IEEE Open Journal of the
Computer Society

Received 20 March 2024; revised 26 April 2024; accepted 28 May 2024. Date of publication 30 May 2024; date of current version 12 June 2024. The review of this article was arranged by Associate Editor Deze Zeng.

Digital Object Identifier 10.1109/OJCS.2024.3407547

Anomaly Detection in Key-Management Activities Using Metadata: A Case Study and Framework

MIR ALI REZAZADEH BAEE [1] (Senior Member, IEEE), LEONIE SIMPSON [1], AND WARREN ARMSTRONG [2]

[1]School of Computer Science, Queensland University of Technology, Brisbane, QLD 4000, Australia
[2]QuintessenceLabs, Canberra, ACT 2609, Australia

CORRESPONDING AUTHOR: MIR ALI REZAZADEH BAEE (e-mail: info@baee.co).

The work was supported by the Cyber Security Research Center Ltd. whose activities are partially funded by the Australian Government's Cooperative Research Centers Program.

This article has supplementary downloadable material available at https://doi.org/10.1109/OJCS.2024.3407547, provided by the authors.

ABSTRACT Large scale enterprise networks often use Enterprise Key-Management (EKM) platforms for unified management of cryptographic keys. Monitoring access and usage patterns of EKM Systems (EKMS) may enable detection of anomalous (possibly malicious) activity in the enterprise network that is not detectable by other means. Analysis of enterprise system logs has been widely studied (for example at the operating system level). However, to the best of our knowledge, EKMS metadata has not been used for anomaly detection. In this article we present a framework for anomaly detection based on EKMS metadata. The framework involves automated outlier rejection, normal heuristics collection, automated anomaly detection, and system notification and integration with other security tools. This is developed through investigation of EKMS metadata, determining characteristics to extract for dataset generation, and looking for patterns from which behaviors can be inferred. For automated labeling and detection, a deep learning-based model is applied to the generated datasets: Long Short-Term Memory (LSTM) auto-encoder neural networks with specific parameters. This generates heuristics based on categories of behavior. As a proof of concept, we simulated an enterprise environment, collected the EKMS metadata, and deployed this framework. Our implementation used QuintessenceLabs EKMS. However, the framework is vendor neutral. The results demonstrate that our framework can accurately detect all anomalous enterprise network activities. This approach could be integrated with other enterprise information to enhance detection capabilities. Further, our proposal can be used as a general-purpose framework for anomaly detection and diagnosis.

INDEX TERMS Anomaly detection, deep learning, enterprise key-management system, framework, metadata analysis.

I. INTRODUCTION

Enterprise information systems produce massive volumes of data across a range of business operations. Much of this information is transmitted over communications networks. In response to rising security concerns, encryption is increasingly used to protect stored and transmitted enterprise data [1].

Applying encryption requires the use of cryptographic keys [2], [3]. Successful key management [4], [5] is necessary to achieve organizational data protection objectives. Organizations may use an Enterprise Key Management (EKM) platform for unified management of the various cryptographic keys required to secure organizational data.

Although encryption is intended as a security measure, it may also be used by attackers to facilitate exfiltration of data from an organization without detection, as network traffic inspection devices are unable to inspect encrypted payload data under these circumstances. EKM usage provides additional data that may be useful for anomaly detection [6]. This article presents the results of analysis of the metadata associated with the use of an EKM platform to determine the information

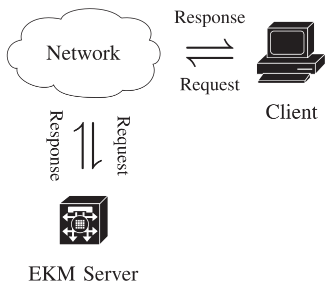

FIGURE 1. **The EKM network model.**

revealed about the use of encryption within the organization is useful in inferring whether activity is normal or abnormal.

The workflow of an EKM architecture and communication model can be considered as a Client-Server model, as illustrated in Fig. 1. Requests and responses commonly use the Key Management Interoperability Protocol (KMIP) [7] format. The client makes a request (e.g., a "Create-key" request) through the Application Programming Interface (API), using a client-specific representation. This can be encoded in a KMIP format and transmitted to the server. The server decodes the request using a KMIP decoder to form an intermediate representation, which is used by the server API to process the request.

Data is a collection of information such as observations, measurements, facts, and descriptions of certain things. Metadata is data about the information, but not the information itself. If it is created and handled properly, metadata can improve the usability of data [8]. In the context of EKMS, metadata includes data about the creation and use of the cryptographic keys, but not the keys themselves.

As an example of EKMS metadata, Listing 1 shows metadata associated with a "Create-key" request from a user, extracted from a log file generated using Rsyslog [9]. Note that the metadata colored in Red contains information needed to understand the creation of a cryptographic key, but it is not the value of the key itself.

Listing 1: EKMS metadata example (create-key request)

```
1-  Awaiting connection at 2021.09.05-01:19:36
2-  New connection accepted from 192.168.27.2:55598
3-  Connection server started
4-  ciphers: AES128-SHA:AES128-SHA256:AES256-
5-  checking ca
6-  Initialising_SSL ctx
7-  Supplied PIN null
8-  Selected OS entropy source on port 10002
9-  accepted connection from client (fd=7)
10- Subject = C=AU/ST=ACT/O=QuintessenceLabs
11- Subject = CN=QUT/C=AU/ST=QLD/L=Brisbane/O=QUT
12- Shared ciphers: AES256-SHA256
13- Current connection cipher AES256-SHA256
14- addr= 192.168.27.2, port=55598, cert_serial=46EE
15- issuer= C = AU, ST = ACT, O = QuintessenceLabs
16- Received 104 byte message
17- Operation Discover Versions succeeded
18- Operation allowed by the action policy
19- Received 384 byte message
20- operation= Create, addr=192.168.27.2, user=client
21- Operation allowed by global rule
22- Operation Create succeeded
23- Connection server stopped
```

Referring to Listing 1, we see that the following important questions are answered using the metadata in the log file:

- When was the connection established? *Line 1*
- Who was talking to us? *Line 2*
- What was the key derived from? *Line 8*
- Who was performing this request? *Line 14*
- Who authorized this client connection? *Line 15*
- What was the requested message size? *Line 16*
- What was the requested operation? *Line 20*

A. RESEARCH MOTIVATION

Collecting and analyzing metadata associated with enterprise activities under normal use conditions reveals patterns that can be used to generate heuristics-based rules that describe this normal behavior. Once these rules are established, future monitoring of systems and corresponding metadata analysis can be compared with the established baseline to identify anomalous events. A substantial difference from the baseline data may indicate a cyber security breach in the enterprise network.

B. RESEARCH AIM AND OBJECTIVES

The aim of this research is to develop a framework that can be applied to EKM metadata to generate heuristics based on categories of behavior that are useful in detecting anomalous enterprise network activities (perhaps indicating malicious activity on the enterprise network) involving cryptographic keys. The investigation of EKM metadata for use in this way represents a novel case study in metadata analysis.

The process of metadata analysis for anomaly detection involves four main stages: automated outlier rejection **(A)**, normal heuristics collection **(B)**, automated anomaly detection **(C)**, and system notification and integration **(D)** with other security tools. We introduce a framework that summarizes relations among different stages of the development of our analysis and detection tool.

C. RESEARCH CHALLENGE

Over the last ten years the use of cryptographic techniques to protect enterprise data has grown, resulting in increased need for EKM systems. Such systems have been described in existing literature, including standards (See NIST SP800-57 [10]). Metadata analysis techniques have been widely applied in network security to build profiles of normal and anomalous behavior for use in intrusion detection [11]. However, to the best of our knowledge, there is no public research on the application of metadata obtained from EKM systems for anomaly detection. This additional EKMS metadata may be useful in enhancing anomaly detection.

A challenging aspect of this research project is determining specific EKM metadata characteristics to extract and use for anomaly detection, in a range of enterprise contexts. Factors to consider include time variant characteristics of the environment, changes in user habits, and variations in workload across different periods (time of day, day of week, time of year, public holiday etc) and context dependent variables:

number of entities making use of the system, entity role in organization, industry sector and data protection requirements.

D. RESEARCH APPROACH

Machine Learning (ML), and particularly Deep Learning (DL) solutions have been used to establish effective intrusion detection systems in large enterprise systems [12]. Auto-encoders are commonly applied in scenarios where, to develop an effective model, the process needs to learn to represent a time-series. Such an approach has previously been successfully applied to operating system log files to develop time-series-based intrusion detection using Long Short Term Memory (LSTM) [13]. Furthermore, when the underlying data is inherently unpredictable, reconstruction-based detection has been shown to be more efficient than prediction-based detection [14].

The LSTM based approach seems well-suited to modeling EKM activities. This research is performed in two stages. We first build a framework to generate EKM metadata datasets to capture normal enterprise behaviors. Then, the generated EKM metadata datasets are analyzed to establish heuristics indicative of specific enterprise activity states (normal/abnormal). Automating this process permits exploration of the datasets to determine optimal factors and weightings for a given context to form a model capable of detecting anomalous behavior.

E. RESEARCH BRIEF

This research makes use of EKM metadata to generate heuristics based on categories of behavior, to enable differentiation between normal and abnormal (possibly malicious) behavior. The heuristics from the metadata are then imported into security measures. Statistical and deep learning-based pattern recognition methods are applied for anomaly detection. We simulated various enterprise contexts and operations using an EKM system, and captured the associated metadata. We used the TSF (Trusted Security Foundation) EKM solution provided by QuintessenceLabs [15]. Simulations could have been similarly performed with another EKM system.

To enable differentiation between normal and abnormal enterprise behavior using information obtained from the EKM metadata, datasets are generated for a variety of use conditions. The generated EKM metadata datasets are analyzed to establish heuristics indicative of specific enterprise activity states (normal/abnormal). This is performed in a Behavioral Security Analysis Engine (BSAE). Automating this process permits exploration of the datasets to determine optimal factors and weightings for effective indicators of abnormal behavior.

F. RESEARCH CONTRIBUTION

This is the first article exploring the use of EKM metadata for enterprise network anomaly detection. Our contributions in developing this framework include:

1) A process to generate datasets[1] containing EKMS metadata for enterprise activities.
2) A process to identify EKMS metadata elements distinctly associated with normal and abnormal behaviors.
3) Application of LSTM auto-encoder neural networks with specific parameters for automated labeling (outlier rejection) and anomaly detection.

We provide a proof of concept by applying our framework using two case studies. We implemented a small network and included an Enterprise Key and Policy Manager solution provided by QuintessenceLabs [15]. We simulated some enterprise operations and performed EKMS metadata analysis. Our experiments clearly show that our approach identifies anomalous enterprise network activities with high accuracy.

G. ORGANIZATION OF THE ARTICLE

This article is organized as follows. Section II gives an overview of existing anomaly detection proposals. Section III introduces the network model, discusses the security objectives and assumptions and defines the capabilities of an adversary. Section IV introduces our framework for anomaly detection using EKMS metadata. Section V describes our case studies where the framework is applied. Section VI explains our experimental settings for implementation and defines the metrics used for evaluating our model. In Section VII, data generation and preparation for automated labeling and detection are discussed. Section VIII reports on our experimental results and discusses the model performance; the detection capability is clearly demonstrated. We conclude the article in Section IX.

II. EXISTING WORK ON ANOMALY DETECTION

The IEEE Standard 1044-2010 [16] defines the term "anomaly" as *"any abnormality, irregularity, inconsistency, or variance from expectations. It may be used to refer to a condition or an event, to an appearance or a behavior, to a form or a function."* Anomalies can be grouped into three major categories: point anomalies (where a single data sample is anomalous to the rest of the data), collective anomalies (where a collection of data samples occuring together appear to be anomalous although the individual instances may not be), and contextual anomalies (where the data appear to be anomalous in a specific context defined for a particular problem) [6].

Anomaly detection methods based on machine learning can be classified into two broad categories: supervised and unsupervised. Classification is based on the machine learning techniques employed and the type of data involved [6]. In supervised methods, model training requires labeled data points that clearly specify the normal instances and the abnormal instances. Then, the trained model can maximize discrimination between normal and abnormal instances. In unsupervised

[1]The open-source datasets produced and shared by this research study to provide a support, accessible via: https://baeeco-my.sharepoint.com/:f:/g/personal/mar_baee_co/ErYVmj7gYoVMg-CTAInuagsBqxzDyxi2It09GKHH5ibLDw

methods, model training does not need labels. Unsupervised training works where abnormal instances are observed to be outlier points that are distant from other instances. Unsupervised learning techniques, such as clustering, can be applied.

A detailed review and evaluation article by He et al. [17] compares six state-of-the-art log-based anomaly detection methods. These are three unsupervised methods (Principal Component Analysis (PCA) [18], Invariant Mining [19], and Log Clustering [20]) and three supervised methods (Decision Tree [21], Support Vector Machines (SVM) [22], and Logistic Regression [23]). He et al. [17] noted that supervised methods can detect anomalies in a much shorter time (less than one minute) than unsupervised methods. However, supervised methods require labeled datasets, and these are not always available, particularly in network security applications.

Recently, Bitton and Shabtai [24] proposed a network-based intrusion detection system for securing remote desktop connections at the operating system level. Their proposal utilizes machine learning for detecting malicious network packets, which may carry dangerous exploits to the remote desktop server. Their approach comprises multiple anomaly detection models such as k-means clustering and the Cluster-Based Local Outliers Factor (CBLOF) [25]. They conducted an empirical evaluation on an avionic system setup consisting of a commercial tablet connected to a real electronic flight bag server through a remote desktop connection. Their results show that the proposed method can detect malicious packets carrying known exploits. The model is shown to be accurate at anomaly detection. A very low false positive rate and a high true positive rate were obtained, outperforming current state-of-the-art algorithms. In addition, the computational complexity of their proposed model is linear with the size of the packets. Empirical analysis of the model shows that the average processing delay is not noticeable by the user. However, this proposal is based on a supervised learning approach, as the entire training dataset contained labeled data points (for both legitimate and malicious sessions).

In the last few years, deep machine learning techniques have been widely used for log anomaly detection at the operating system level due to their significant improvement in performance over classical machine learning approaches. For instance, Du et al. [26] introduce DeepLog, which first models log entries as a sequence and then trains the normal sequence with LSTM networks. DeepLog is mainly designed for collective anomaly detection but it can also detect point anomalies. It flags anomaly as “True” if there are data elements that do not conform with the normal model. Extensive experimental evaluations over large log data show that DeepLog outperformed other existing log-based anomaly detection methods based on traditional data mining methodologies. DeepLog applies a supervised approach in the training stage; all entries must come from a normal system execution path. However, normal system execution data is not always easily collectable due to large number of entries in a log file that may contain anomalies. Hence, this approach requires an additional strategy to perform normal log collection.

Vinayakumar et al. [27] review the effectiveness of LSTM network models to detect and classify the anomalous events accurately in sensor log files. Unlike DeepLog [26], their model was trained using data labeled as either normal or anomalous. The model is shown to be accurate at anomaly detection with a very low false positive rate and a high true positive rate. However, such a supervised learning approach that requires labeled datasets may not be suitable in network security applications.

Insider threat detection has attracted a considerable attention from both academia and industry. Yuan et al. [28] provide a framework to facilitate the detection of insider threats using LSTM units that are trained in a supervised manner. The LSTM units extract user behavior features from sequences of user actions and generate fixed-size feature matrices, to be classified as normal or anomalous. However, in addition to the limitations imposed by the requirement of labels, they use datasets with only 16 unique labels in model training.

Lu et al. [29] extend Yuan et al.’s work [28] and present an LSTM-based anomaly detection system called Insider Catcher. The LSTM model is trained based on employees’ online behavior to predict a user’s next possible action. As long as the prediction and the real user action do not have a significant difference, the user follows his or her normal online behavior and the action is normal. After completing the anomaly detection procedure, all records for potential threat event will be stored and passed to the insider threat analysis. These records will be compared with the user’s historical records before the identification of insider threat. However, this approach requires an additional strategy to perform normal log collection from employees’ online behavior.

Villarreal-Vasquez et al. [30] present LADOHD (LSTM-based Anomaly Detector Over High-dimensional Data), a generic LSTM-based anomaly detection framework to protect against insider threats. The training data contain the behavior of different actors collected by monitoring 30 isolated machines operated in normal situation where no attack was reported during the collection period. The authors highlight that LSTM-based models perform better compared to other models in the detection of anomalies. However, their approach requires an additional strategy to perform normal log collection.

To reduce the amount of noise and outliers in the training data, a well-designed framework to guide collection of such data in an automated manner is needed. Maleki et al. [31] propose a probability criterion based on the classical central limit theorem. This allows evaluation of the likelihood that a data point is drawn from non-anomalous data. Hence, this approach enables labeling of the data on the fly and ensures that no anomalous data is passed for training. However, their work assumes that much of the initial training data is clean, i.e., it requires that temporal datasets have been up and running for a sufficient time and that it is relatively anomaly-free.

Kennedy et al. [32], [33] present a methodology for learning without class labels. Their procedure iteratively cleans the training dataset by removing instances that have an error value

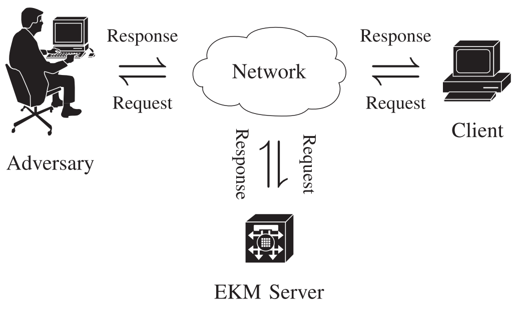

FIGURE 2. **The network model including adversary.**

above a value computed during the training steps (outliers). As the iterative process executes, the training dataset is incrementally cleaned of the minority instances, while leaving as much of the majority class as possible. The next iteration's learner is trained on a subset of data that contains a higher percentage of the majority class, its performance on unseen test data improves. However, outliers do not necessarily represent abnormal behavior. After the unsupervised outlier rejection stage, detected outliers must be verified to ensure that abnormal activities occurring with high frequency (majority) are not considered as normal. In this case, legitimate activities occurring less frequently (minority) may appear as an outlier.

The anomaly detection proposals discussed in this section were applied to log data particularly at the operating system level. To the best of our knowledge, there has been no public research into the application of metadata obtained from EKM systems for anomaly detection purposes. We fill this gap in the current article In addition, most of these proposals require labeled datasets for training. This is not a suitable approach for detecting unknown anomalous events accurately. Further, normal datasets containing non-anomalous event logs are not always easily collectable. A well-designed framework to guide collection of such data in an automated manner is needed.

To address the above-mentioned limitation, this study provides a framework that not only facilitates automated anomaly detection in EKM systems, but can be used as a general-purpose tool to improve the design of past (e.g., DeepLog [26]) and future proposals in anomaly detection.

LSTM seems to be one of the most prominent methods for time-series data modeling and capturing long-range temporal dependencies across time steps [27]. As a proof of concept for our framework, in this study the LSTM networks [34] are used as the auto-encoder.

III. PROBLEM STATEMENT

This section outlines the network model and assumptions, the adversary's capabilities, and the design objectives.

A. NETWORK MODEL AND ASSUMPTIONS

The EKM network model considered in this study consists of a Client, an EKM Server, and an Adversary. This EKM architecture and communication model is illustrated in Fig. 2. We discuss each component below.

Client: sends different cryptographic-key-related requests to the EKM Server in accordance with a predefined protocol such as the KMIP. Once the client's credentials are verified (they have been authenticated and authorized), the server provides the requested services.

EKM Server: controls the distribution, use, update, and revocation of cryptographic keys. The server may provide some additional services, such as generating authentication tags or encrypting data on behalf of the clients.

Adversary: is a hostile actor with the following assumptions:

- Their goal is to steal sensitive data.
- They have established a position in the enterprise network.
- They have credentials to access some secret keys on the EKMS.
- The data generated by Adversary is anomalous due to either:
- the use of invalid credentials to retrieve existing keys (or the previously destroyed keys) which belong to others.
- the use of valid credentials but originating from an IP address that differs from expected address.

B. ADVERSARY'S CAPABILITIES AND MOTIVATIONS

In this work, we define the capabilities of an adversary in the EKM as monitoring data exchange between the clients and server, stealing sensitive data, delaying their transmission, as well as tampering with messages and replacing the original messages with modified versions. Furthermore, a malicious adversary may deliberately generate large amounts of both legitimate and invalid messages in a relatively short period of time to tie up server resources in processing those requests, denying its services to legitimate clients. In Section V, we outline four important scenarios that specifically address this definition and clarify the security objectives.

C. PROBLEM FORMULATION

We implement a sequence-to-sequence LSTM auto-encoder model to identify anomalous activities within the EKMS network. The objective is to reconstruct the log data samples using an encoded representation of the time-series input sequences.

Such a model is capable of learning what constitutes authentic system activities by removing noise during the encoding process, without the need for attack examples. Then, once it learns the non malicious patterns of the data, anything that differentiates from it would be flagged as an anomaly.

Ideally, as long as the input is non-malicious, the ideal model will always output the same sequence that was used as input. However, the reconstructed traffic originated from malicious sequences would present a high-degree of reconstruction error, thus indicating an anomaly in the source

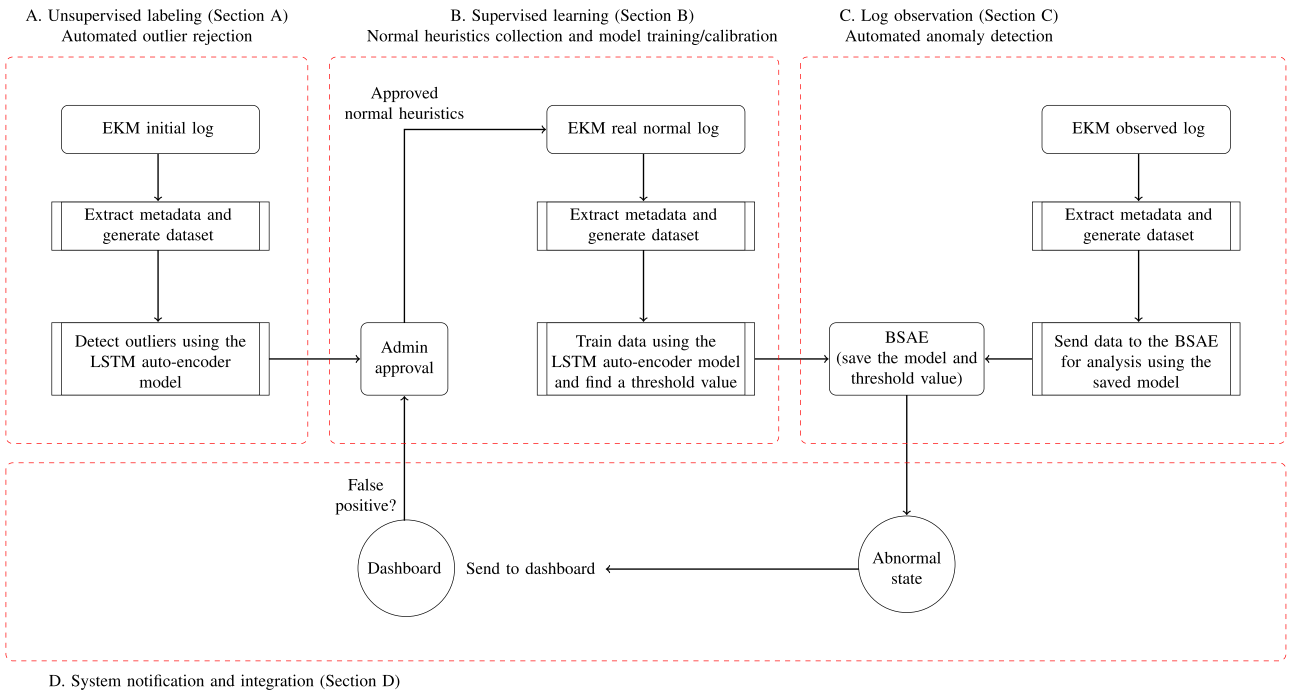
A. Unsupervised labeling (Section A)
Automated outlier rejection
B. Supervised learning (Section B)
Normal heuristics collection and model training/calibration
C. Log observation (Section C)
Automated anomaly detection
EKM initial log
Extract metadata and generate dataset
Detect outliers using the LSTM auto-encoder model
Approved normal heuristics
Admin approval
EKM real normal log
Extract metadata and generate dataset
Train data using the LSTM auto-encoder model and find a threshold value
BSAE (save the model and threshold value)
EKM observed log
Extract metadata and generate dataset
Send data to the BSAE for analysis using the saved model
False positive?
Dashboard
Send to dashboard
Abnormal state
D. System notification and integration (Section D)

FIGURE 3. Our proposed framework. The framework involves automated outlier rejection (Stage "A"), normal heuristics collection (Stage "B"), automated anomaly detection (Stage "C"), and system notification and integration (Stage "D") with detailed explanations in Sections IV-A, IV-B, IV-C, and IV-D, respectively.

sequence. A post-processing module is thus required to process this information and to generate a configurable anomaly score that might indicate an attack.

IV. THE PROPOSED FRAMEWORK

In this work, the process of metadata analysis for anomaly detection involves four main stages: automated outlier rejection (**A**), normal heuristics collection (**B**), automated anomaly detection (**C**), and system notification and integration (**D**) with other security tools. We introduce a framework that summarizes relations among different stages of the development of our analysis and detection tool. Our framework can be used to provide guidance in automating metadata analysis to use for anomaly detection.

The design and implementation of an automated metadata analysis tool is not a simple and straightforward process. Fig. 3 shows the proposed framework for anomaly detection in EKM systems. The framework uses specific ordered stages, with input to each stage defined with respect to the previous stage.

A. AUTOMATED OUTLIER REJECTION (STAGE "A")

In this stage, we capture the generated log associated with various enterprise contexts and operations within the EKM system. The log contains two general categories of network activity: normal and abnormal. Note that our final model needs to learn from the non-malicious patterns of the data; anything that differs from it will be flagged as an anomaly.

The initial log must be fed into an initial analysis model to detect any observations that are distant from the mean or location of a distribution. These observed data points are called outliers, and do not necessarily represent abnormal behavior. If the training data (final dataset) contains anomalies, our model learns to reconstruct anomalies with a minimal error, resulting in the dismissal of similar types when new data is processed.

Before outlier rejection begins, we need to generate datasets of metadata. These metadata must satisfactory answer four important questions: when (operation time), who (the user name and IP), what (the requested operation), and how (policy rules and operation success status). This is an unsupervised approach, as we prepare datasets of unknown (unlabeled) behavior associated with both categories of network activity: normal and abnormal. This dataset is then fed into our LSTM auto-encoder model (refer to Fig. 4), to detect any data points with high reconstruction loss value as outliers. Please note that each new dataset requires a new automated data labeling for outlier rejection. However, the trained model for anomaly detection can be saved for future anomaly detection purposes.

The architecture of our applied LSTM auto-encoder model for outlier rejection and anomaly detection is illustrated in Fig. 4. Given a multivariate sequence dataset $\{x^1, x^2, \ldots, x^T\}$, where $x^T \in \mathbb{R}_m$ represents the m-dimensional vector at time step T, the input data is squeezed into a single latent vector with smaller dimension than the original input.

This architecture applies LSTM neural network cells in the auto-encoder model based on the four steps of creating the

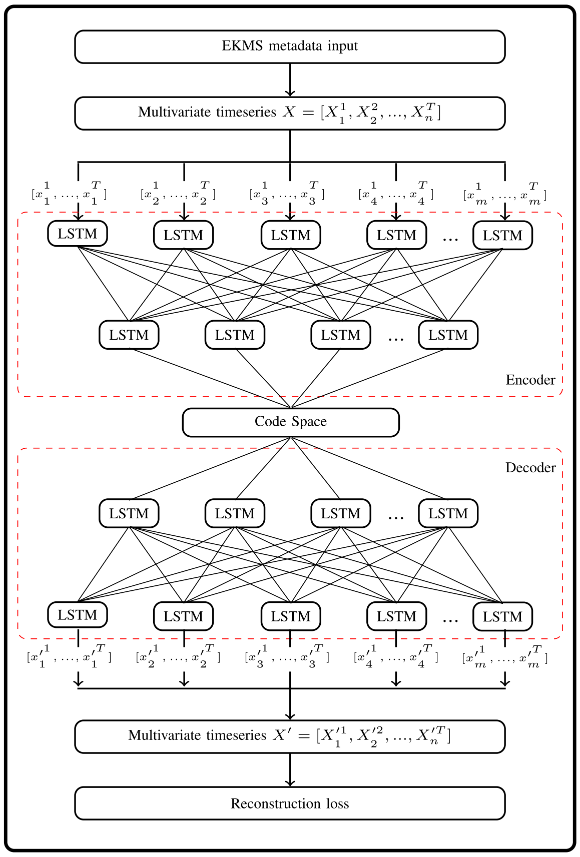
EKMS metadata input
Multivariate timeseries $X = [X_1^1, X_2^2, \ldots, X_n^T]$
$[x_1^1, \ldots, x_1^T]$
$[x_2^1, \ldots, x_2^T]$
$[x_3^1, \ldots, x_3^T]$
$[x_4^1, \ldots, x_4^T]$
$[x_m^1, \ldots, x_m^T]$
LSTM
Encoder
Code Space
Decoder
$[x'^1_1, \ldots, x'^T_1]$
$[x'^1_2, \ldots, x'^T_2]$
$[x'^1_3, \ldots, x'^T_3]$
$[x'^1_4, \ldots, x'^T_4]$
$[x'^1_m, \ldots, x'^T_m]$
Multivariate timeseries $X' = [X'^1_1, X'^2_2, \ldots, X'^T_n]$
Reconstruction loss

FIGURE 4. **The architecture of our applied LSTM auto-encoder model for outlier rejection and anomaly detection.**

input sequence, encoding, decoding, and measuring reconstruction loss for detecting anomalies. As a proof of concept, this study utilizes two LSTM hidden layers in both encoder and decoder.

In the encoding stage, the model builds a fixed-length vector that contains all of the information and time-wise relationships of the input sequence. This stage provides a compressed representation of the input data. Then, a repeat vector layer (code space) distributes the compressed representational vector across the time steps of the decoder. In the decoding stage, the model expands the compressed vector. The goal is to create an output that is as close as possible to the original input.

B. NORMAL HEURISTICS COLLECTION, MODEL TRAINING, AND CALIBRATION (STAGE "B")

After the unsupervised outlier rejection stage, detected outliers must be verified by a system administrator. This ensures that abnormal activities occurring with high frequency are not considered as normal. In this case, legitimate activities occurring less frequently may appear as an outlier in an automated process. Hence, a system administrator must perform this initial check manually until sufficient data history is acquired. Once sufficient log entries associated with normal activities have been collected, the manual administrator role will be reduced significantly, and most of the outlier rejection can be automated.

The aim is generating datasets of metadata associated with normal behavior. Similar to the previous stage, we generate datasets of metadata from which four questions are answered: when (operation time), who (the user name and IP), what (the requested operation), and how (policy rules and operation success status). This is a supervised approach, as we prepared datasets of normal behavior, but these are unlabeled. That is, there is no one-to-one correspondence between samples and the associated input features, and their corresponding output labels.

We use this dataset to train our model which is a combination of LSTM and auto-encoder. This permits determination of optimal factors and weightings to establish heuristics for effective detection of anomalies based on the analysis of time-series data.

Once our model (refer to Fig. 4) is trained, a suitable threshold value for identifying anomalies is determined. The distribution of the calculated loss in the training set can help us to find such a threshold. To calibrate our model, we can set a threshold above the "loss level" so that false positives are not triggered. A false positive corresponds to identification of an anomalous event, when the event is actually legitimate.

C. AUTOMATED ANOMALY DETECTION (STAGE "C")

In this stage, the trained model with a calibrated threshold value is saved into the BSAE. For example, one can save the trained model and its learned weights in the ".h5" format. This can be deployed for future anomaly detection. Using an established (saved) learning process rather than performing a new one permits us to save time in future applications.

When new EKM activities are observed, the associated metadata is extracted into a dataset and sent to the BSAE for differentiation between normal and abnormal enterprise behavior. Note that the observed log may contain both "normal" data (within the norm) and "abnormal" data corresponding to anomalies (exceptions to the norm). We aim to detect anything that deviates from the normal pattern "norm". This includes, for example, those data points which have not been seen before by machine.

D. SYSTEM NOTIFICATION AND INTEGRATION (STAGE "D")

The detected anomalous EKMS behavior should be integrated with existing Security Information and Event Management (SIEM) products to sharpen anomaly detection capabilities. For example, requests for key material from processes associated with staff accounts for staff known to be on leave can be indicators of potential enterprise system compromise. In such a case, a notification is sent to the SIEM dashboard to alert the administrator. If "Abnormal State" was falsely called, then the system administrator amends that specific data point into the datasets of normal behavior to prevent these false positive flags in future.

The corrected dataset of normal behavior can be fed into our model on a regular basis. For example, each night (at a fixed time) the model may undergo regular learning, with the old model values being replaced with new values, within the BSAE.

V. CASE STUDIES

This section defines four scenarios across two case studies that we use to demonstrate the effectiveness of our proposed framework.

A. CASE STUDY 1

This case study focuses on scenarios in which a change in normal requests is detected by the BSAE. Below, we explain two such scenarios and the associated system symptoms.

Scenario 1: A trusted system user, who has credentials to access some keys on the EKMS, turns rogue and decides to steal sensitive research data before decamping to another company. The user is aware that the company monitors activity on their workstations and does not want to be too obvious about their activities. They plan to download and exfiltrate the data using a colleague's computer.

Symptom: Valid "Get" request for specific Key IDs originating from an address (e.g., IP or MAC) that doesn't usually request those keys, or valid requests using credentials of employees not currently in the building.

Scenario 2: A hostile actor has managed to obtain an encryption key used to protect network traffic on a link encryptor, and is decrypting and monitoring that traffic. The actor is aware that the organization implements date-based key rotation, and does not want the supply of intelligence to dry up. Using a credential stolen from an administrative work station, the actor sends a request to the EKMS server to modify the "Deactivation Date" and "Protect Stop Date" attributes of the currently used key, ensuring automated key rotation is delayed.

Symptom: Valid "Modify" requests which are not part of the usual EKM activities.

B. CASE STUDY 2

This case study focuses on scenarios in which a spike of requests is received by the EKMS. Below, we explain two such scenarios and their associated system symptoms.

Scenario 3: A hostile actor has managed to log in to the computer of a trusted employee. The credentials needed to retrieve encryption keys are saved on this computer. The hostile actor retrieves the keys required to locally decrypt data stored in a remote database for local processing. The actor starts bulk downloading and decrypting the data, intending to exfiltrate it and sell it on the black market.

Symptom: The frequency of "Get-key" requests for this particular user spikes; for example, going from one request per minute to one request per second.

Scenario 4: A hostile actor has exploited a vulnerability in a web-based management screen and gained administrative access to the corporate EKMS. They have the ability to create new credentials, and they become aware that the EKMS is a virtual system and lacks a high-volume source of entropy. The actor attempts to exhaust the entropy pool of the server during a small time window, with the goal that keys created during this window will be weak (easily guessed). Using their administrative access, they create multiple new EKMS clients to avoid large spikes in the traffic of any particular user, or existing users.

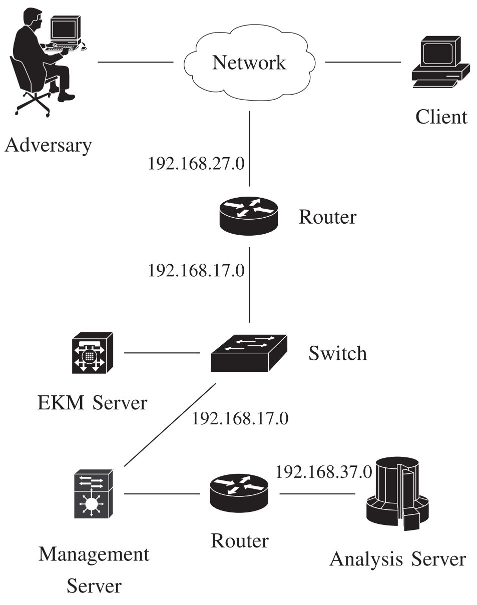

FIGURE 5. The network topology, actors, and nodes.

Symptom: A spike in new user credentials being created/observed in use; a spike in "Create-Key/Key-Pair" requests, aggregated across all requests from all users.

VI. IMPLEMENTATION AND EVALUATION

In this section, we explain the experimental settings applied in our implementation, and define the metrics used for evaluating our model.

A. EXPERIMENTAL SETTINGS

We extend the network model shown in Fig. 2 to a model which consists of a Client, an EKM Server, an Adversary, a Management Server, an Analysis Server, and the communication hardware between these entities. This enables the collected metadata to be used in the subsequent analysis phase. We deploy an environment to simulate this extended network model. Fig. 5 shows our implemented network topology, actors, and nodes. There are three networks, namely:

- network 192.168.27.0 that includes a client and an adversary,
- network 192.168.17.0 that includes an EKM Server and a Management Server, and
- network 192.168.37.0 that includes an Analysis Server.

The Management Server pulls the log files from the EKM Server and pushes them to the isolated Analysis Server, which

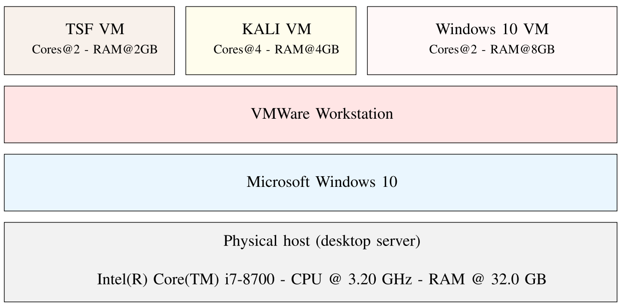

FIGURE 6. The simulation environment configuration.

is responsible for parsing log files, training our model, and automating the outlier/anomaly detection. The result of analysis is then pushed into the Management Server for notification and integration.

We have configured our experimental environment as illustrated in Fig. 6. Three Virtual Machines (VMs) run on a VMWare Workstation. These include a TSF VM (the EKM solution provided by QuintessenceLabs), a KALI Linux VM (the Analysis Server), and a Windows 10 VM (the Management Server). These are real-world commercial software products. The VMWare Workstation runs on a Windows 10 operating system, hosted by an Intel Core Desktop Machine.

Our Deep Learning model implementation makes use of Python 2.7 and libraries including: Numpy 1.16.1, Scipy 1.2.3, Scikit_learn 0.20.4, Theano 1.0.5, TensorFlow 2.1.0, Keras 2.7.0, Pandas 0.24.2, and Matplotlib 2.2.5 installed on a KALI GNU/Linux 2021.2.

B. EVALUATION METRICS

This section defines the metrics used in our evaluation. These are precision, recall, $F1$, and accuracy.

Precision is the ratio of the number of true positives to the sum of true and false positives. Recall focuses on measuring the ability of the methods to detect anomalous events. We can calculate precision by:

$$precision = \frac{TP}{TP + FP} \quad (1)$$

and calculate recall by:

$$recall = \frac{TP}{TP + FN}, \quad (2)$$

and use (1) and (2) to calculate $F1$:

$$F1 = \frac{2 \times precision \times recall}{precision + recall}, \quad (3)$$

where TP is true positives, FP is false positives, and FN is false negatives. The accuracy score is calculated by dividing the number of correct predictions by the total prediction number.

VII. DATA GENERATION AND PREPARATION

As the objective of the project is to be able to differentiate between normal and abnormal (possibly malicious) enterprise

Listing 2: Data preparation for Case study 1

```
Metadata:
01,104,DiscoverVersions,192168272,client,policy,Success,320,
Locate,192168272,client,owner,0,0,0,Success,Activate,192168272,
client,0,UIDexists,owner,ActivatefromPre-ActivetoActive,0,Success

Hashed value of Metadata:
94412611,409347,13759287,29137660,46053976,49786144,31170436,
10248116,65443403,29137660,46053976,90956564,41564428,41564428,
41564428,31170436,76943746,29137660,46053976,41564428,57841413,
90956564,49349486,41564428,31170436
```

behavior using information obtained from the EKM metadata, data must be generated for a variety of use conditions. For the set of predefined scenarios outlined in Section V, we have generated the Client to Server requests, including: Get-key, Create-key, Activate-key, Add-attribute, Modify-attribute, Revoke-key, and Destroy-key, for both normal and adversarial activities. Through the Management Server, we pulled the collected metadata on the EKM Server and pushed to the Analysis Server for processing.

We parsed [35] the EKMS metadata and generated datasets for training and testing. This includes the removal of characters that are common across all files (e.g., “.”, “_”, and blank characters) and unnecessary words (e.g., “byte”). Also, empty cells are replaced with “0”. The Analysis Server is to perform the tasks discussed in this stage.

VIII. PROCESS, RESULTS, DISCUSSION, AND COMPARISON

In this section, we first report on the process and our experimental results for the two case studies, in line with the four stages of our framework: automated outlier rejection (**A**), normal heuristics collection (**B**), automated anomaly detection (**C**), and system notification and integration (**D**). Then, we discuss our framework and model performance, and compare it to the other approaches published in literature.

A. PROCESS AND RESULTS

This section reports on the process and our experimental results.

Stage “A” (Identical for Case Studies 1 and 2): We capture the EKMS metadata and generate a dataset associated with various enterprise contexts and operations on the EKM Server (as outlined in Section VII). The initial dataset in Stage “A” of our framework contains 1526 data points, representing mostly normal activities, with some outliers which need to be removed.

Each parsed and normalized data cell must be converted into an 8-digit hash value. This is to ensure that any change in the cell information (before conversion) makes a huge change after conversion, which helps our model to output a large reconstruction loss for values with lower occurrence. Listing 2 shows an example of EKMS metadata and the hash value associated with an “Activate-key” request from a user.

The initial dataset is then fed into our LSTM auto-encoder model (refer to Fig. 4). Stage “A” utilizes two LSTM hidden layers in both encoder and decoder, with 40 and 10 LSTM

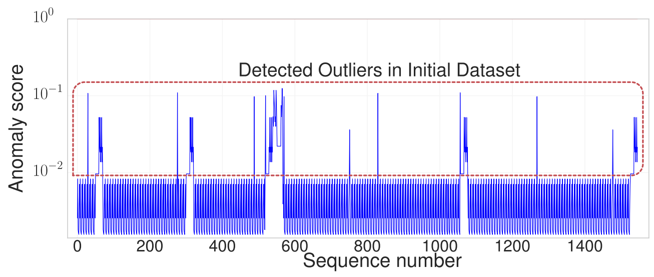

FIGURE 7. **Automated outlier rejection in Stage “A”.**

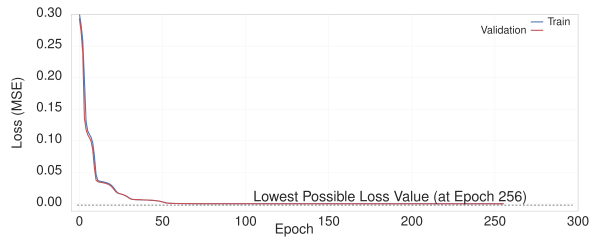

FIGURE 8. **The training losses.**

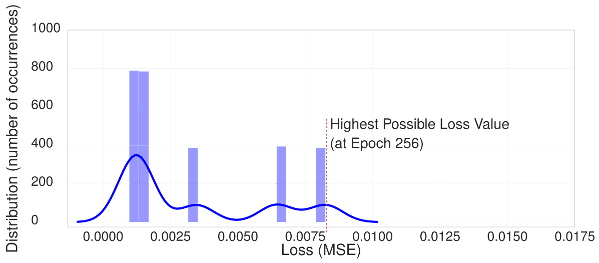

FIGURE 9. **Distribution of calculated loss in Training set.**

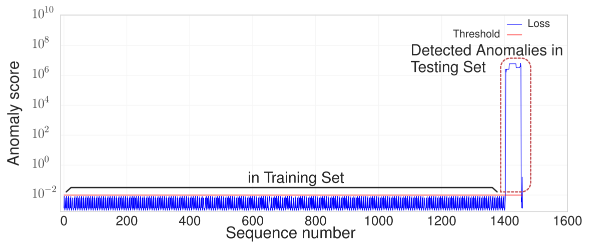

FIGURE 10. **The results of anomaly detection over time.**

units, respectively. Note that our model incorporates the Rectified Linear Activation Unit or “ReLU” which is the most common choice of activation function in multi-layer neural networks or deep neural networks. We set “Adam” as our neural network optimizer algorithm. Adam is a learning rate optimization algorithm for training deep learning models, designed to accelerate the optimization process. This decreases the number of function evaluations required to reach good results. We set Mean Squared Error (MSE) for calculating our loss function. For regression tasks, MSE is a popular choice of loss function [13]. Compared to the Mean Absolute Error (MAE), MSE is more sensitive to outliers. The number of epochs and batch size are 256 and 64, respectively.

Fig. 7 shows the detected outliers in Stage “A” using our LSTM auto-encoder model. These minority instances (within the highlighted rectangle) are distant from the mean or location of the distribution, with a reconstruction loss value greater that 10^{-2}. These observed outliers represent abnormal behavior. If the final training dataset contains anomalies, our model learns to reconstruct anomalies with a minimal error, resulting in the dismissal of similar types when new data is processed. Hence, this stage is necessary, to identify only those EKM metadata elements distinctly associated with normal behaviors.

After removing these outliers, we have a dataset of behaviors under normal operating conditions. This dataset enables us to proceed to the Stage “B” in which the trained model for anomaly detection can be saved for future anomaly detection purposes. Please note that a newly generated dataset in Stage “A” of our framework requires a new outlier rejection.

Stages “B”, “C”, and “D” (for Case Study 1): The dataset used in Scenarios 1 and 2 contains 1404 data points, representing normal operating conditions. This is achieved after passing the “Stage A” of our framework. The Testing dataset contains 54 data points, which represents a mix of normal (4 samples) and abnormal (50 samples) operating conditions.

Each parsed and normalized data cell is converted into an 8-digit hash value. This is to ensure that any change in the cell information (before conversion) makes a huge change after conversion, which helps our trained model to output a large reconstruction loss value for any new input data that has never seen before.

Stage “B” in Case Study 1 utilizes two LSTM hidden layers in both encoder and decoder, with 40 and 10 LSTM units, respectively. Our model incorporates the “ReLU” activation function, the “Adam” neural network optimizer algorithm, and MSE for calculating our loss function.

The dataset is split for training and validation. We instantiate and train our model on 1333 samples, and validate our model on 71 samples, where the number of epochs and batch size are 256 and 64, respectively.

Fig. 8 shows the training losses, and is used for evaluating our model’s performance. We need to determine a suitable threshold value for identifying an anomaly. The distribution of the calculated loss in the Training set can help us to find such a threshold (refer to Fig. 9). Hence, we can set a threshold above the “loss level” so that false positives are not triggered.

From the above loss distribution (Fig. 9), we see that the maximum loss value is ≈ 0.01. Therefore, we set an anomaly threshold value of 0.01, a threshold above the “loss level” so that false positives are not triggered. This flags an anomaly when the reconstruction loss in our Test set is greater than 0.01. Fig. 10 visualizes the results over time, where the red line indicates our threshold value of 0.01.

Our LSTM auto-encoder model for anomaly detection is able to flag the anomalous activities within an observed log (Stage “C”), with 100% accuracy. Listing 3 shows four rows

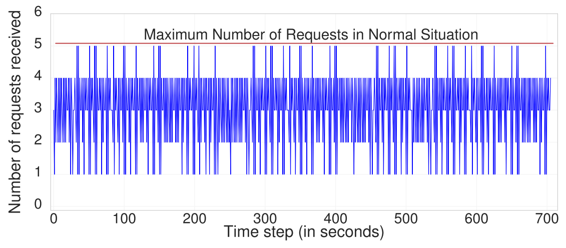

FIGURE 11. The number of requests received in each one second (Training set).

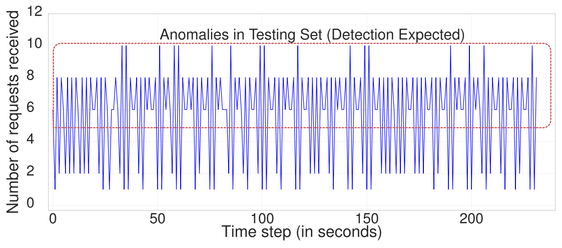

FIGURE 12. The number of requests received in each one second (Testing set).

Listing 3: Four samples along with their loss values

```
02,104,DiscoverVersions,192168275,client,policy,Success,320,
Locate,192168275,client,owner,Success,Get,192168275,client,
UIDexists,owner,Success

01,104,DiscoverVersions,192168272,Insider,policy,Success,320,
Locate,192168272,Insider,owner,0,0,0,Success,Get,192168272,
Insider,0,UIDexists,owner,0,Success,0

04,104,DiscoverVersions,192168272,client,policy,Success,2240,
ModifyAttribute,192168272,client,UIDexists,owner,Success

01,104,DiscoverVersions,192168272,client,policy,Success,320,
Locate,192168272,client,owner,0,0,0,Success,Get,192168272,client,
0,UIDexists,owner,0,Success,0

Loss (MSE)              Threshold       Anomaly
2552176.9780013524      0.01            True
1657971.17256199        0.01            True
3218440.9133525514      0.01            True
0.0013321462355466      0.01            False
```

in the final report generated for Testing set in Stage "D". Observing these four rows, three rows contain anomalous events (flagged as True) where:

1) the request from user name "client" does not usually come from IP address "192.168.27.5" at "02" o'clock,
2) the request from IP address "192.168.27.2" does not usually come from user name "Insider", and
3) 'the 'Modify Attribute" request is not part of the usual EKM activities.

Hence, the loss value for each of the above three items goes over the Threshold value 0.01. However, the last request in the list is a part of usual EKMS activities, and as a result, the loss value for this item goes under the Threshold value 0.01.

Stages "B", "C", and "D" (for Case Study 2): The dataset used in Scenarios 3 and 4 contains 2149 data points, representing normal operating conditions within 706 seconds. This is achieved after passing the "Stage A" of our framework. The Testing dataset contains 1309 data points within 232 seconds, which represents a mix of 70 normal seconds (105 data points) and 162 abnormal seconds (1204 data points). In each one second, a different number of requests (data points) was recorded. For instance, at "01:06:08" on "2021.09.05" four requests were received. This format for recording the number of requests per second is used for both training and testing.

Fig. 11 shows the frequency of requests received in each one second in our Training dataset. The highest number of requests received per second is 5. Therefore, we consider the upper bound to be 5 requests per second.

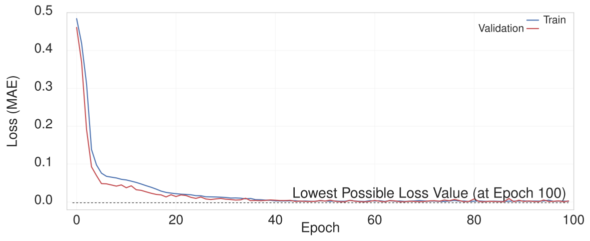

FIGURE 13. The training losses.

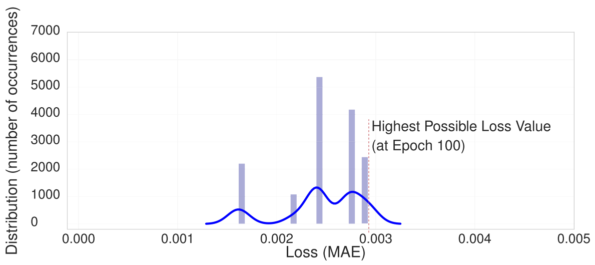

FIGURE 14. Distribution of calculated loss in Training set.

Fig. 12 show the frequency of requests received in each one second in our Testing dataset. The highest number of requests received per second is 10. Therefore, we expect our automated model to detect any one-second time period in which the number of received requests is higher than our upper bound that is 5 requests per second.

Stage "B" in Case Study 2 utilizes two LSTM hidden layers in both encoder and decoder, with 16 and 4 LSTM units, respectively. Our model incorporates the "ReLU" activation function, the "Adam" neural network optimizer algorithm, and MAE for calculating our loss function. Please note that both MSE and MAE can be used in this Case Study. As we reduced the number of features to only the number of requests per seconds, MAE can produce enough sensitivity to outliers.

The dataset is split for training and validation. Then, we instantiate and train our model on 670 samples, and validate our model on 36 samples, where the number of epochs and batch size are 100 and 10, respectively.

Fig. 13 shows the training losses, and is used for evaluating our model's performance. We need to determine a suitable threshold value for identifying an anomaly. The distribution of the calculated loss in the Training set can help us to find such a threshold (refer to Fig. 14). Hence, we can set

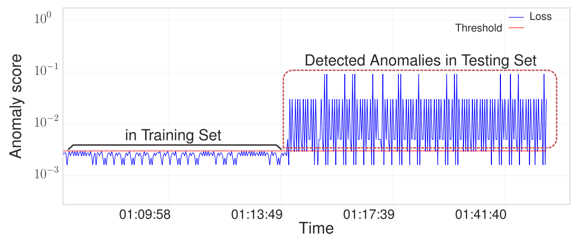

FIGURE 15. The results of anomaly detection over time.

Listing 4: Four samples along with their loss values

```
2021-09-04 01:40:06,6
2021-09-04 01:40:07,1
2021-09-04 01:40:08,8
2021-09-04 01:40:09,2

Loss (MAE)              Threshold       Anomaly
0.0049970149993896484   0.003           True
0.0016150027513504028   0.003           False
0.029864788055419922    0.003           True
0.002921879291534424    0.003           False
```

a threshold above the "loss level" so that false positives are not triggered.

From the above loss distribution (Fig. 14), we see that the maximum loss value is ≈ 0.003. Therefore, we set an anomaly threshold value of 0.003, a threshold above the "loss level" so that false positives are not triggered. This flags an anomaly when the reconstruction loss in our Test set is greater than 0.003. Fig. 15 visualizes the results over time, where the red line indicates our threshold value of 0.003.

Our LSTM auto-encoder model for anomaly detection is able to flag the anomalous activities within an observed log (Stage "C"), with 100% accuracy. Listing 4 shows four rows in the final report generated for Testing set in Stage "D". Observing these four rows, two rows contain anomalous events (flagged as True) where the number of requests per second goes over 5 which means the loss value goes over the Threshold value 0.003.

B. DISCUSSION

According to our investigations, the required data for training to reach the highest level of accuracy can be reduced. In Case Study 1, we used 1404 samples under "normal" condition. We can get equivalent accuracy with only 400 samples without changing the Threshold. However, for further reductions (from 400 to 100 samples) to retain this accuracy we needed to increase the Threshold value to ≈ 0.1.

While we did not use a GPU with TensorRT, the processing time for the model was short. The recorded training time in Case Studies 1 and 2 are ≈ 17 and ≈ 18 seconds, respectively. Also, the saved model could perform detection in a dataset of 1500 samples in ≈ 20 milliseconds.

In some scenarios there may be a level of false-positive detection. For example, if new staff come on or take on new responsibilities, "normal" access patterns change. This change in pattern could be detected as abnormal which is important in resilient systems where the system must continue to carry out its mission in the face of adversity. This issue can be solved by excluding the new staff member from the datasets during data preparation. The new staff member's activity can be separately monitored and logged (e.g., for one week) before its amendment into the main dataset of normal behavior.

C. COMPARISON

Table 1 presents a summary of the focus and details found in the prior work [26], [27], [28], [29], [30], [31], [32], [33] discussed in Section II, for comparison purposes with our research.

Compared to the other approaches published in literature (e.g., [26] and [27]), our framework guarantees 0% false negative detection. Also, in both Case Studies, the results prove that our framework can perform automated detection with 100% accuracy.

The outlier rejection method presented by Maleki et al. [31] applies a probabilistic approach that assigns a probability to the data points; a probability criterion which evaluates the likelihood that a data point deviates from the normal pattern, addressing the outlier rejection problem. The main limitations of this approach is availability and reliability of the historical data. It is assumed that the underlying system has been working for some time, and sufficient data has been observed. Additionally, it is also assumed that the underlying system is correctly designed so that an abnormal operation's probability is considerably lower. The approach presented in this study does not introduce such a limitation. Our proposed outlier rejection in "Stage A" generates datasets of normal behavior without the need for reliability of the historical data. This is illustrated in Fig. 7.

The detection method presented by Maleki et al. [31] fails to produce a large reconstruction error against minor changes in a given EKM observed log (in "Stage C"). For example, the requests from an unseen IP address from a known user is not detected. In our proposed "Stage A" (for both Case Studies) and "Stage B" (for Case Study 1), each parsed and normalized data cell must be converted into an 8-digit hash value. This is to ensure that any change in the cell information (before conversion) makes a huge change after conversion, which helps our model to output a large reconstruction loss for values with lower occurrence or minor change (e.g., change in IP address or arrived packet size). This is outlined in Listing 3.

Table 2 shows the performance comparison of our work to the detection method presented by Maleki et al. [31] which is an LSTM auto-encoder based solution. This comparison is based on different datasets. A detailed performance comparison based on EKM metadata is provided in Table 3 which demonstrates the superiority of our approach where minor changes occur in the IP address and packet size information.

IX. CONCLUSION AND FUTURE DIRECTION

This article presented a framework for anomaly detection based on EKMS metadata. To the best of our knowledge, this is the first article exploring the use of EKM metadata for

TABLE 1. Comparison With Existing Frameworks

Research	Properties			
	Technique	Purpose	Unsupervised Dataset Labeling	Strong Outlier Rejection
Du et al. [26]	LSTM	anomaly detection/enterprise system	×	×
Vinayakumar et al. [27]	LSTM	anomaly detection/sensor network	×	×
Yuan et al. [28]	LSTM	anomaly detection/enterprise system	×	×
Lu et al. [29]	LSTM	anomaly detection/enterprise system	×	×
Villarreal-Vasquez et al. [30]	LSTM	anomaly detection/commercial network	×	×
Maleki et al. [31]	LSTM auto-encoder	anomaly detection/abrupt change/incident	✓	×
Kennedy et al. [32]	auto-encoder	fraud detection/credit card system	✓	×
Kennedy et al. [33]	auto-encoder	anomaly detection/cognition system	✓	×
Baee et al. [this work]	LSTM auto-encoder	anomaly detection/key-management system	✓	✓

–Note: The property is satisfied [✓]. The property is not satisfied [×].

TABLE 2. Comparison With Existing LSTM Auto-Encoder - Different Datasets

Method	Results			
	Precision	Recall	F1	Dataset
Maleki et al. [31]	0.90	1.0	0.95	AWS^1
Maleki et al. [31]	0.99	0.95	0.97	Siemens^2
This work	1.0	1.0	1.0	QLabs EKMS^3

1- The Amazon Web Services (AWS) monitoring CPU usage data.
2- The Siemens Industrial Turbomachinery real-time data.
3- The QuintessenceLabs (QLabs) EKM solution data (refer to Note 1).

TABLE 3. Comparison With Existing LSTM Auto-Encoder - EKM Dataset (Refer to Note 1)

Method	Results on QLabs^1 Dataset When Minor Change Occurs in IP Address and Packet Size		
	Outlier Rejection	False Negative Rate	Accuracy at the Event
Maleki et al. [31]	×	100%	0%
This work	✓	0%	100%

1- The QuintessenceLabs (QLabs) EKM solution data (refer to Note 1).
Note: The property is satisfied [✓]. The property is not satisfied [×].

enterprise network anomaly detection. We demonstrated that our approach is successful.

The framework involves four stages: automated outlier rejection, normal heuristics collection, automated anomaly detection, and system notification and integration with other security tools. This framework is developed through investigation of EKMS metadata, determining characteristics to extract for dataset generation, and looking for possible patterns from which behaviors can be inferred. For automated labeling and detection, a deep learning-based model was applied. Our framework not only facilitates automated anomaly detection in EKM systems, but also can be used as a general-purpose framework to help improving the design of past and future proposals in anomaly detection. The framework should perform in exactly the same way if hardware modules are used.

As noted in Section I, the client requests can be encoded in a KMIP format and transmitted to the EKM server. The server decodes the request using a KMIP decoder to form an intermediate representation, which is used by the server API to process the request. However, to the best of our knowledge, there have been no research efforts that explore the use of KMIP metadata for enterprise network anomaly detection. This is a potential avenue for future research.

ACKNOWLEDGMENT

All authors are participants in the Cyber Security Cooperative Research Center.

REFERENCES

[1] D. Sitaram, S. Harwalkar, U. Simha, S. Iyer, and S. Jha, "Standards based integration of advanced key management capabilities with openstack," in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets*, 2015, pp. 98–103.
[2] M. A. R. Baee, "Privacy-preserving authentication and key management for cooperative intelligent transportation systems," Ph.D. thesis, Queensland Univ. of Technol., Brisbane City, QLD, Australia, 2021.
[3] M. A. Rezazadeh Baee, L. Simpson, X. Boyen, E. Foo, and J. Pieprzyk, "ALI: Anonymous lightweight inter-vehicle broadcast authentication with encryption," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 3, pp. 1799–1817, May/Jun. 2023.
[4] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot, *Handbook of Applied Cryptography*, 1st ed. Boca Raton, FL, USA: CRC Press, 1996.
[5] M. A. R. Baee, L. Simpson, X. Boyen, E. Foo, and J. Pieprzyk, "A provably secure and efficient cryptographic-key update protocol for connected vehicles," *IEEE Trans. Dependable Secure Comput.*, early access, Dec. 21, 2023, doi: 10.1109/TDSC.2023.3345406.
[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009.
[7] T. Cox and C. White"OASIS Key management interoperability protocol specification version 2.0," OASIS Standard, Oct. 2019. [Online]. Available: https://docs.oasis-open.org/kmip/kmip-spec/v2.0/kmip-spec-v2.0.html
[8] J. Diederich and J. Milton, "Creating domain specific metadata for scientific data and knowledge bases," *IEEE Trans. Knowl. Data Eng.*, vol. 3, no. 4, pp. 421–434, Dec. 1991.
[9] R. Gerhards, "Rsyslog: Going up from 40K messages per second to 250K," in *Proc. Conf. Linux Kongress*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:204845665

[10] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, “NIST special publication 800-57,” *NIST Special Publication*, vol. 800, no. 57, pp. 1–142, 2007.
[11] F. Skopik, M. Landauer, and M. Wurzenberger, “Online log data analysis with efficient machine learning: A review,” *IEEE Secur. Privacy*, vol. 20, no. 3, pp. 80–90, May/Jun. 2022.
[12] F. Skopik, M. Landauer, and M. Wurzenberger, “Online log data analysis with efficient machine learning: A review,” *IEEE Secur. Privacy*, vol. 20, no. 3, pp. 80–90, May/Jun. 2022.
[13] W. G. Hatcher and W. Yu, “A survey of deep learning: Platforms, applications and emerging research trends,” *IEEE Access*, vol. 6, pp. 24411–24432, 2018.
[14] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, “LSTM-based encoder-decoder for multi-sensor anomaly detection,” *CoRR*, vol. abs/1607.00148, 2016. [Online]. Available: http://arxiv.org/abs/1607.00148
[15] “QuintessenceLabs; Trusted security foundation enterprise key and policy manager,” Accessed on: May 03, 2023. [Online]. Available: https://www.quintessencelabs.com
[16] *IEEE Standard Classification for Software Anomalies*, IEEE Standard 1044-2009 (Revision of IEEE Standard 1044–1993), 2010, pp. 1–23.
[17] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience report: System log analysis for anomaly detection,” in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng.*, 2016, pp. 207–218.
[18] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, “Detecting large-scale system problems by mining console logs,” in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, 2009, pp. 117–132.
[19] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, “Mining invariants from console logs for system problem detection,” in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2010, Art. no. 24.
[20] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, “Log clustering based problem identification for online service systems,” in *Proc. 38th Int. Conf. Softw. Eng. Companion*, 2016, pp. 102–111.
[21] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, “Failure diagnosis using decision trees,” in *Proc. Int. Conf. Autonomic Comput.*, 2004, pp. 36–43.
[22] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, “Failure prediction in IBM BlueGene/L event logs,” in *Proc. 7th IEEE Int. Conf. Data Mining*, 2007, pp. 583–588.
[23] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, “Fingerprinting the datacenter: Automated classification of performance crises,” in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 111–124.
[24] R. Bitton and A. Shabtai, “A machine learning-based intrusion detection system for securing remote desktop connections to electronic flight bag servers,” *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 3, pp. 1164–1181, May/Jun. 2021.
[25] Z. He, X. Xu, and S. Deng, “Discovering cluster-based local outliers,” *Pattern Recognit. Lett.*, vol. 24, no. 9, pp. 1641–1650, 2003.
[26] M. Du, F. Li, G. Zheng, and V. Srikumar, “DeepLog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2017, pp. 1285–1298.
[27] R. Vinayakumar, K. P. Soman, and P. Poornachandran, “Long short-term memory based operation log anomaly detection,” in *Proc. IEEE Int. Conf. Adv. Comput., Commun. Inform.*, 2017, pp. 236–242.
[28] F. Yuan et al., “Insider threat detection with deep neural network,” in *Proc. Int. Conf. Comput. Sci.*, 2018, pp. 43–54.
[29] J. Lu and R. K. Wong, “Insider threat detection with long short-term memory,” in *Proc. Australas. Comput. Sci. Week Multimedia Conf.*, 2019, pp. 1–10.
[30] M. Villarreal-Vasquez, G. Modelo-Howard, S. Dube, and B. Bhargava, “Hunting for insider threats using LSTM-based anomaly detection,” *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 451–462, Jan./Feb. 2023.
[31] S. Maleki, S. Maleki, and N. R. Jennings, “Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering,” *Appl. Soft Comput.*, vol. 108, 2021, Art. no. 107443.
[32] R. K. L. Kennedy, Z. Salekshahrezaee, and T. M. Khoshgoftaar, “A novel approach for unsupervised learning of highly-imbalanced data,” in *Proc. IEEE 4th Int. Conf. Cogn. Mach. Intell.*, 2022, pp. 52–58.
[33] R. K. Kennedy, Z. Salekshahrezaee, and T. M. Khoshgoftaar, “Unsupervised anomaly detection of class imbalanced cognition data using an iterative cleaning method,” in *Proc. IEEE 24th Int. Conf. Inf. Reuse Integration Data Sci.*, 2023, pp. 303–308.
[34] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[35] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “An evaluation study on log parsing and its use in log mining,” in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2016, pp. 654–661.