Received October 31, 2019, accepted November 24, 2019, date of publication December 2, 2019,
date of current version December 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2957055

Insider Threat Identification Using the Simultaneous Neural Learning of Multi-Source Logs

LIU LIU[1], **CHAO CHEN**[1], **JUN ZHANG**[1], **OLIVIER DE VEL**[2], **AND YANG XIANG**[1]
[1]School of Software and Electrical Engineering, Swinburne University of Technology, Hawthorn, VIC 3122, Australia
[2]Defence Science and Technology Group, Department of Defence, Edinburgh, SA 5111, Australia

Corresponding author: Liu Liu (liuliu@swin.edu.au)

This work was supported by the Australia Defence Science and Technology (DST) Group.

ABSTRACT Insider threat detection has drawn increasing attention in recent years. In order to capture a malicious insider's digital footprints that occur scatteredly across a wide range of audit data sources over a long period of time, existing approaches often leverage a scoring mechanism to orchestrate alerts generated from multiple sub-detectors, or require domain knowledge-based feature engineering to conduct a one-off analysis across multiple types of data. These approaches result in a high deployment complexity and incur additional costs for engaging security experts. In this paper, we present a novel approach that works with a variety of security logs. The security logs are transformed into texts in the same format and then arranged as a corpus. Using the model trained by Word2vec with the corpus, we are enabled to approximate the posterior probabilities for insider behaviours. Accordingly, we label the transformed events as suspicious if their behavioural probabilities are smaller than a given threshold, and a user is labelled as malicious if he/she is associated with multiple suspicious events. The experiments are undertaken with the Carnegie Mellon University (CMU) CERT Programs insider threat database v6.2, which not only demonstrate that the proposed approach is effective and scalable in practical applications but also provide a guidance for tuning the parameters and thresholds.

INDEX TERMS Cybersecurity, data analytics, insider threats, word embedding.

I. INTRODUCTION

Malicious insiders have been recognised as the most critical security threat to an organisation [1], [2]. As reported in the Clearswift Insider Threat Index (CITI) annual report 2017 [3], 92% of the organisations suffered IT or data security incidents in the past 12 months, where 74% of the incidents resulted from insiders. Furthermore, as insiders are privileged to access an organisation's network and data, they can easily evade ordinary detection mechanisms, leading to greater damages and huge financial losses for victim organisations [4]. Thus, it is imperative to develop new approaches for detecting malicious insiders.

Insider threat detection has been difficult due to several challenges [2]. Firstly, as an organisation's security mechanisms are not primarily aimed at people who reside inside the network, this presents an opportunity for a motivated insider with privileged access to take advantage of, undertaking malicious actions without triggering alerts. Besides, such subtle actions allow an insider to leave only a limited number of extremely weak *indicators of compromise* (IoCs). Secondly, most insider attacks are executed in multiple phases over a long period of time. Therefore, effective insider threat detection has to be undertaken using long-term monitoring across a wide range of audit data sources. Moreover, the increased scale and complexity of a modern network introduce a considerable amount of noisy information, significantly increasing the cost of data collection, storage and analysis. In summary, weak IoCs, complex audit data sources, and expensive cost in handling big data are the main challenges for an effective and efficient detection of malicious insiders.

Currently, most insider threat detection approaches are data-driven [2], and generally can be classified into two categories. One category deploys multiple sub-detectors, where each sub-detector focuses on a specific type of suspicious activity such as users who access data they do not need to

The associate editor coordinating the review of this manuscript and approving it for publication was Junaid Arshad.

TABLE 1. Comparison with the existing approaches.

Category	Approach	Independent of domain knowledge	Multiple types of audit data	Interpretability
Rule Based	Unusual access [5], [11]	×	×	Strong
	Signature match [12]	×	×	Strong
Graph-based	GBAD [13]	√	×	Medium
Statistical	Gaussian mixture model (GMM) [14]	√	×	Weak
Machine/deep learning	PRODIGAL [6], [7]	√	√	Weak
	Beehive [8],	√	√	Weak
	Autoencoder [15]	√	√	Weak
	RNN [16]	√	√	Weak
	Our work	√	√	Strong

know [5]. Malicious insiders are then identified by orchestrating alerts captured by these sub-detectors [6], [7]. The other category applies machine learning algorithms to work on the features extracted from all relevant audit data, then flags those that significantly deviate from the rest as suspicious [8]. These approaches, however, suffer from one or more following limitations: (1) Individual sub-detectors fail to indicate the presence of an insider with both high confidence and a low false positive rate (FPR). These sub-detectors need to be combined using an orchestration engine to improve the overall detection accuracy; otherwise, it requires heavy workload of security experts in validating false positives. (2) Due to more complicated processes for data collection and storage and orchestration, the engineering cost is higher when multiple sub-detectors are deployed separately. (3) Feature engineering relies on domain knowledge about how an insider attack is characterised, resulting in additional research costs and a limited visibility to unknown insider attacks. (4) Many machine learning algorithm-based approaches have difficulty in providing a straightforward interpretation and explanation of the output results, and the relationship between each result and the corresponding events. This has been an issue for machine learning algorithm-based approaches, as they need to transform raw data from a human-recognisable form, into a structured machine-recognisable form, during which, much of the raw data semantics is lost [9]. To overcome the aforementioned limitations, we present a novel approach that realises behavioural analysis based insider threat detection using a corpus transformed from various security logs.

In this paper, we design and implement a Word2vec-based approach [10] for insider threat detection. We obtain the likelihood of a particular behaviour to be suspicious using the similarities between words by querying the Word2vec model trained with the corpus generated from multiple types of security logs. Based on such likelihoods, we are able to detect insiders who behave unusually. Particularly, the proposed approach comprises three components. First, a *log2text* component parses and transforms events that come from different security logs into identically formatted texts. The texts are then concatenated and sorted in line with their timestamps to yield a corpus, which is undertaken by the *text2corpus* component. Thirdly, the *anomaly detection* component trains a Word2vec model with the corpus, and computes an approximation of the posterior probability for the behaviour represented by the transformed event, namely $p(behaviour|user)$. By comparing the probability against a given threshold, the event associated with an unusual behaviour is labelled as suspicious. The *anomaly detection* component labels a person who has generated multiple suspicious events as a malicious insider.

This paper makes the following contributions:

- We propose a new approach to deal with insider threats, which reconstructs semantic properties from multiple types of security logs and detects insider threats from a behaviour analysis's perspective. This approach transforms different security logs into the same format and conduct an universal analysis, reducing costs in engineering and orchestrating many sub-detectors in practical applications.
- The transformation that reconstructs semantic properties does not rely on any domain knowledge. In other words, the proposed approach works in a purely "let the data talk" manner, offering much higher flexibility.
- As the analysis is conducted on the texts transformed from the security logs, the proposed approach comes with much stronger ability to explain the level of insider suspiciousness.

The rest of this paper is organised as follows: Section II presents the related works. The *log2text* and *text2corpus* components are detailed in Section III. Section IV introduces how to detect a malicious insider based on Word2vec. Numerical experiments are presented in Section V, including a summary of the experimental datasets, the experimental settings and performance evaluation. Finally, Section VI concludes this work.

II. RELATED WORK

Most of the existing approaches identify malicious insiders from an anomaly detection perspective [17]–[19]. More specifically, these are approaches can be categorised as rule-based, graph-based, statistical and machine/deep learning-based [2]. As Table 1 shows, previous approaches tended to be rule-based, such as [5], [11], and [12]. These approaches all focused on detecting insiders who undertake data exfiltration activities, leveraging a series of rules to expose unusual access to files and directories [11], access to data without a 'need-to-know' requirement [5], or transfers of large amounts of data to recipients who do not exist in the organisational white-listed name-space [12]. Unfortunately, rule-based approaches rely heavily on domain knowledge and are unable to deal with previously unseen insider attacks. Furthermore, rule-based approaches can only respond to an insider attack when a clear and distinctive IoC is identified. In order to tackle an insider attack as early as possible, more recent approaches have analysed audit data using more complex graph models [13], statistical models [14] or machine/deep learning algorithms [6]–[8], [15], [16], which are more likely to defend against a variety of attacks with less dependence on domain knowledge.

In particular, a graph-based approach [13] models system calls which are related to a user's logon/off activities and file operations, such as *exec, execve, time, login, logout, su, rsh, rexecd, passwd, rexd* and *ftp*. It then applies a graph-based anomaly detection (GBAD) algorithm to test whether each chunk of system calls (i.e., a contiguous set of system calls) is consistent with the previously established normal behaviour patterns. Similarly, Song's approach analyses file system- and process-related system calls [14], which labels any unusual (e.g., frequency) or unauthorised access to specific file system locations as suspicious, as well as inconsistent or unauthorised child process forks. The PRODIGAL (PROactive Detection of Insider threats with Graph Analysis and Learning) [6], [7] is a representative machine learning based approach. This approach extracts more than 100 features from a wide range of security logs such as email, web proxy and Lightweight Directory Access Protocol (LDAP). A number of sub-detectors are designed specifically for a subset of the features, using various algorithms such as KDE, GMM, LR, k-NN, HMM, STINGER and seed set expanse (SSE). Finally, a scoring mechanism groups anomalies as a function of a given domain entity (e.g., user or computer ID), enabling most suspicious events to be displayed at the top of the score list. The Beehive [8] approach takes advantage of some domain knowledge, extracting 15 features from web proxy, DHCP, VPN and LDAP logs related to host, traffic and policy. Detection is implemented by means of PCA and k-means clustering algorithms. More recently, deep learning-based approaches have shown great potential in handling a large number of features, allowing raw features to be used directly without explicit feature engineering, thereby reducing the dependency on domain knowledge. For example, Tuor's approach [16] uses 408 continuous features and six categorical features in total from email, web proxy and file access logs, for training a deep recurrent neural network (RNN) to perform detection in real-time. Liu et al. also propose a deep learning based approach to detect malicious insiders [15], where four deep autoencoders are trained with the features extracted from web proxy, authentication, file access and operating system logs. The feature extraction leverages only common sense to transform the hourly occurrences of each behaviour into features, and the autoencoder reconstruction error is employed as the means of performing detection.

The graph-based and statistical approaches have resulted in some improvements compared with the rule-based approaches [13], [14]. Even though these approaches adopt a single type of audit data, they do not need any domain knowledge to establish rules for identifying anomalies. However, there is still a limitation that they are unable to prevent malicious insiders during the early stage of the attack due to the lack of total visibility of insider behaviour. In contrast, the machine/deep learning based approaches analyse multiple types of audit data and can provide a much deeper insight into insider behaviour. These approaches can either orchestrate multiple sub-detectors, each of which tackles a specific insider behaviour [6], [7], [15], or employ just one algorithm to handle features extracted from various audit data [8], [16]. The former category generally suffers from a high design and deployment complexity, and has limited extendability. Working with a large number of features does offer greater flexibility. However, the computational cost can be extremely high, especially when dealing with large datasets [16]. Alternatively, domain knowledge can be used to reduce the number of features to some extent [8], but it increases the reliance on human analysts. In this paper, we propose a novel approach that addresses the aforementioned problems.

III. TRANSFORMING SECURITY LOGS TO CORPUS

As introduced in section II, most existing approaches analyse security logs to detect insiders, which are the most common audit data available to an organisation. In this section, we introduce how to transform security logs into textual form for all events, and then produce the Word2vec trainable corpus with the transformed texts. In the meantime, a brief introduction to the CMU CERT Programs insider threat database [20], [21] is given. All the experiments throughout the rest of this paper are conducted with datasets extracted from this database, which comprises various security logs collected from a medium-sized organisation over 18 months, with a range of insider attacks appeared and labelled.

A. SECURITY LOGS AND THE CMU'S INSIDER THREAT DATABASE V6.2

Traditionally, a log is defined as a record of the event occurring within an organisation's computer systems and networks [22]. A log that contains security-related information is referred to as a security log, for example:

- Logs generated from security software,
- Operating system and application logs that are related to security.

Typical security logs can be collected from a variety of sources, such as antivirus software, intrusion detection/prevention systems, remote access software, web proxies, authentication servers, routers, firewalls, system events, audit records and various other applications [22]–[26]. They can be broadly categorised as network, host and contextual data, and currently constitute the primary source of security audit data for detecting malicious insiders [2]. There also exists other data sources that are potentially effective in tackling malicious insiders, such as social communication patterns, psychological assessment profiles [27], keystroke dynamics [28], mouse movements [29] and eye movements [30]. However, these data sources are not always available to an organisation due to high collection costs, or issues raised from privacy concerns. Hence, we focus primarily on security logs.

Oliner *et al.* presented how to leverage security logs to detect security breaches or misbehaviour [31]. However, in the context of malicious insider detection, the scenario is often more complicated, as an insider often takes a series of actions to achieve the final target. For example, a possible scenario could be: during after-hours a user logs into a computer that he/she does not regularly use, browses websites and downloads files which may contain sensitive/protected information, and uses a removable drive more frequently compared to his/her usual activity to exfiltrate data [1]. In this scenario, the insider can only be confidently labelled as malicious until most of the suspicious actions have been captured, since every single action is likely to be a weak IoC, and not sufficient for making a decision. Such an insider attack can be addressed by analysing authentication, web proxy, file access and system logs separately and correlating the suspicious events according to a specific primary key such as *user* or *IP address*. In particular, analysis of authentication logs helps to decide how likely a computer is being used by a specific user and what the standard working hours are; web proxy and file access logs can reflect the sensitivity of the information being accessed, and system logs contain detailed usage activities relating to removable drives. The above example has demonstrated how we can detect a malicious insider with various security logs.

The CMU's insider threat database is to date the best benchmark dataset, which is created by simulating a medium-sized organisational intranet [20], [21]. The full database (version 6.2) is approximately 200 GB and contains 4000 users with their daily activities being logged between January/2010 and June/2011. Totally, there are 5 users who have taken malicious actions and affected other 23 users, each of which represents a typical scenario of insider attack. The database is primarily comprised of various security logs, including authentication, system (removable drive usage activities), web proxy, email, file access and LDAP. At the same time, the database also comes with psychometric and decoy data which are supposed to provide some contextual information. With a potential pre-processing procedure, the security logs are transformed and correlated, resulting in some clean and tidy datasets (CSV files) namely http, file, device, logon and email. The http.csv is transformed from web proxy logs, which suggests that how the user access Internet ('visit', 'download' or 'upload'), and provides the summarised text for the web page accessed. The file.csv corresponds to the system and file access logs, reflecting how a user accesses a removable drive and what files are copied, opened or deleted. The device.csv should be transformed from the system logs, which are only about a removable drive's connect/disconnect and the file tree. The logon.csv is a transformed result from the authentication logs, which specifically details a user's logon/logoff and its timestamp. The email.csv provides information about a user's daily email communication, such as send/receive, email size, attachment count and content. Since, according to the ground truth, an email's connection with an insider attack can only be exposed by text mining (e.g., topic model), it is inconsistent with the ways of processing http.csv, file.csv, device.csv and logon.csv (as detailed later). Thus, in this paper, we don't use email.csv.

B. LOG2TEXT

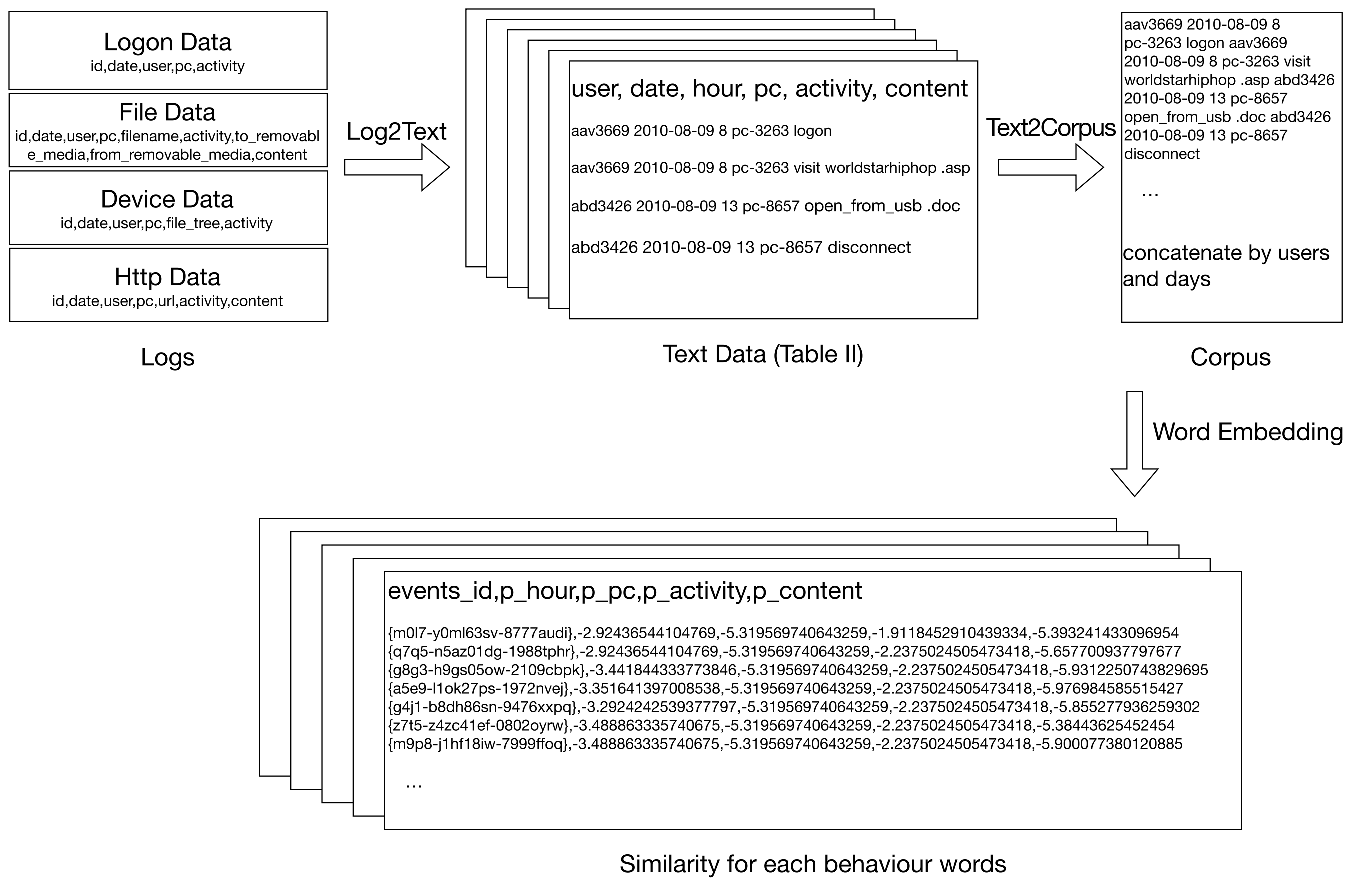
In this paper, we apply Word2vec to analyse different types of security logs simultaneously, which not only reduces the complexity of deploying multiple sub-detectors, but also simplifies the decision-making process. The remainder of this section presents how to consolidate various security logs into a Word2vec-trainable corpus, and Figure 1 illustrates how the components *log2text* and text2corpus work together.

Training a Word2vec model requires a large-sized corpus of text, organised as documents, paragraphs or sentences [10]. We may think of a security log as natural language text; for example as follows:

```
Feb-1 00:00:02 bridge kernel: INBOUND TCP:
IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1
SRC=192.150.249.87 DST=11.11.11.84 LEN=40
TOS=0x00 PREC=0x00 TTL=110 ID=12973
PROTO=TCP SPT=220 DPT=6129 WINDOW=16384
RES=0x00 SYN URGP=0
Feb-1 00:00:02 bridge kernel: INBOUND TCP:
IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1
SRC=24.17.237.70 DST=11.11.11.95 LEN=40
TOS=0x00 PREC=0x00 TTL=113 ID=27095
PROTO=TCP SPT=220 DPT=6129 WINDOW=16384
RES=0x00 SYN URGP=0.
```

By concatenating the words occurring in the above events, a corpus can be produced. However, in contrast to natural language text, such a simple concatenation fails to take into account the semantic and syntactic relations among 'words'. Furthermore, noisy event information and the existence of duplicated entries significantly impact the linguistic properties of the produced corpus. This may result in a large but useless Word2vec model. To overcome this, we design two specific transformation components, *log2text* and *text2corpus*, for dealing with security logs.

The *log2text* component applies a 4W ('who', 'when', 'where', 'what') sentence template to transform each event into text. This enables us to reconstruct the 'linguistic semantic' properties from orderless words that a security log presents. The 'who', in this context, often indicates the entity that an event is associated with, e.g., *user*, *IP address*, *email address*, *hostname* and so forth. The 'when' represents the information extracted from the timestamps, which may vary according to the given time granularity. For example, the analysis conducted daily may need *date* and *hour* to be explicitly extracted. Moreover, if seasonality is essential in the analysis, the *day-of-the-month* and *day-of-the-week* may also


Logon Data
id,date,user,pc,activity
File Data
id,date,user,pc,filename,activity,to_removable_media,from_removable_media,content
Device Data
id,date,user,pc,file_tree,activity
Http Data
id,date,user,pc,url,activity,content
Logs
Log2Text
Text Data (Table II)
Text2Corpus
concatenate by users and days
Corpus
Word Embedding
Similarity for each behaviour words

```
user, date, hour, pc, activity, content
aav3669 2010-08-09 8 pc-3263 logon
aav3669 2010-08-09 8 pc-3263 visit worldstarhiphop .asp
abd3426 2010-08-09 13 pc-8657 open_from_usb .doc
abd3426 2010-08-09 13 pc-8657 disconnect
```

```
aav3669 2010-08-09 8
pc-3263 logon aav3669
2010-08-09 8 pc-3263 visit
worldstarhiphop .asp abd3426
2010-08-09 13 pc-8657
open_from_usb .doc abd3426
2010-08-09 13 pc-8657
disconnect
   ...
```

```
events_id,p_hour,p_pc,p_activity,p_content
{m0l7-y0ml63sv-8777audi},-2.92436544104769,-5.319569740643259,-1.9118452910439334,-5.393241433096954
{q7q5-n5az01dg-1988tphr},-2.92436544104769,-5.319569740643259,-2.2375024505473418,-5.657700937797677
{g8g3-h9gs05ow-2109cbpk},-3.441844333773846,-5.319569740643259,-2.2375024505473418,-5.9312250743829695
{a5e9-l1ok27ps-1972nvej},-3.351641397008538,-5.319569740643259,-2.2375024505473418,-5.976984585515427
{g4j1-b8dh86sn-9476xxpq},-3.2924242539377797,-5.319569740643259,-2.2375024505473418,-5.855277936259302
{z7t5-z4zc41ef-0802oyrw},-3.488863335740675,-5.319569740643259,-2.2375024505473418,-5.38443625452454
{m9p8-j1hf18iw-7999ffoq},-3.488863335740675,-5.319569740643259,-2.2375024505473418,-5.900077380120885
   ...
```

FIGURE 1. How security logs are transformed into a corpus.

be required. In some cases, the 'where' is interchangeable with the 'who', conveying auxiliary information about the entity location. For example, if *user* is the entity, the 'where' can be expressed by *IP address* or *hostname*, namely a user undertaking an activity from a certain IP address or computer. Of course, geographical locations such as *city* and *country* are also valid information for the 'where'. Lastly, what actions the entity has taken are summarised by the 'what'. This largely varies according to the type of security log. For example, in a web proxy log the 'what' can be abstracted from the *URL* accessed and its *category*, the *bytes* transferred between client and server, the *HTTP method* and *HTTP status*, and in an email log the *to/from*, *subject* meta-data and *attachment*.

By using the 4W sentence template, this paper implements a relatively straightforward and compact instance of *log2text* for consolidating web proxy, authentication, file access and removable drive usage logs which, in the CMU's insider threat database, are equivalent to *http*, *logon*, *file* and *device* respectively. It is worth mentioning again that the insider threat database does not present data in their original forms. In other words, they have been pre-processed. Consequently, in practice, security logs require to be pre-processed similarly before applying the components *log2text* and *text2corpus*.

TABLE 2. The 4W sentence template.

4W Template	Word(s)	Description
who	*user*	user's unique ID
when	*date*	date extracted from timestamp
	hour	hour extracted from timestamp
where	*pc*	computer's unique ID, i.e. hostname
what	*activity*	download, upload, copy, delete, etc.
	content	separately defined

As listed in Table 2, the *log2text* abstracts each event as six separate words (strings) regardless of its type, where *user*, *date*, *hour* and *pc* can be easily obtained from an event, but the words *activity* and *content* are parsed differently. In particular, a *http* event yields only 'visit', 'upload' or 'download' for the *activity*, and we extract the domain and file extension from its full URL as the *content*. From a *logon* event, 'logon' or 'logoff' are extracted as the *activity* and a null string is inserted as a placeholder for the *content*. A *file* event may contain activities including 'open', 'copy', 'delete' and 'write'. We concatenate these strings with 'to_usb', 'from_usb' or 'local' to generate the *activity* for each event, while the file extension is extracted as the *content*. A *device* event indicates the 'connect' or 'disconnect' of a removable drive which is parsed to produce the *activity*, and the file paths accessed are

TABLE 3. Examples of transformed events by the *log2text* component.

Security Log Type	User	Date	Hour	PC	Activity	Content
http	aav3669	2010-08-17	11	pc-3263	visit	macrumors .aspx
file	abd3426	2010-08-16	8	pc-8657	copy_from_usb	.zip
device	abd3426	2010-08-17	9	pc-8657	connect	_abd3426
logon	abd3426	2010-08-16	8	pc-8657	logon	

combined into one string as the *content*. Similarly, different security logs can be transformed into an identical format. Table 3 shows the examples drawn from the experimental dataset for each type of security log. At this time, the *text* can be generated by concatenating the six words for each event. The entire process is implemented via the *log2text* component.

C. TEXT2CORPUS

Once the texts are produced by the *log2text* component, the *text2corpus* component is able to generate the corpus subject to the given rules. Since we detect malicious insiders depending on their daily behaviours in this paper, the texts are grouped by *user* and *date*, and then each group is sorted by timestamp. For example:

```
abd3426 2010-08-16 8 pc-8657 logon
abd3426 2010-08-16 8 pc-8657 visit fedex .jsp
abd3426 2010-08-16 8 pc-8657 connect _abd3426
abd3426 2010-08-16 8 pc-8657 open_local .doc
abd3426 2010-08-16 17 pc-8657 logoff
```

The resulting corpus can be regarded as a diary of user activities, with linguistic properties reconstructed from raw orderless words within the events. In addition, the *log2text* and *text2corpus* components depend solely on some general knowledge of behaviour analysis rather than any specific security-related domain knowledge.

IV. MALICIOUS INSIDER DETECTION VIA WORD EMBEDDING

In the proposed approach, the detection is realised by following the general concept of anomaly detection [17]. We train a Word2vec model with data collected for a particular period of time as, for example, one week. Thus, the conditional probability of a behaviour represented by any two words can be obtained by computing the word similarities. Then, a behaviour will be labelled as an anomaly if its probability is smaller than a given threshold. If a number of anomalies are attributed to the same user, this user will be identified as a malicious insider. The following two subsections introduce the basic concept of Word2vec and how we apply it to insider threat detection.

A. WORD EMBEDDING WITH WORD2VEC

Word2vec was created by Mikolov *et al.* for learning word embedding in natural language text [10], [32]. Word embedding are dense representations of words in the form of numeric vectors. There are two commonly used models, namely the continuous Bag-of-Words (CBOW) model and the Skip-gram model. The models can be optimised using either hierarchical softmax or negative sampling in order to produce the distributed representations. Taking a large corpus of text as an input to Word2vec, the output is a vector space in which each unique word is coded as a high-dimensional vector. This vector space arranges words that come from a similar context in the corpus, proximal to each other. As suggested by [10], optimising a Skip-gram model using a hierarchical softmax is slightly slower, but better for uncommon words, while the CBOW model with negative sampling is better at handling frequent words with low dimensional vectors. In this work, since the corpus is produced by the *log2text* and *text2corpus* components without noisy information and duplicates, we choose the Skip-gram model and a hierarchical softmax in the experiments.

Next, we briefly present how the Skip-gram model works with hierarchical softmax. Given a sequence of training words $w_1, w_2, \cdots, w_N$, the objective function of a Skip-gram model is formulated as

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{-l\le j\le l, j\ne 0}\log p(w_{i+j}|w_i) \tag{1}$$

where l is the fixed size of a sliding window. In theory, $p(w_{i+j}|w_i)$ can be resolved with a multi-classifier such as softmax, i.e.

$$p(w_{i+j}|w_i) = \frac{\exp(v'^{T}_{w_{i+j}} v_{w_i})}{\sum_{w\in W}\exp(v'^{T}_{w} v_{w_i})} \tag{2}$$

where v_w and v'_w are the "input" and "output" vector representations of w, and W is the dictionary containing all of the unique words extracted from the corpus. However, softmax is computationally infeasible in practice since W is often huge. Alternatively, the hierarchical softmax [33] can be applied to approximate the softmax by constructing the output layer as a binary tree which assigns short codes to frequent words and evaluates only approximately $\log_2(|W|)$ nodes rather than $|W|$ nodes. The hierarchical softmax defines the approximation as:

$$p(w|w_i) = \prod_{j=1}^{L(w)-1}\sigma(\langle n(w,j+1) = ch(n(w,j))\rangle v'^{T}_{n} v_{w_i}) \tag{3}$$

where $n(w, j)$ is the jth node on the path from the root to w, the angled brackets represent a Boolean check for returning 1 if the case is true and -1 otherwise, $L(w)$ is the depth of the tree and $ch(n)$ the child of node n and, finally

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

B. MALICIOUS INSIDER DETECTION USING BEHAVIOURAL PROBABILITIES

With the *log2text* and *text2corpus* components, the raw security logs are transformed into texts that reflect insider behaviours and, a trained Word2vec model retains the information that describes correlations among words. Consequently, it is possible to undertake insider threat detection by querying the model about a user's behaviour. For example, the following text

abd3426 2010-08-16 16 pc-8657 copy_from_usb .pdf

describe the user 'abd3426's behaviours, from which we are interested to know how likely they are to be undertaken: the user 'abd3426' was active at 4:00 PM, copying some items (here a PDF file) from a removable drive and etc. More formally, we name the words, which are listed in the columns *date*, *hour*, *pc*, *activity* and *content* of the transformed events, aside from the *user* word, as behaviour-related words. The following behavioural probabilities for each pair of the behaviour-related words are computed for deciding whether this event is suspicious or not: $p(date|user)$, $p(hour|user)$, $p(pc|user)$, $p(activity|user)$ and $p(content|user)$, where each behavioural probability is actually a posterior probability that indicates the likelihood of the behaviour. If the number of suspicious events generated by one user exceeds a given threshold, the user would be labelled as malicious/suspicious.

However, the Word2vec model does not provide a shortcut for computing $p(\mathbf{Y}|\mathbf{X})$. As introduced in IV-A, when each word is coded as a fixed-length vector, the cosine similarity between any two vectors is the metric that indicates how close the two words are from a semantic perspective. The cosine similarity can be expressed as

$$\text{similarity}(x, y) = \cos(\theta) = \frac{\mathbf{v}_x \cdot \mathbf{v}_y}{\|\mathbf{v}_x\| \|\mathbf{v}_y\|} \tag{4}$$

where x and y are two words, $\mathbf{v}_x$ and $\mathbf{v}_y$ their vectors in the model, and θ the angle between the two vectors. According to Bayes' theorem, $p(\mathbf{Y}|\mathbf{X})$ can be rewritten as

$$p(\mathbf{Y} = y|\mathbf{X} = x) = \frac{p(\mathbf{X} = x, \mathbf{Y} = y)}{\sum_{y \in \mathbf{Y}} p(\mathbf{X} = x, \mathbf{Y} = y)} \tag{5}$$

Although $p(\mathbf{Y}, \mathbf{X})$ is also not immediately achievable, we can use the cosine similarities of two words to approximate the posterior probability [34], namely

$$\hat{p}(\mathbf{Y} = y|\mathbf{X} = x) = \frac{\text{similarity}(x, y)}{\sum_{y \in \mathbf{Y}} \text{similarity}(x, y)}. \tag{6}$$

It should be noted that similarity(x, y) is not a valid approximation for $p(\mathbf{X} = x, \mathbf{Y} = y)$ but $\hat{p}(\mathbf{Y}|\mathbf{X})$ is. This is because similarity(x, y) represents the distance between x and y for $\forall y, y \in Y$ in line with the occurrences of y in x's neighbouring window, which is a result from equation 1. Thus, similarity(x, y) dividing by its sum over Y for a specific x yields $\hat{p}(\mathbf{Y}|\mathbf{X})$. In the context of malicious insider detection, we explore the posterior probabilities for all $\hat{p}(\mathbf{Y} = behaviour|\mathbf{X} = user)$, i.e.,

$$[\hat{p}(date|user)\ \hat{p}(hour|user) \cdots \hat{p}(content|user)]$$

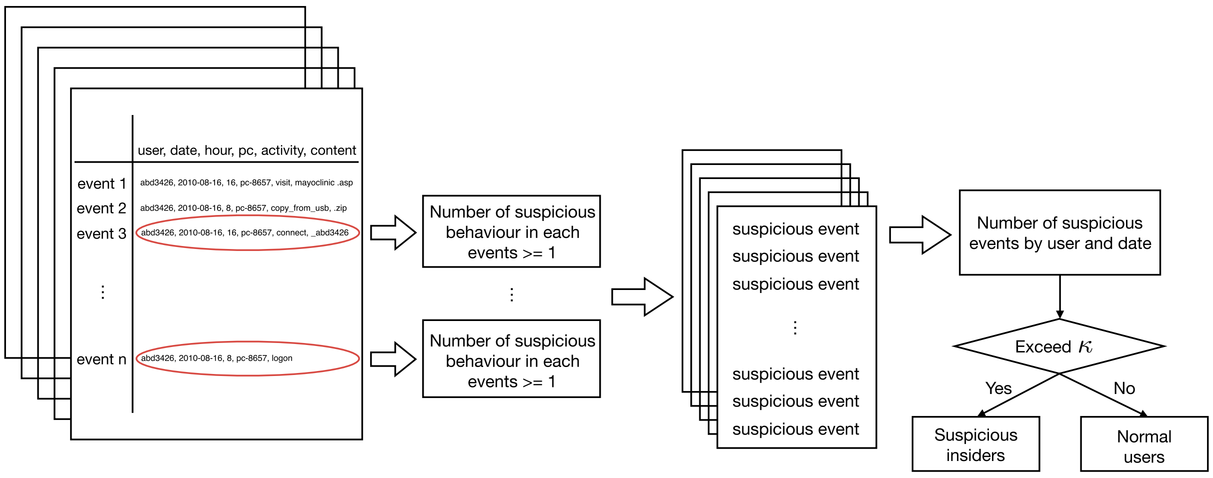
to determine how suspicious the user's behaviour is.

Algorithm 1 The Complete Train-Detect Cycle

```
Data: http, file, logon, device logs
Result: Report suspicious user, date for further
        investigation
while week do
    transform the logs into texts using log2text;
    combine the texts into a corpus using text2corpus;
    train a Word2vec model using the corpus;
    foreach event in the corpus do
        if $\hat{p} \leq \tau$ then
            Label event as suspicious;
        end
    end
    foreach user, date in the corpus do
        if suspicious events $\geq \kappa$ then
            Label user as suspicious on date;
        end
    end
end
```

The complete train-detect cycle is summarised by Algorithm 1 and presented as follows. Since an insider attack often persists with a series of unusual actions, there is a better chance to detect the insider if the behaviours are monitored over a relatively broad period of time. In this paper, we choose the detection to be performed for each user daily, which is a typical time granularity for observing the behaviours [15], [16]. This usually requires multiple days of data for training, in order to supply sufficient temporal and spatial references from an anomaly detection's perspective. In the proposed approach, the Word2vec model is trained weekly. In other words, we collect every week's security logs, produce a corpus, train a Word2vec model and compute the likelihood of being malicious for each user daily during the given week. When the above-mentioned behavioural probabilities are computed for all transformed events, the τ-th percentile of a behaviour word is specified as the threshold. For example, given $\tau = 1$ and the behaviour word *activity*, an event containing the top 1 percent smallest $\hat{p}(activity|user)$ is labelled as an unusual behaviour for the activity. If an event includes multiple unusual behaviours (i.e. more than one), it is labelled as a suspicious event. At this time, the other threshold κ is employed to label a suspicious user on that day. Figure 2 briefly illustrates how the detection is performed.

It does not have to train the Word2vec model every week and perform detection on the days of the week. Alternatively,



```
user, date, hour, pc, activity, content
event 1  abd3426, 2010-08-16, 16, pc-8657, visit, mayoclinic .asp
event 2  abd3426, 2010-08-16, 8, pc-8657, copy_from_usb, .zip
event 3  abd3426, 2010-08-16, 16, pc-8657, connect, _abd3426
⋮
event n  abd3426, 2010-08-16, 8, pc-8657, logon
```

FIGURE 2. An illustration of insider threat detection through insider behaviours probability approximation.

we can also train a daily model to detect a user's hourly behaviour. The choice depends on the application scenarios. The current setting is aligned with the assumption that the IoCs are weak and only appear in possibly a couple of non-contiguous time intervals (hours in this case) in a day and a malicious insider takes actions only during one or two days a week. Secondly, threshold selection usually determines the performance of anomaly detection and can often be subjective [17]. The proposed approach is thus designed to work with two thresholds, τ and κ, which can be empirically selected in practical applications. We found that the undertaking of numerical trial-and-error experiments is the best approach for selecting appropriate thresholds. For validation, we also assume that security experts are available for post-processing to investigate the detected incidents in order to eliminate false positives and reveal the details of the insider attack vectors.

V. EXPERIMENTS AND RESULTS

The numerical experiments are undertaken with the Carnegie Mellon University (CMU) CERT Programs insider threat database v6.2 [20] and implemented using Gensim [35]. The following subsections detail how we create the experimental dataset, tune the parameters for training the Word2vec models, select the thresholds and evaluate the experimental results. We then present a comparison study with other approaches.

A. THE DATASET

The processing of the entire original 200 GB dataset would need rather large computing resources, which are unavailable for this study. Intuitively, we should undertake a random sampling from the raw dataset, which is supposed to involve all the insider attack scenarios to be tested without loss of generality. In particular, the sampling is realised from two aspects. Firstly, we randomly choose 500 out of the 4000 users as the representatives, where the known malicious insiders 'PLJ1771', 'ACM2278' and 'CDE1846' must be involved. Secondly, as the detector works on a weekly basis, we split the *http*, *file*, *device* and *logon* logs into a number of weekly ones and only choose those containing the events affected by the insider attacks. Thus, only 9 discontinuous weeks of security logs are kept in the reduced experimental dataset, instead of the raw dataset which spans over 18 months. On average, the size of the weekly security logs for 500 users is about 200 MB. As mentioned in Section III, the weekly security logs are transformed using the *log2text* and *text2corpus* components to produce the Word2vec trainable corpus which, in general, is comprised of around 200,000 events and 30 - 40 MB. As stated later, the training time for such a scale of corpus varies between 30 and 400 seconds with a high-specification laptop, enabling us to conduct a extensive set of experiments to test the proposed detector comprehensively. Table 4 summarises the ground truth for the three malicious insiders and the dates that they undertake the insider attacks. Therefore, some preliminary sub-selection is undertaken. We randomly chose a subset of users (500 users) and their logs to conduct the experiments. As mentioned in section III, we transformed the security logs using the *log2text* and *text2corpus* components to produce the corpus. We expect that the 500 randomly chosen users should include some of the malicious insiders.

TABLE 4. Ground truth data for the three insider attacks (one user per insider attack). N.B.: Date format is year-month-day.

User	Week	Date
PLJ1771	20100809-20100815	20100812, 20100813
ACM2278	20100816-20100822	20100818, 20100819
	20100823-20100829	20100824
CDE1846	20110314-20110320	20110317
	20110321-20110327	20110322
	20110328-20110403	20110329
	20110404-20110410	20110406, 20110407
	20110411-20110417	20110411, 20110415
	20110425-20110501	20110425

We also intentionally remove some of the scenarios that do not involve the above-mentioned types of security logs. Moreover, as the training-detect cycle occurs weekly, only those weeks containing the relevant events are kept in the reduced experimental dataset. There are a total of three insider attacks (associated with three users) appearing over a period of nine weeks.

B. EXPERIMENTAL SETTING

For training a Word2vec model, the parameters *vector_size*, *window*, *epoch* and *min_count* are important. *vector_size* and *epoch* have a significant impact on the training time, namely a larger *vector_size* and *epoch*, results in longer training times. Empirically, the value of *vector_size* should be in the range of 100 to 1000 [32]. In the following experiments, we evaluated *vector_size* in the range of 100 to 500 with a step size of 100. The model's effectiveness is also very sensitive to the value of *window*. A *window* value that is too small takes into account only the correlation of words with a tight neighbourhood, which may result in a over-fitted model. Conversely, a *window* value that is too large may result in a model which is under-fitted, amplifying the impact of distant words. Since the *log2text* and *text2corpus* components have transformed each event into 5 or 6 words and arranged events according to their timestamps, intuitively, the value of *window* should be at least large enough to reflect correlations among words occurring in neighbouring events. Two values were tested for *window* namely, 5 and 10. The value of *epoch* specifies the number of iterations over the corpus. Increasing the value of *epoch* theoretically leads to more accurate word embedding, however, at the expense of a dramatic increase in training times. We used values equal to 5, 10 and 20 for *epoch*. The value of *min_count* is set to remove words which do not frequently appear while at the same time contributing little to the contexts. However, less frequent words have already been filtered during the transformation stage, therefore, *min_count* is set equal to 1 in all of the experiments.

The effectiveness of the detection depends on two thresholds: τ and κ, where τ is employed to label an event that includes multiple unusual behaviours and κ determines whether the user on that day is suspicious. As mentioned in subsection IV-B, the value of τ represents the τ-th percentile of the behavioural probabilities $\hat{p}(behaviour\ word|user)$ computed for all events in the corpus, and κ defines the threshold for labelling a user as suspicious if he/she generates more than κ suspicious events on a given day. It should be noted that there are multiple words in *content* for events generated from *http* log and, in this case, $\hat{p}(content|user)$ is defined as

$$\frac{1}{|content|}\sum_{word\in content}\hat{p}(word|user).$$

In theory, a smaller value of τ indicates a tighter metric to label an unusual behaviour, which may simultaneously decrease the true positive rate (TPR) and FPR. On the other hand, a larger value of κ yields a higher threshold to label a user as suspicious. We experimentally determine the appropriate ranges in selecting τ and κ . In total, we tested five values for τ ranging from 0.1 to 0.9 with a step size of 0.2 and κ ranging from 1 to 10 with a step size of 1. The receiver operating characteristic (ROC) curve for different values of κ's was then generated to illustrate the performance results.

C. PERFORMANCE RESULTS

TPR and FPR are the two most important metrics to evaluate the results, reflecting the performance of the detection. Furthermore, we are also interested in observing how different parameters impact on the results. We want to determine the sensitivity of the thresholds and also compare training times for different settings.

We first demonstrate how the performance is dependent on *vector_size*. Since τ varies between 0.1 and 0.9 and we have previously observed that the values 0.3 and 0.5 yield acceptable performances, Figure 3 shows the results for $\tau = 0.5$ where the legends indicate the other parameters. Overall, the best performance is achieved when *vector_size* = 200, although in theory a larger value of *vector_size* yields more accurate word embedding. Moreover, it can be observed that small values of *vector_size* work better with small values of *window* and *epoch*, and vice versa; for example, the best performance occurs when "*window* = 5, *epoch* = 5, *vector_size* = 100", "*window* = 5, *epoch* = 10, *vector_size* = 200" and "*window* = 10, *epoch* = 20, and *vector_size* = 300". The performances stabilise in the range up to *vector_size* = 300 and then start declining from *vector_size* = 500. The above observation suggests that an effective *vector_size* is strongly dependent on the corpus. In our experimental dataset, the corpus is almost comprised of 200,000 sentences (rows) and 12,000 unique words.

Figure 4 illustrates the performance comparison between *window* = 5 and *window* = 10 for *vector_size* equal to 200 and 300, respectively. It shows that, in most cases, *window* = 10 outperforms *window* = 5 and, especially for large values of *epoch*. It achieves a 100% TPR at an FPR of around 8%.

With a fixed *window* = 10, Figure 5 presents how performance varies according to *epoch*. Therefore, we could perceive that performance improves when *epoch* increases. However, comparable performance can be achieved for small values of *epoch* when the other parameters are judiciously chosen; for example, a TPR of 92% can be achieved for *epoch* = 5 at an FPR of 8% as shown in Figure 5a. This observation suggests that, in practice, we can choose a small value of *epoch* to achieve a suboptimal performance and avoid the higher computational cost when using a large value of *epoch*.

Table 5 details the best pair of TPR and FPR amongst different values of κ for each of the above experiments, where the 'best' is defined as the pair that produces the minimal value of $\frac{TPR}{FPR}$. Equivalently, this metric can be regarded as the minimal number of false positives being investigated (the cost) to find out a true positive. We also list

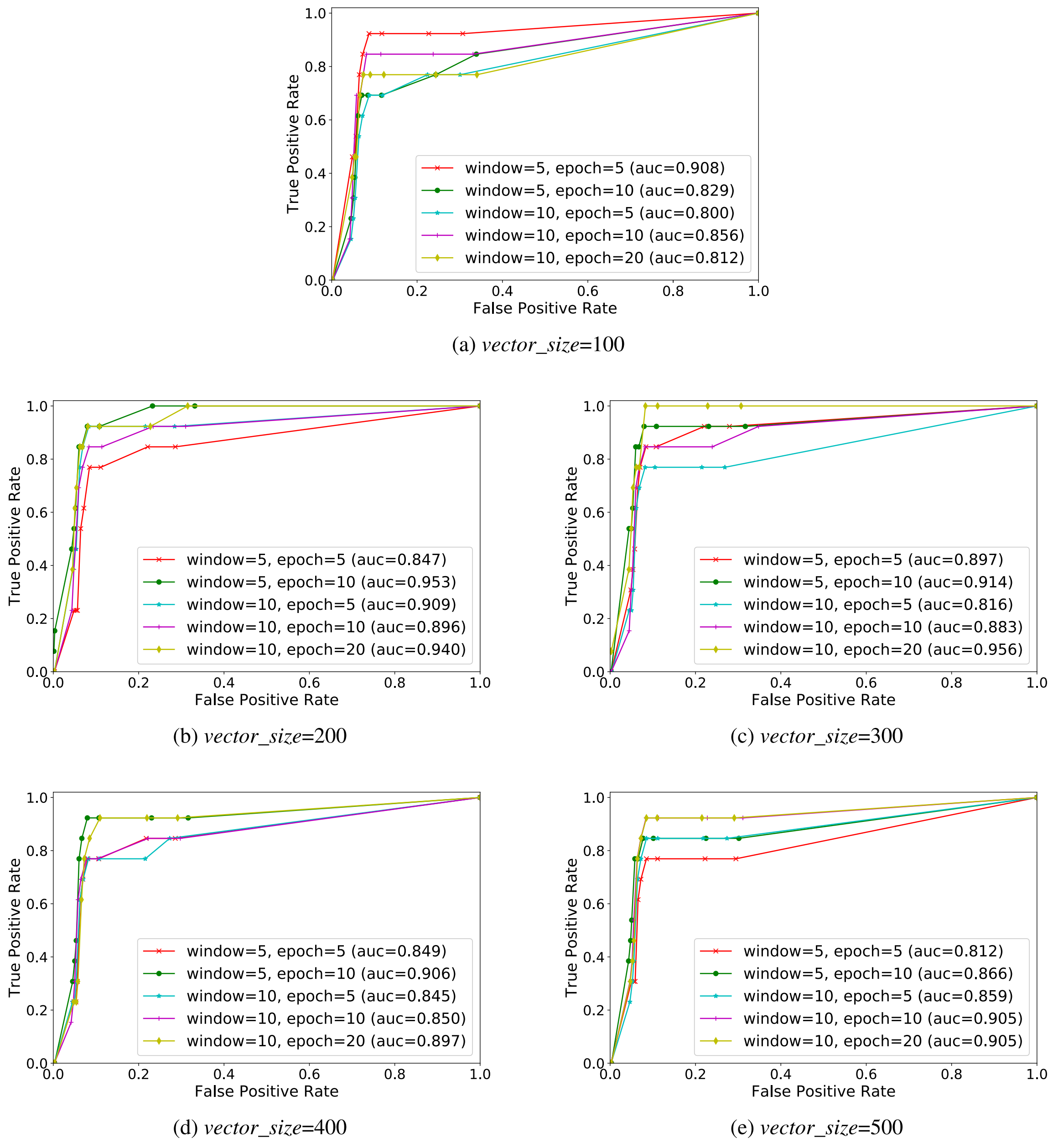
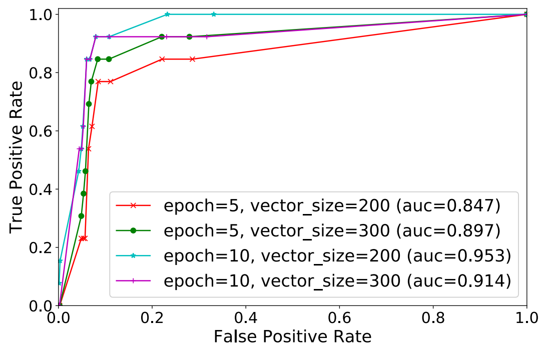


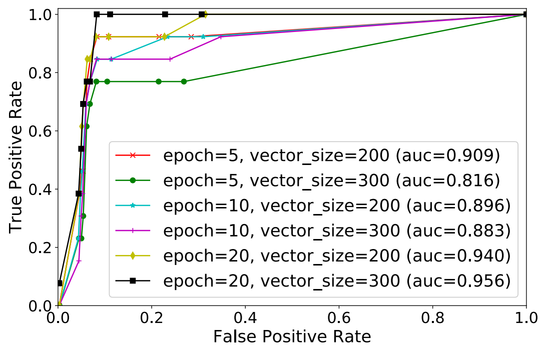
FIGURE 3. Impact on performance for different values of *vector_size* with $\tau = 0.5$.

the average training times for each experiment in the same table.

A few conclusions about parameter selection can be drawn so far. *Vector_size* is a parameter closely related to the corpus' size and, as a rule of thumb, challenging to choose. However, for dealing with security logs, we may be able to reuse the rule obtained from the experiments: i.e., a value of *Vector_size* between 200 and 300 should work well for a corpus that contains around 10,000 unique words. Furthermore, Table 5 has shown that the training time is linearly growing along with the values of *vector_size*. Thus, in practice, a relatively small *vector_size* is preferred as long as the performance is satisfactory. A larger value of *window* tends to produce better results with only a slightly higher computational cost. At the very least, the value of *window* should allow each word to have visibility in the context of the event where it resides, namely selecting a value larger than half of the words number in an event. We have observed that it is safe to select a larger value for *window* to ensure that each word is appropriately contextualised. As demonstrated in Figure 5 and Table 5, the performance can be improved in general for large values of *epoch*, but with an exponentially growing computational cost. Also, as mentioned above, continuously increasing the value of *epoch* can only improve performance by a small amount. Therefore, an arbitrarily large value of *epoch* is not necessary.

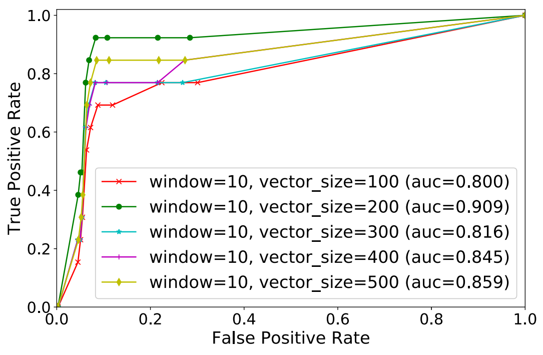


(a) *window*=5

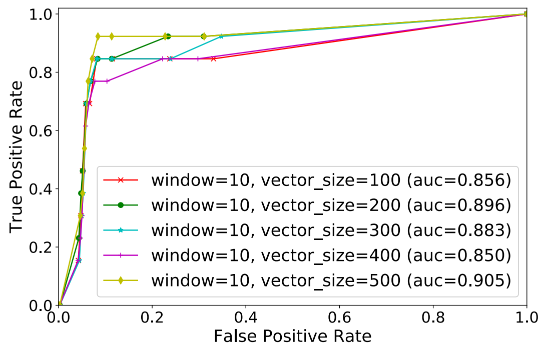


(b) *window*=10

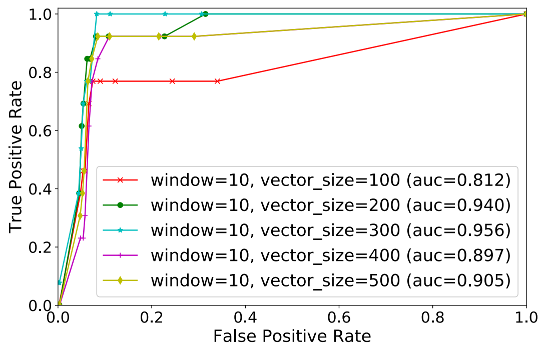
FIGURE 4. Impact on performance for different values of *window* with $\tau = 0.5$.



(a) *epoch*=5



(b) *epoch*=10

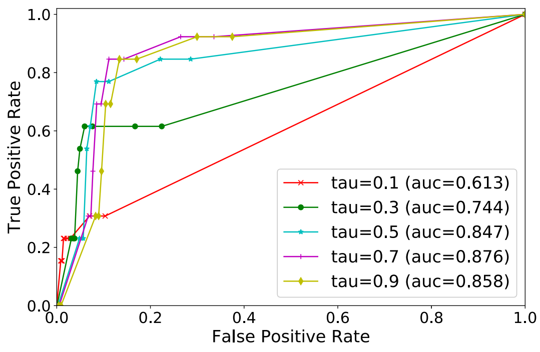


(c) *epoch*=20

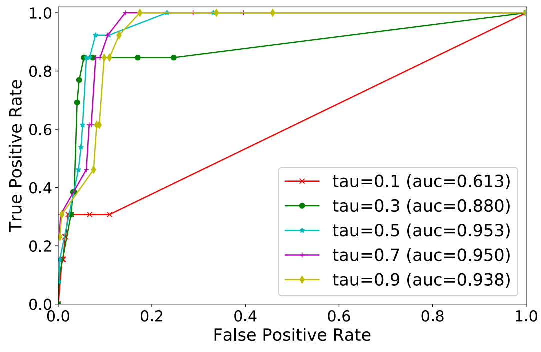
FIGURE 5. Impact on performance for different values of *epoch* with $\tau = 0.5$.

We also undertook experiments to determine how the thresholds impact on the performance. Figure 6 shows the ROC curves for κ values between 1 and 10 with a step size of 1, and illustrates the performance relating to τ. Since 200 is the most commonly valid value, *vector_size* is fixed to 200, and each of the five sub-figures represents a specific set of parameters. According to the above figures and the details provided in Table 5, varying κ from 4 to 6 produces a valid result and, usually, when $\kappa = 4$ the TPRs can exceed 0.6 while the FPRs are below 0.1. This is also consistent with the ground-truth in which most of the malicious insiders take more than 4 different suspicious actions in a given day. In terms of the area under the ROC curve (AUC), $\tau = 0.5$ and $\tau = 0.7$ are the best settings in most cases. However, it fails to perform the detection effectively when $\tau = 0.1$. When $\tau < 0.5$, the FPRs can generally be reduced to as low as 0.03 for a moderate value of κ; however, such a tight threshold results in many suspicious events undetected.

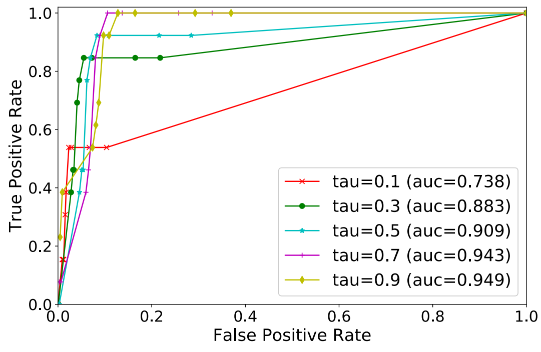
Finally, we discuss the effectiveness and scalability of the proposed approach in practical applications. Considering the highest TPRs as the metric for 'best', a few sets of parameters can achieve a 100% TPR at a FPR smaller than 10%, such as



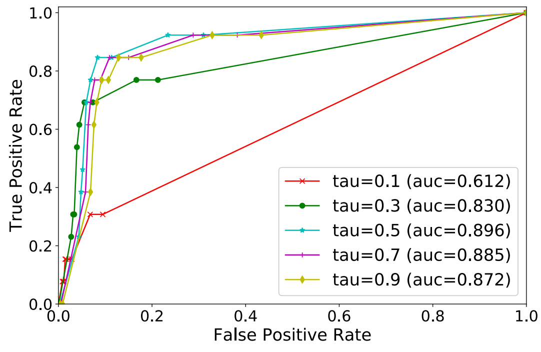
(a) *window*=5, *epoch*=5



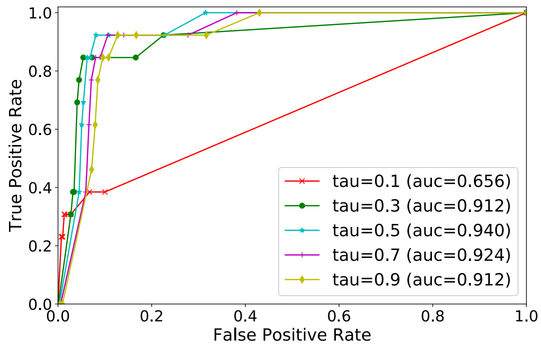
(b) *window*=5, *epoch*=10



(c) *window*=10, *epoch*=5



(d) *window*=10, *epoch*=10



(e) *window*=10, *epoch*=20

FIGURE 6. Impact on performance for different values of τ.

"$window = 10$, $epoch = 5$, $vector\_size = 200$, $\tau = 0.7$" and "$window = 10$, $epoch = 20$, $vector\_size = 300$, $\tau = 0.5$". These results indicate that all malicious insiders can be detected when security experts investigate up to 50 suspicious insider behaviour on average every day, which should be a reasonable workload for most organisations. Moreover, since the corpus has compiled users daily behaviours together in the formatted texts, each of the events could be labelled with the number of unusual behaviours. It saves security experts' much effort in searching evidenced contexts. The 500 users weekly corpus is usually 30-40 MB (i.e., 220,000 events and 12,000 words), resulting in a training time ranging from 40 seconds to 8 minutes on an Intel Core i7/16GB RAM laptop. In addition to the training time, the time of generating the corpus and performing the detection takes around 5-10 minutes. Given an enterprise-level computing capability, it is easy to scale the approach up to deal with a setting of 5,000-10,000 users. Accordingly, it can be concluded that the proposed approach is effective and scalable.

D. COMPARATIVE STUDY

In this section, we compare the proposed approach with the other existing approaches. Firstly, we summarise the features of the proposed approach. Subsequently, two comparative experiments are conducted to prove how the performance is

TABLE 5. Best performance for different combinations of parameters and the resulting averaged training times.

vector_size	*window*	*epoch*	τ	κ	TPR	FPR	Time(s)
100	5	5	0.5	6	0.77	0.064	38.16
200	5	5	0.5	4	0.77	0.084	48.12
300	5	5	0.5	5	0.77	0.070	68.59
400	5	5	0.5	5	0.69	0.069	87.01
500	5	5	0.5	4	0.69	0.072	106.15
100	10	5	0.5	5	0.62	0.072	39.76
200	10	5	0.5	6	0.77	0.061	53.78
300	10	5	0.5	5	0.69	0.068	73.81
400	10	5	0.5	5	0.69	0.069	89.05
500	10	5	0.5	6	0.69	0.064	112.64
100	5	10	0.5	6	0.62	0.061	77.08
200	5	10	0.5	6	0.85	0.060	98.73
300	5	10	0.5	6	0.85	0.060	140.09
400	5	10	0.5	6	0.77	0.060	172.34
500	5	10	0.5	6	0.77	0.058	204.15
100	10	10	0.5	6	0.69	0.058	78.29
200	10	10	0.5	6	0.69	0.059	107.23
300	10	10	0.5	6	0.69	0.060	157.98
400	10	10	0.5	5	0.69	0.064	187.15
500	10	10	0.5	4	0.77	0.063	222.36
100	10	20	0.5	6	0.69	0.065	164.58
200	10	20	0.5	6	0.85	0.062	233.98
300	10	20	0.5	7	0.69	0.053	327.09
400	10	20	0.5	5	0.77	0.072	369.6
500	10	20	0.5	6	0.77	0.064	453.57

improved while working with multiple types of security logs simultaneously and why the proposed approach outperforms another state-of-the-art deep learning based approach.

In terms of the literature, existing approaches can be roughly classified as rule/signature-based and anomaly-based. The first category creates rules/signatures to detect specific insider attacks [5], [11], [12]. This category normally is accurate in detecting the known attacks; however, it highly relies on domain knowledge, and be not able to handle previously unseen attacks. The anomaly-based approaches often train a model to fit the extracted features to label the ones which significantly deviate from the model as anomalous. Many statistical/machine learning algorithms such as GMM [14], k-means clustering [8], decision tree, SVM [6], [7] and GBAD [13] can be employed. In this case, the detector often comes with the ability to tackle new attacks but still requires some domain knowledge for feature extraction. More recently, some anomaly-based approaches that are realised using deep learning algorithms came [15], [16], [36]. These approaches almost remove the dependence on domain knowledge by taking advantage of deep learning algorithms' strong capability in representing features. However, due to the features being abstracted at a higher level, it is difficult for security analysts to track the attack through the audit date, which may result in alarm fatigue. Compared with existing approaches, the proposed approach is advanced with almost zero dependence on domain knowledge, and strong interpretability due to the nature of a corpus.

In addition to the above mentioned issues, many existing approaches are not able to work on multiple types of audit data straight away. Instead, they either provide only a detector that aims at a specific type insider attacks by

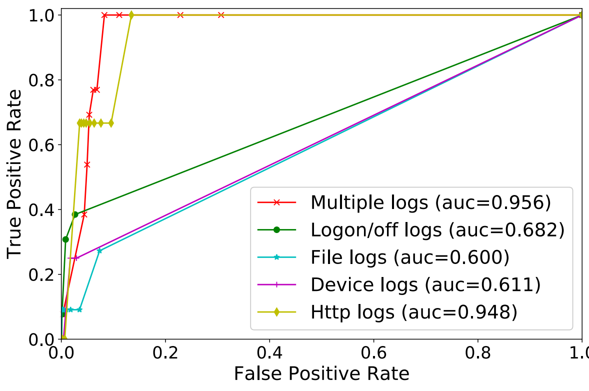


FIGURE 7. Performance comparison between multiple types of logs and single type of log.

analysing a single type of audit data, or need an orchestration to make a final decision with multiple sub-detectors. As a result, we leverage the first set of comparative experiments to demonstrate how the proposed approach performs when using single or multiple types of security logs. In particular, the detection is realised with *http*, *file*, *device* and *logon* logs respectively and all together. Figure 7 shows the best performances obtained for different cases. Apparently, in general, the case of using multiple types of security logs can produce the best result, namely the largest AUC, which can detect all the affected events with a low FPR equal to 8%. The second best one is the result from *http* log, which also detects all the affected events but the FPR rises up to 13%. But, the single *logon*, *file* or *device* log is unable to produce an acceptable performance.

Next, the proposed approach is compared with the other representative state-of-the-art deep learning based approaches [15], [16], [36]–[43] from a performance perspective. Technically, as mentioned earlier, the deep learning based approaches learn feature representations from a complex dataset, with do not require too much domain knowledge for feature engineering. These approaches tend to encode insider behaviours into a series of frequency vectors along the timeline, which are then employed as the features to be trained with a (deep) neural network. Following the similar idea, using the above-mentioned four types of security logs, we count the hourly frequencies of the activities (e.g., download, upload, copy, delete and etc.) for each user and each day, yielding a 264 dimensional feature vector space. The features are trained using a deep auto encoder and, for each feature vector, it will be labelled as suspicious if the reconstruction error is beyond a given threshold [15]. The results are shown in Figure 8. The autoencoder based approach can also obtain a TPR of 100% with a FPR slightly lower than 10%. However, on average, it takes around 1 hour for training, which is much longer than the training time required by the Word2bec based approach (normally 100 seconds as shown in Table 5). Moreover, once we engineer the features as numeric vectors, we have lost interpretability to some extent. Even if a feature vector that corresponds to a malicious insider's daily

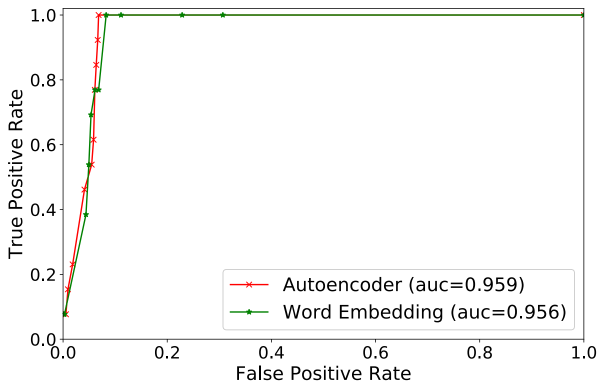


FIGURE 8. **Performance comparison between the Word2vec based approach and the autoencoder based approach.**

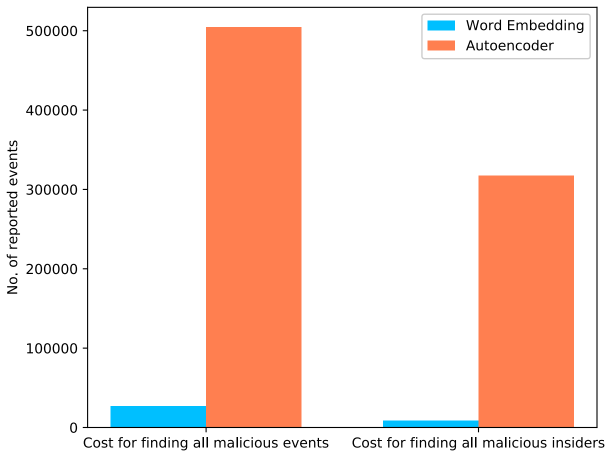


FIGURE 9. **Workload comparison between the Word2vec based approach and the autoencoder based approach.**

behaviours is properly labelled as suspicious by the autoencoder based approach, it is difficult for a security analyst to track what really happens along the attack vector. But, using the Word2vec based approach, firstly, we don't transform the texts into numeric vectors. Alternatively, the transformed events are still human-readable (i.e., the 4W sentence template). Secondly, the approach labels suspicious transformed events, which might further save security analysts time to locate what behaviours have been affected by an insider attack. Radically, the two approaches result in different workloads for security analysts to investigate. As shown in Figure 9, it clearly shows a comparison between the two approaches about the workload, where we use the number of the events potentially needed to be inspected as the metric of workload.

VI. CONCLUSION

In this paper, we propose a new approach to detect malicious insiders by assessing word similarities across multiple types of security logs. Events from any type of security log are transformed into an identical format using the *log2text* component based on a 4W sentence template, and the *text2corpus* component produces a Word2vec trainable corpus by arranging the transformed events by users and dates. After a Word2vec model is trained, the detection is performed on each user's daily behaviour by examining the model to determine which behaviours are unusual, where the likelihood of users' abnormal behaviours resembles the word similarities. Extensive numerical experiments provide best practice inputs for tuning the parameters and thresholds. They also demonstrate that the proposed technique is effective and scalable. The comparative study shows that our approach reduces the analysts' input in order to inspect suspicious insiders. In future, we intend to improve the approach as follows. Firstly, there is a potential to inject more information from the raw security logs into the transformed events. For example, in terms of web proxy logs, the HTTP user agent and category can be added into the 'what' word of the 4W sentence template. Secondly, the examination of the model to determine the likelihood of each insider behaviour is clumsy. Instead, we propose to design a phrase/sentence based examination mechanism to determine whether a transformed event is anomalous or not. Finally, in our current system we only train a Word2vec model using the security log corpus. However, by rearranging the corpus differently we should be able to take advantage of correlations among the 'paragraphs' in the corpus by using a Doc2vec model [44], resulting in a more efficient approach.

REFERENCES

[1] M. Collins, "Common sense guide to mitigating insider threats," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2016-TR-015, 2016.

[2] L. Liu, O. de Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1397–1417, 2nd Quart., 2018.

[3] Clearswift. (2017). *Clearswift Insider Threat Index (CITI)*. Accessed: Jun. 9, 2018. [Online]. Available: https://www.clearswift.com/about-us/pr/press-releases/insider-threat-74-security-incidents-come-extended-enterprise-not-hacking-groups

[4] A. Azaria, A. Richardson, S. Kraus, and V. S. Subrahmanian, "Behavioral analysis of insider threat: A survey and bootstrapped prediction in imbalanced data," *IEEE Trans. Comput. Social Syst.*, vol. 1, no. 2, pp. 135–155, Jun. 2014.

[5] M. A. Maloof and G. D. Stephens, "Elicit: A system for detecting insiders who violate need-to-know," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Berlin, Germany: Springer, 2007, pp. 146–166.

[6] E. Ted, H. G. Goldberg, A. Memory, W. T. Young, B. Rees, R. Pierce, D. Huang, M. Reardon, D. A. Bader, and E. Chow, "Detecting insider threats in a real corporate database of computer usage activity," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 1393–1401.

[7] W. T. Young, H. G. Goldberg, A. Memory, J. F. Sartain, and T. E. Senator, "Use of domain knowledge to detect insider threats in computer activities," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2013, pp. 60–67.

[8] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proc. 29th Annu. Comput. Secur. Appl. Conf.*, 2013, pp. 199–208.

[9] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," 2017, *arXiv:1702.08608*. [Online]. Available: https://arxiv.org/pdf/1702.08608.pdf

[10] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[11] N. Nguyen, P. Reiher, and G. H. Kuenning, "Detecting insider threats by monitoring system call activity," in *Proc. IEEE Syst., Man Soc. Inf. Assurance Workshop*, Jun. 2003, pp. 45–52.

[12] M. Hanley and J. Montelibano, "Insider threat control: Using centralized logging to detect data exfiltration near insider termination," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2011-TN-024, 2011.

[13] P. Parveen, J. Evans, B. Thuraisingham, K. W. Hamlen, and L. Khan, "Insider threat detection using stream mining and graph mining," in *Proc. IEEE 3rd Int. Conf. Privacy, Secur., Risk Trust, IEEE 3rd Int. Conf. Social Comput.*, Oct. 2011, pp. 1102–1110.

[14] Y. Song, M. B. Salem, S. Hershkop, and S. J. Stolfo, "System level user behavior biometrics using Fisher features and Gaussian mixture models," in *IEEE Security Privacy Workshops*, May 2013, pp. 52–59.

[15] L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang, "Anomaly-based insider threat detection using deep autoencoders," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2018, pp. 39–48.

[16] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," in *Proc. Workshops 31st AAAI Conf. Artif. Intell.*, 2017, pp. 224–231.

[17] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.

[18] B. Liu, Y. Xiao, P. S. Yu, Z. Hao, and L. Cao, "An efficient approach for outlier detection with imperfect data labels," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 7, pp. 1602–1616, Jul. 2014.

[19] B. Liu, Y. Xiao, L. Cao, Z. Hao, and F. Deng, "SVDD-based outlier detection on uncertain data," *Knowl. Inf. Syst.*, vol. 34, no. 3, pp. 597–618, 2013.

[20] J. Glasser and B. Lindauer, "Bridging the gap: A pragmatic approach to generating insider threat data," in *Proc. IEEE Secur. Privacy Workshops*, May 2013, pp. 98–104.

[21] B. Lindauer, J. Glasser, M. Rosen, K. C. Wallnau, and L. ExactData, "Generating test data for insider threat detectors," *JoWUA*, vol. 5, no. 2, pp. 80–94, 2014.

[22] K. Kent and M. Souppaya, "Guide to computer security log management," NIST, Gaithersburg, MD, USA, Tech. Rep. Special Publication 800-92, 2006, vol. 92.

[23] N. Sun, J. Zhang, P. Rimba, S. Gao, Y. Xiang, and L. Y. Zhang, "Data-driven cybersecurity incident prediction: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1744–1772, 2nd Quart., 2018.

[24] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 104–117, Jan. 2013.

[25] R. Coulter, Q.-L. Han, L. Pan, J. Zhang, and Y. Xiang, "Data-driven cyber security in perspective–intelligent traffic analysis," in *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2019.2940940.

[26] M. Xie, S. Han, B. Tian, and S. Parvin, "Anomaly detection in wireless sensor networks: A survey," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1302–1325, 2011.

[27] O. Brdiczka, J. Liu, B. Price, J. Shen, A. Patil, R. Chow, E. Bart, and N. Ducheneaut, "Proactive insider threat detection through graph learning and psychological context," in *Proc. IEEE Symp. Secur. Privacy Workshops*, May 2012, pp. 142–149.

[28] A. A. E. Ahmed, "Employee surveillance based on free text detection of keystroke dynamics," in *Handbook of Research on Social and Organizational Liabilities in Information Security*. Hershey, PA, USA: IGI Global, 2009, pp. 47–63.

[29] J. S. Valacich, J. L. Jenkins, J. F. Nunamaker, Jr., S. Hariri, and J. Howie, "Identifying insider threats through monitoring mouse movements in concealed information tests," in *Proc. Hawaii Int. Conf. Syst. Sci. Deception Detection Symp.*, 2013.

[30] S. Eberz, K. B. Rasmussen, V. Lenders, and I. Martinovic, "Looks like eve: Exposing insider threats using eye movement biometrics," *ACM Trans. Privacy Secur.*, vol. 19, no. 1, p. 1, 2016.

[31] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Commun. ACM*, vol. 55, no. 2, pp. 55–61, Feb. 2012.

[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*. [Online]. Available: https://arxiv.org/pdf/1301.3781.pdf

[33] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. Int. workshop Artif. Intell. Statist.*, vol. 5, 2005, pp. 246–252.

[34] C. Liu, "The Bayes decision rule induced similarity measures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1086–1090, Jun. 2007.

[35] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proc. LREC Workshop New Challenges NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[36] P. Chattopadhyay, L. Wang, and Y.-P. Tan, "Scenario-based insider threat detection from cyber activities," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 3, pp. 660–675, Sep. 2018.

[37] F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang, "Insider threat detection with deep neural network," in *Proc. Int. Conf. Comput. Sci.* Cham, Switzerland: Springer, 2018, pp. 43–54.

[38] S. Wen, M. S. Haghighi, C. Chen, Y. Xiang, W. Zhou, and W. Jia, "A sword with two edges: Propagation studies on both positive and negative information in online social networks," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 640–653, Mar. 2015.

[39] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Security*, vol. 15, no. 1, pp. 987–1001, Jul. 2019.

[40] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," *Comput. Secur.*, vol. 76, pp. 265–284, Jul. 2018.

[41] J. Jiang, S. Wen, S. Yu, Y. Xiang, and W. Zhou, "Identifying propagation sources in networks: State-of-the-art and comparative studies," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 465–481, 1st Quart., 2017.

[42] M. Xie, J. Hu, S. Han, and H.-H. Chen, "Scalable hypergrid k-NN-based online anomaly detection in wireless sensor networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1661–1670, Aug. 2013.

[43] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures," *ACM Comput. Surv.*, vol. 52, no. 2, p. 30, 2019.

[44] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1188–1196.