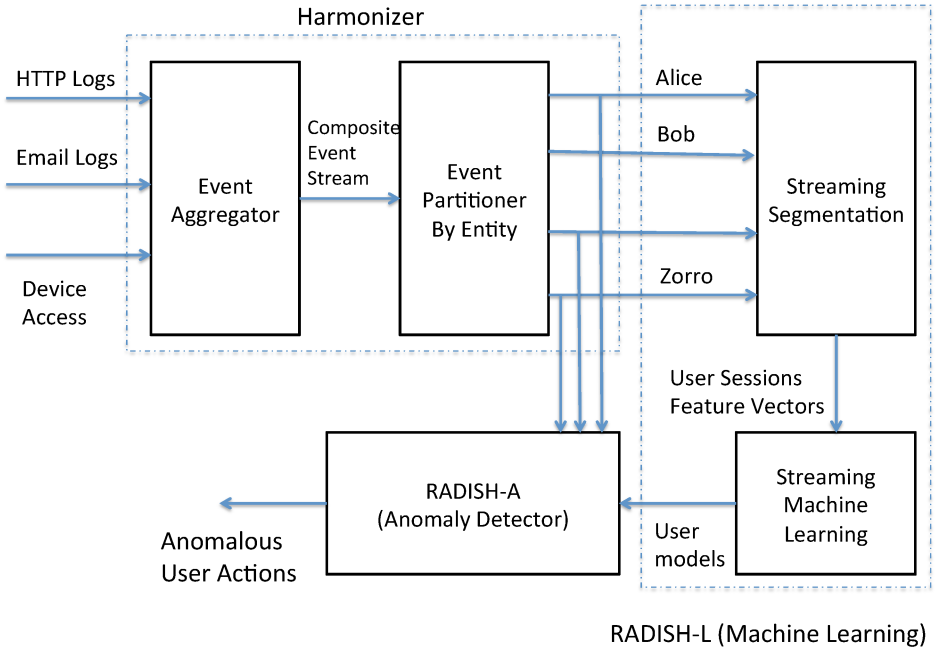



Fig. 4: Data Processing Pipeline for Detecting Insider Threats

is that if the first few sessions are in any way anomalous to an individual's normal behavior, the model may exhibit some undesirable characteristics, such as generating many initial false positives. This is alleviated by incorporating bayesian priors, domain expertise, or additional information about the new users to initialize expected behavior.

The naïve algorithm for implementing kNN is as follows: For every new point $s$ that arrives, we compute the distance of $s$ from all other points seen so far. This allows us to compute the kNN metric of the point that is based on the $k$-th smallest distance. However, this method scales poorly as the data size increases, since its total runtime over $n$ arrivals increases as $\Theta(n^2)$, making it unsuitable for handling large data sets.

A faster implementation of kNN is obtained as follows: A k-d tree [26] is a data structure for multi-dimensional points that organizes points in a tree by recursively partitioning the universe using axis-parallel hyperplanes. These provide an efficient solution to a number of multi-dimensional data structuring problems, including range searching and nearest neighbors. We adapt the k-d tree for finding the k nearest neighbors of a query point. One issue with the k-d tree is that the data structure is static, i.e. the points have to all be provided in advance before the data structure is constructed. We derive a dynamic k-d tree through applying the Bentley-Saxe transform [27] to the static k-d tree [26]. This allows us to design a dynamic data structure by repeatedly constructing larger (static) k-d trees through merging smaller k-d trees (see also [28]).

The dynamic structure based on k-d trees is much faster than the naïve implementation whose cost grows as $O(n^2)$ (for $n$ insertions). To compare the costs, we ran an experiment

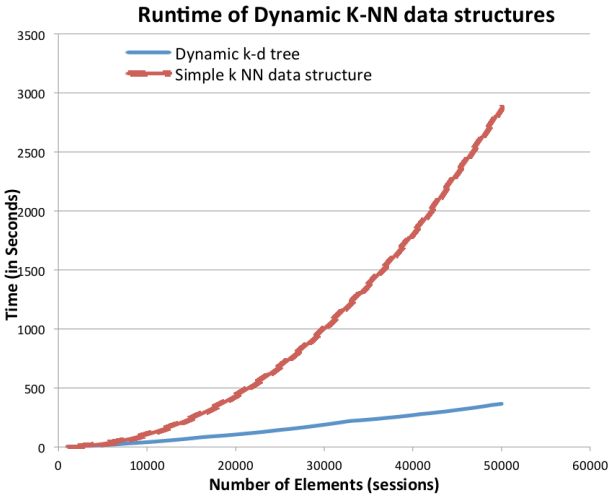


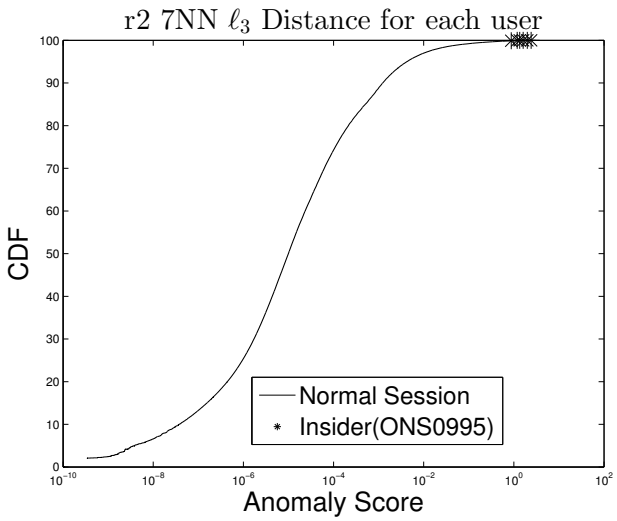

Fig. 5: The dynamic k-d tree implementation is much faster than simple kNN data structures on large datasets.

comparing our implementations of both data structures when applied to kNN, testing using synthetic data. The experiment inserted a sequence of $n$ points and, before each insertion, queried for the set of $k = 3$ nearest neighbors to the inserted point. The total time is shown in Figure 5. This clearly shows that the algorithm for kNN based on a dynamic k-d tree significantly outperforms the naïve algorithm. The runtime of the kNN algorithm increases almost linearly with the size of the stream and the number of queries.

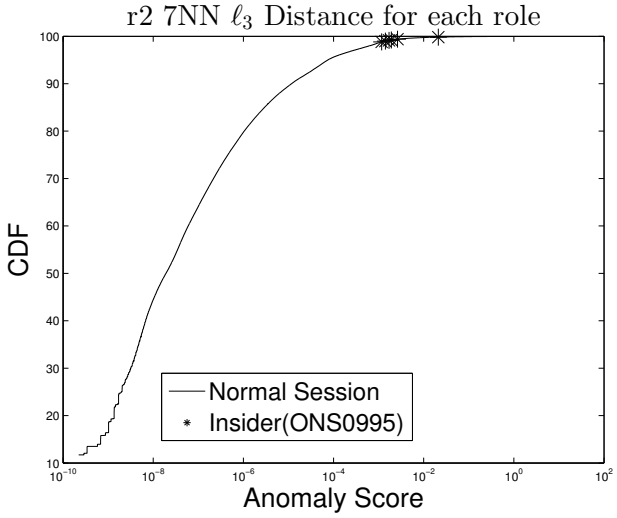
Our algorithm for kNN anomaly detection is built using the dynamic k-d tree index described above. The first few sessions for each user are used only for training and anomaly detection is not performed. Then upon arrival of a new session, the system checks if it is anomalous using the kNN query using the index. The point is also inserted into the index. If the session is deemed anomalous then an alert is generated.

We briefly contrast the streaming with batch kNN. In batch kNN, the entire batch of points is analyzed all at once. For each data point, the $k$th smallest distance to another point in the set is used to compute the anomaly score of the point. There is room to pre-process the entire dataset so that nearest neighbor distance queries are answered as quickly as possible. The statically generated index tends to be faster than the dynamically generated one, so the total time of anomaly detection using streaming kNN is greater than the total time using the static kNN data structure.

The set of attributes used in our feature vectors are primarily chosen to achieve transparency and interpretibility in our models, and to serve as a baseline for more sophisticated models to come in the future. Partitioning events by user sessions has the intuitive meaning of grouping sequences of events between natural breaking points in a user's day. Sliding window-based approaches that evaluate sequences of events every hour or several hours are a natural extension of this idea and address, at least partially, the problem of delayed detection that is inherent to using a session-based approach. Further feature transformation and extraction techniques along with nonlinear machine learning models may also improve accuracy on this dataset. Nevertheless, exploring these ideas, among various others, is beyond the scope of this current work, which is intended to be a proof-of-concept framework for applying streaming anomaly detection algorithms at scale, where any set of features and machine learning algorithms may be plugged in easily. Follow on work will consider a deeper analysis of the aforementioned topics and will also use real rather than synthetic datasets.




(a) User-based Detection




(b) Role-based Detection

Fig. 6: Sessions ranked by the $\ell_3$ 7NN score for the r2 dataset, using user-based (6a) and role-based (6b) partitioning.

### C. Experimental Results

We now describe the results of applying RADISH to the ADAMS sets in terms of accuracy and performance.

**Accuracy:** We considered two approaches for event partitioning: 1) User-based detection, and 2) Role-based detection. In user-based detection, we computed the kNN anomaly score of each user session with respect to previous sessions of the same user (seen so far). In role-based detection, we computed the kNN anomaly score of each user session with respect to all previous sessions of all users with the same role within the organization, as observed in the LDAP user directory (also provided as a part of the dataset).

Figure 6 shows the cummulative distribution function (CDF) of the anomaly scores derived applying both user- and role-based detection for over 375 thousand sessions of the r2 dataset, including the 6 malicious sessions. The sessions are ranked in increasing order of their anomaly scores. For user-based detection, the malicious sessions are ranked in the top 500 sessions. For role-based detection, the malicious sessions are ranked in the top 10,000 sessions. Thus, at least in this instance, user-based partitioning is more accurate than role-based partitioning. Our system will provide support for different types of partitioning as appropriate.

For illustration, we examine the characteristics of representative feature vectors in user-based detection, to see how user behavior affects the anomaly score of the coresponding session. Figures 6a and 6b show the session data and raw feature vectors for one malicious and one normal session by ONS0995, a malicious user. For simplicity, we compare the raw feature vectors of the malicious and normal sessions, that are determined directly from the associated session events. For a complete determination of the anomaly score, the normalized feature vector is used, which relies on knowledge of all prior feature vectors for a user. The normal session, Figure (7b), with an anomaly score of $5.3 \times 10^{-3}$, is in some sense, representative of the normal behavior of the user. Comparing the feature vector of the normal session with that of the mali-

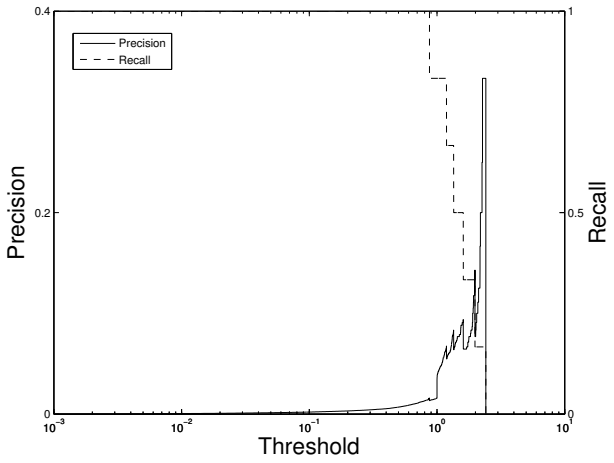
cious session, Figure (7a), we can see that for an $\ell_3$ distance measure, three components will contribute significantly and result in a large anomaly score; 1) the unusual start time; 2) the lack of significant e-mail activity; and 3) the mounting of an external storage device.

Applying a specific threshold on the anomaly score leads to a method that can classify user sessions as either malicious or benign. Figure 8 shows the precision and recall using the two methods described above, user-based detection and role-based detection, as a function of the threshold.

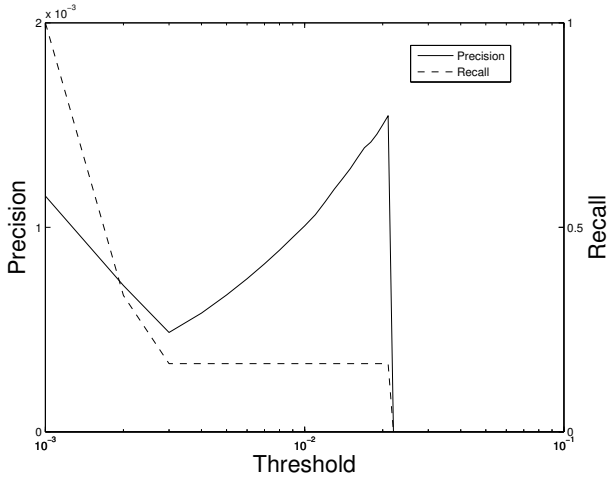
We note there is a scarcity of published work providing concrete precision and recall results of insider threat detection using public datasets, including the DARPA ADAMS dataset. To put our results in context, consider the user-based detection results in Figure 8. A threshold value slightly greater than 1 yields a recall of about 0.5 and a precision of about 0.08. Thus, about 50 percent of all malicious sessions are detected and about 92 percent of sessions that are flagged malicious are actually benign. While this may seem high, the amount of labor required to follow up on the false positives is still lower than the potential losses caused by an undetected malicious event. For example, suppose an enterprise security operator takes on average four hours to adjudicate a detected session as truly malicious or not and costs $200,000 per year working 40 hours a week. Over the 18 month duration of the ADAMS set, continuous 24x7 monitoring would require 4.2 operators costing $1.26 million to adjudicate a total of 3,285 sessions. According to [29] the average cost to remediate a successful insider attack is $445,000, or $2.67 million for all 6 malicious sessions of the ADAMS set that would have been detected. Thus, when compared with the significant losses that could be incurred by an insider attack, the cost of handling false alarms is small.

For user-based detection, we observe that precision increases monotonically with the threshold and recall decreases monotonically. For role-based detection, the precision is much worse than with user-based detection, and the best precision we can achieve is about $0.0016$. Role-based detection is doing worse because there are "normal" sessions from other users that are close to the malicious sessions of the insider that lead to a smaller anomaly score for the malicious sessions. Broadening the class of sessions to all sessions of users with the same role does not seem to help, in this case.

**Performance:** Figure 9 shows the result of our study investigating the performance of the RADISH-L training component with an eye towards estimating the maximum capacity (in events/second) that can be handled by the trainer. We examine the batch processing time, i.e. the amount of time it takes for a 10 second mini-batch of data to be processed, as a function of the average throughput of the system. The time it takes to process each batch is a critical parameter for ensuring that all data arriving to the system can be fully processed before ingesting the next batch. If the batch processing time exceeds the delay between batches, the system can become unstable. This scan was repeated for a single node pseudo distributed cluster and a fully distributed three node cluster. As can be seen in Figure 9, at 230 events per second, the single node system is barely taxing the CPU, with spare batch processing




(a) User-based Detection




(b) Role-based Detection

Fig. 8: Precision and recall as a function of the threshold with 7NN using $\ell_3$ distance on r2 dataset

time of approximately 8.4s. Assuming a linear increase in delay with throughput, a least squares fit to the data in Figure 9 yields an estimate of the relationship between throughput and delay of $delay_{1Node} = 0.0064 * throughput + 0.0587$. Given this estimate, we expect that the pseudo cluster can handle an average throughput of approximately 1,550 events per second. As would be expected, expanding to a three node cluster reduces the batch processing time by approximately a third; $delay_{3Node} = 0.0021 * throughput + 0.3288$. At the low end of the throughput, approximately 45 events per second, the pseudo cluster and the three node cluster have near identical performance. This odd behavior results from the fact that the load on the cluster CPUs is very light at these low volumes and the major cost associated with the processing is incurred by network traffic and the overhead of administering a process over a distributed cluster.

## VI. Future Work and Conclusion

**Future Work.** The RADISH system described above aims to extend the massively parallel on-line machine learning techniques pioneered by companies such as Amazon and LinkedIn

| Source | Event Log Line |
|---|---|
| logon.csv | 3/19/2010 22:04:38,ONS0995,PC-3585,Logon |
| device.csv | 3/20/2010 1:47:28,ONS0995,PC-3585,Insert |
| http.csv | 3/20/2010 01:59:32,ONS0995,PC-3585,http://wikileaks.org |
| device.csv | 3/20/2010 5:38:08,ONS0995,PC-3585,Remove |
| logon.csv | 3/20/2010 8:10:12,ONS0995,PC-3585,Logoff |

Raw Feature Vector

| | |
|---|---|
| 22 | logon hour |
| 606 | duration |
| 1 | # http |
| 0 | # file |
| 0 | # e-mails |
| 1 | USB connect |

(a) Malicious Session for user ONS0995; User-based detection anomaly score = 1.8

| Source | Event Log Line |
|---|---|
| logon.csv | 2/15/2010 8:45:00,ONS0995,PC-3585,Logon |
| email.csv | 2/15/2010 9:38:46,Cameran.Mira.Short@dtaa.com…,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 13:24:11,Gareth.Erich.Cash@dtaa.com…,Otto.Nero.Schwartz@dtaa.com |
| http.csv | 02/15/2010 14:09:53,ONS0995,PC-3585,http://bright.net |
| email.csv | 2/15/2010 16:21:44,Cameran.Mira.Short@dtaa.com,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 18:07:20,Gareth.Erich.Cash@dtaa.com,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 18:53:45,Alfonso.Maxwell.Phelps@dtaa.com,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 19:43:16,Cameran.Mira.Short@dtaa.com,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 20:39:08,Gareth.Erich.Cash@dtaa.com,Otto.Nero.Schwartz@dtaa.com |
| email.csv | 2/15/2010 20:41:58,Cameran.Mira.Short@dtaa.com…,Otto.Nero.Schwartz@dtaa.com |
| logon.csv | 2/15/2010 20:46:00,ONS0995,PC-3585,Logoff |

Raw Feature Vector

| | |
|---|---|
| 8 | logon hour |
| 721 | duration |
| 1 | # http |
| 0 | # file |
| 8 | # e-mails |
| 0 | USB connect |

(b) Normal Session for user ONS0995; User-based detection anomaly score = $5.3 \times 10^{-3}$

Fig. 7: Example event data and feature vectors for a malicious (7a) and normal (7b) session.

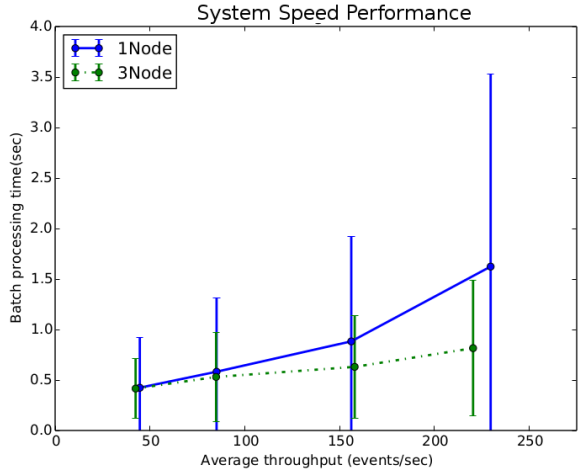



Fig. 9: Batch processing delay for the training system on a one node and three node clusters.

to provide a comprehensive system for anomaly detection in general, and for detecting and alerting on insider threats specifically. RADISH is capable of continuously modeling the actions of individual elements of an organization and then identifying suspicious activities in real time. Furthermore, the system is readily scalable due to its design using open source software for distributed processing frameworks and commodity hardware.

Further work will augment this system as follows:

1) Implement and test the system with much larger data and stream sizes
2) Incorporate a feedback mechanism to include data that identified by RADISH and labeled by an analyst
3) Model session evolution so that abnormality conditions can be determined while the session is still in progress
4) Model additional aspects of the organization, including document and hardware resources

**Conclusion.** RADISH aims to ameliorate the modern insider threat issue. Using real-time streaming data analytics and machine learning techniques, RADISH automatically identifies the normal behaviors within an organization, allowing security operators to quickly focus on unusual and suspicious activities. By encompassing analysis of, and correlation among, multi-stream data, RADISH also ensures that bread crumbs to malicious activities that are spread across numerous domains do not go unnoticed as they might in siloed applications intended to protect individual assets.

As a comprehensive system, RADISH enables organizations to focus their security personnel where they can be the most effective. The automatic aspect of the learning phase frees the analyst from the tedious and time consuming task of profiling individual users of a system. Counter intuitively, this automation given the analyst the capacity to spread her investigation to all levels of the organization, not just individuals that have been identified to pose a high threat. Further, the multi-tiered detection system allows analysts to operate at the highest level in the information chain, crafting policies that can be applied across any segment of the organization. The open architecture allows easy implementation of custom sensors for analyzing behavior and/or Tier 1 logic for identifying suspicious events. Together, these features allow RADISH to be tailored to a specific organization at all levels.

## References

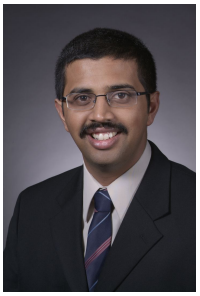

[1] J. Lewis and S. Baker, "The Economic Impact of Cybercrime and Cyber Espionage," Centre for Strategic and International Studies, Tech. Rep., July 2013. [Online]. Available: http://www.mcafee.com/us/resources/reports/rp-economic-impact-cybercrime.pdf

[2] T. Stapleton, "Data Breach Cost," Zurich American Insurance Corp, Tech. Rep., July 2012. [Online]. Available: http://http://www.zurichna.com/

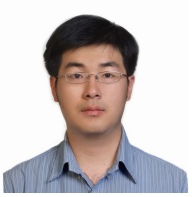
[3] C. Miller, "Revelations of NSA Spying Cost US Tech Companies," New York Times, Tech. Rep., March 2014. [Online]. Available: http://www.nytimes.com/2014/03/22/business/fallout-from-snowden-hurting-bottom-line-of-tech-companies.html
[4] M. Riley, "Exxon, Shell, BP Said to Have Been Hacked Through Chinese Internet Servers," Bloomberg L.P., Tech. Rep., February 2011. [Online]. Available: http://www.bloomberg.com/news/2011-02-24/exxon-shell-bp-said-to-have-been-hacked-through-chinese-internet-servers.html
[5] The CERT Division and ExactData LLC, *Insider Threat Tools, the CERT division*, accessed December, 2015. [Online]. Available: https://www.cert.org/insider-threat/tools/
[6] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *Security and Privacy Workshops (SPW), 2013 IEEE*. IEEE, 2013, pp. 98–104.
[7] *Wave - Data Protection*, accessed February, 2014. [Online]. Available: https://wave.com/data-protection
[8] *SureView, Raytheon Institute*, accessed February, 2014. [Online]. Available: https://www.trustedcs.com/products/SureView.html
[9] A. Cummings, T. Lewellen, D. McIntire, A. P. Moore, and R. F. Trzeciak, "Insider Threat Study: Illicit Cyber Activity Involving Fraud in the U.S. Financial Services Sector," Carnegie Mellon University-Software Engineering Institute, Tech. Rep., July 2012. [Online]. Available: http://www.sei.cmu.edu/reports/12sr004.pdf
[10] *Palisade - Cyber Security Intelligence Management, Lockheed Martin*, accessed February, 2014. [Online]. Available: http://www.lockheedmartin.com/us/what-we-do/information-technology/cyber-security/cyber-intelligence-enterprise.html
[11] *Prelert*, accessed February, 2014. [Online]. Available: http://info.prelert.com/
[12] *Securonix*, accessed February, 2014. [Online]. Available: http://www.securonix.com/
[13] H. Eldardiry, E. Bart, J. Liu, J. Hanley, B. Price, and O. Brdiczka, "Multi-domain information fusion for insider threat detection." in *IEEE Symposium on Security and Privacy Workshops*. IEEE Computer Society, 2013, pp. 45–51. [Online]. Available: http://dblp.uni-trier.de/db/conf/sp/spw2013.html#EldardiryBLHPB13
[14] M. Wall, "Big data: Are you ready for blast-off?" British Broadcasting Corporation, Tech. Rep., March 2014. [Online]. Available: http://www.bbc.com/news/business-26383058
[15] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A Distributed Messaging System for Log Processing," in *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB)*, 2011, pp. 2–4.
[16] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. V. Ryaboy, "StormTwitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014, pp. 147–156.
[17] *Apache Spark Streaming*, accessed February, 2014. [Online]. Available: https://spark.apache.org/streaming/
[18] B. Snyder, D. Bosnanac, and R. Davies, *ActiveMQ in Action*. Manning, 2011.
[19] P. Hintjens, *Ømq-The Guide*, accessed February, 2014. [Online]. Available: http://zguide.zeromq.org/page:all
[20] *Detailed Auditing of Privileged User Sessions on Windows, Linux and UNIX Systems, Centrify Server Suite*, accessed February, 2014. [Online]. Available: http://www.centrify.com/enterprise-edition/overview.asp
[21] *InfoSphere Guardium Data Security, IBM*, accessed February, 2014. [Online]. Available: http://www-01.ibm.com/software/data/guardium/
[22] "Spotting the Adversary with Windows Event Log Monitoring," National Security Agency/Central Security Service, Tech. Rep., December 2013. [Online]. Available: http://cryptome.org/2014/01/nsa-windows-event.pdf
[23] *28/REQREP - ZeroMQ Request-Reply Pattern*, accessed February, 2014. [Online]. Available: http://rfc.zeromq.org/spec:28
[24] (accessed June, 2013) EsperTech Inc. Event Stream Intelligence. [Online]. Available: http://www.espertech.com/
[25] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009.
[26] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.
[27] J. L. Bentley and J. B. Saxe, "Decomposable searching problems I: static-to-dynamic transformation," *J. Algorithms*, vol. 1, no. 4, pp. 301–358, 1980.
[28] B. Naidan and M. L. Hetland, "Static-to-dynamic transformation for metric indexing structures (extended version)," *Inf. Syst.*, vol. 45, pp. 48–60, 2014.
[29] H. Schulze, "Insider Threat Spotlight Report," http://images.response.spectorsoft.com/Web/SpectorSoftCorporation/%7B8788ff89-b481-45ed-bdce-efc40f75457b%7D_062315_Insider_Threat_Report_Spectorsoft.pdf, 2015.

**Dr. Brock Böse** is a Data Scientist in Northrop Grumman's ISR Division. After receiving his PhD in Plasma Physics at MIT in 2010, he joined Northrop Grumman as a Future Technology Leader. After a two-year stint at Accenture's Architectural Innovation Group he rejoined Northrop Grumman in his current position where he is working on various R&D projects in machine learning.

**Bhargav Avasarala** is a Data Scientist in the ISR division of Northrop Grumman. He graduated with his Masters in Electrical Engineering: Systems at the University of Michigan in 2010, and joined Northrop Grumman as a Future Technology Leader. He primarily works on applying machine learning techniques to real-world data.



**Dr. Srikanta Tirthapura** received his Ph.D. in Computer Science from Brown University in 2002, and his B.Tech. in Computer Science and Engineering from IIT Madras in 1996. He is an Associate Professor in the department of Electrical and Computer Engineering at Iowa State University. He has worked at Oracle Corporation and is a recipient of the IBM Faculty Award (twice), and the Warren Boast Award for excellence in Undergraduate Teaching. His research interests include cybersecurity and streaming analytics.



**Yung-Yu Chung** received his bachelors degree from National Taiwan University, Taipei, Taiwan in 2007, and a masters degree from Iowa State University (ISU), in 2010. He expects to receive his Ph.D. in computer engineering from ISU in 2016. He worked as a Research Assistant with National Taiwan University from 2008-09, and with Academia Sinica, Taipei from 2009-10. His research interests include algorithms for stream mining and machine learning.

**Dr. Donald Steiner** is Principal Technologist and Technical Fellow at Northrop Grumman. He is a Principal Investigator for R&D projects involving data analytics, cybersecurity, and cloud computing and manages the Northrop Grumman Cybersecurity Research Consortium. Previously, Dr. Steiner was CTO of Quantum Leap Innovations and Co-Founder and Chief Scientist of WebV2, Inc. He earned his PhD in Mathematics from Iowa State University in 1984. His research interests include artificial intelligence and multi-agent systems.