2018 IEEE Symposium on Security and Privacy Workshops

Detection of Masqueraders Based on Graph Partitioning of File System Access Events

Flavio Toffalini,* Ivan Homoliak,* Athul Harilal,* Alexander Binder,* and Martín Ochoa*†

* *ST Electronics-SUTD Cyber Security Laboratory*
Singapore University of Technology and Design
flavio_toffalini@mymail.sutd.edu.sg
{ivan_homoliak, athul_harilal, alexander_binder, martin_ochoa}@sutd.edu.sg

† *Department of Applied Mathematics and Computer Science*
Universidad del Rosario, Bogotá, Colombia
martin.ochoa@urosario.edu.co

***Abstract*—Masqueraders are users who take control of a machine and perform malicious activities such as data exfiltration or system misuse on behalf of legitimate users. In the literature, there are various approaches for detecting masqueraders by modeling legitimate users' behavior during their daily tasks and automatically determine whether they are doing something suspicious. Usually, these techniques model user behavior using features extracted from various sources, such as file system, network activities, system calls, etc. In this work, we propose a one-class anomaly detection approach that measures similarities between a history of a user and events recorded in a time-window of the user's session which is to be classified. The idea behind our solution is the application of a graph partitioning technique on weighted oriented graphs generated from such event sequences, while considering that strongly connected nodes have to belong into the same cluster. First, a history of vertex clusters is build per each user and then this history is compared to a new input by using a similarity function, which leads either to the acceptance or rejection of a new input. This makes our approach substantially different from existing general graph-based approaches that consider graphs as a single entity. The approach can be applied for different kinds of homogeneous event sequences, however successful application of the approach will be demonstrated on file system access events only. The linear time complexity of the approach was demonstrated in the experiments and the performance evaluation was done using two state-of-the-art datasets – WUIL and TWOS – both of them containing file system access logs of legitimate users and masquerade attackers; for WUIL dataset we achieved an average per-user AUC of 0.94, a TPR over 95%, and a FPR less than 10%, while for TWOS dataset we achieved an average per-user AUC of 0.851, a TPR over 91% and a FPR around 11%.**

***Keywords*—Insider threat, masquerader, anomaly detection, graph partitioning, Markov cluster, file system.**

I. Introduction

In a *masquerade attack* an attacker performs actions on behalf of a legitimate user of a system [25]. A masquerade attacker (masquerader) may be either an internal user of a system, such as a colleague of the victim (insider), or an external entity (outsider), both belonging into the identity theft problem. Consequences of a masquerade attack can be extremely severe, especially in the case of an inside masquerader, who can cause considerably higher damage to an organization than an external masquerader thanks to his major advantage over outsiders – the knowledge of the target system.

Although public cyber-security reports and surveys usually do not distinguish between internal and external cases of the identity theft problem, the CERT database [8] contains documented insider cases that involved masqueraders who caused an average financial loss of 40M$ in 2012. Regarding external cases of identity theft, *Verizon* indicated more than $9,000$ phishing incidents and around $1,400$ cases of credential theft in 2015 [28]. Therefore, the detection of masqueraders is an important research topic that requires new insights.

Modeling and analyzing masqueraders is particularly challenging because it involves detecting malicious behaviors performed by humans. Hence, it differs from other kinds of cyber-security issues such as malware or intrusion detection due to the unpredictable nature of human beings. Previous works in this domain proposed several solutions mostly focused on detection of masquerade behaviors using machine learning (ML) approaches [5], [18], [15], [13], [17], [25], [26], [30]. Among those, we can identify the two-classes/multi-classes techniques [5], [18], [15], [13], which require labeled malicious samples for the training phase, and therefore consider a certain knowledge of the attacking scenarios for their correct recognition. In contrast to two-classes/multi-classes techniques, there exist one-class approaches [17], [25], [26], [30], which do not require any malicious samples for training, and therefore are advantageous for anomaly detection of a wider range of masquerader behaviors [13], [25].

In this work, we propose a one-class approach for user-behavior modeling based on graph partitioning. Graphs and graph-theoretical metrics have been largely used in various domains to perform tasks ranging from software classification [21] to social network analysis [29], [2]. Usually, these strategies aim to label similar graphs according to some metrics based on generic features, locality features, isomorphism, graph edit distances, or they are used to model group of users according to their interaction [29], [2]. In the malware detection and classification, such techniques have been applied on a variety of system-interaction induced graphs, such as call-

DOI 10.1109/SPW.2018.00037

graphs and data-flow graphs [22], [32], [16]. In comparison to them, we propose to aggregate sequences of events generated by user's activities as graphs that model the *pairwise order* in which events have been executed. For instance, consider a software developer who interacts with source files in a repository, or an employee who works with a web application. In these cases, users will generate sequences of events forming logs that may be of interest when assessing their behavior: file system access log for a software developer or HTTP request log for an employee interacting with web application. These logs represent how a task has been performed by a user; a task is expressed by the concrete resources accessed and the order in which they have been accessed. We assume that part of these logs will be repeated in the future, perhaps with slight differences, due to the fact that users are likely to perform certain routine tasks in a similar fashion.

Therefore, we aim at identifying recurring routine tasks, and leverage on this knowledge to decide whether newly performed tasks are potentially anomalous. To achieve this goal we interpret time-stamped event logs representing resource interactions of users (*e.g.,* file-system activities, requests to URLs) as graphs where nodes are events/resources (*e.g.,* file paths, URLs) and directed edges indicate mutual order of occurrence of two consecutive events; for example, interaction with a file A immediately followed by interaction with a file B generates an edge from A to B.

We conjecture that our approach is agnostic to the nature of events as it does not extract any domain-specific features. However, in this work we will constrain ourselves to file system access events due to the availability of some masquerader-based state-of-the-art datasets containing such events. We assume certain time window of events, during which a user may concurrently perform several activities, thus we expect to find sub-graphs linked to specific tasks. When a user performs a task on a set of resources, he will generate strongly connected nodes; hence we expect to see similar structures of nodes when the task is repeated in the future. We call these sets of nodes *vertex clusters*. Using clustered graph representation, any two graphs can be compared and similarity score computed. For the purpose of graph partitioning, we use a method called *Markov Chain Cluster* (MCL) [27], which has been primarily employed in biomedicine (e.g., [9], [23]), but it has some applications in computer science as well (e.g., [2]). To the best of our knowledge, no one had employed MCL for masquerader detection yet.

Other works dealing with masquerader detection in file system access logs [11], [25], [6], [7] adopt feature extraction considering the tree structure of file system paths and further contextual information derived from it. In contrast to them, we do not consider the file system structure, instead we replace each unique event with a random token. This feature of our approach ensures high privacy of anonymized input data, as no partial information about a directory structure is exposed after anonymization, in contrast when anonymization has to preserve the sub-paths of file system's hierarchy.

Attacker Model: We assume several aspects of the masquerade attacker, regardless he is insider or outsider. First, we assume that masquerader has bypassed the existing authorization mechanisms, and he steals information or misuses a penetrated system itself. Second, we assume that a legitimate user performs routine tasks that are reoccurring and similar to some extent, while an attacker has different objectives, and therefore we assume he produces never-before-seen behaviors.

Problem Statement: We address the following question: Is it possible to design a generic approach for anomaly detection of masqueraders, which would achieve better results than existing ad-hoc one-class approaches, and moreover will it be comparable to the two-class ad-hoc approaches in terms of classification performance?

Contributions:

- **a)** We propose a generic one-class approach for user-behavior modeling that is based on graph-clustering of homogeneous user events and a comparison of such clustered graphs by similarity functions. The novelty of our approach is a combination of three contributions: 1) a sequence of events (including a history) is represented in a weighted oriented graph, capturing pairwise transition frequencies; 2) in order to account for partial matches between parts of both graphs (the history and the time-window-delimited events) are partitioned by Markov Clustering; this yields a set of vertex clusters such that strongly connected vertices are preserved in each cluster; 3) the similarity is defined over pairs of vertex clusters, one captured from the history and one from the events of an input time window. This makes our approach substantially different from existing general graph-based approaches that consider graphs as a single entity.
- **b)** We perform an evaluation of our approach on the WUIL dataset containing synthetically injected masquerade activities and obtain high classification performance.
- **c)** We perform a further evaluation of our approach on the TWOS dataset that contains masqueraders sessions performed by human subjects; we show that our technique performs well in detection of non-synthetic malicious activities as well.

The rest of the paper is structured as follows: In Section II, we provide the formal description of our approach. Section III presents an empirical evaluation of our solution, experiments aimed at practical aspects. In Section IV we discuss limitations and possible improvements of the approach. We state the related work in Section V, and finally conclude the paper in Section VI.

II. Approach

We posit that common user's tasks (such as software development or using a web-based applications) involve activities in a host machine, which are executed according to certain user specific patterns. These activities can be represented by finite sequences of homogeneous events ES:

$$ES = (e_1, \ldots, e_n), \quad n \in \mathbb{N}. \tag{1}$$

It means that ES is formed using events from the only domain, for instance it can be either file system access events or HTTP request or SQL queries etc. In the case of file system access events, each e_i is a file path, and in the case of HTTP requests, each e_i is a URL. In our approach, we aim at modeling and recognizing patterns that occur across such sequences of events, regardless of the nature of the

event. Starting from this assumption, we expect that legitimate users who perform similar tasks will produce similar patterns in sequences of considered event-space, while in contrast a masquerade attacker has other goals than legitimate user, and therefore we expect him to perform different tasks leading to different patterns than in the case of legitimate user.

To achieve this goal we define and develop a supervised anomaly-based one-class technique, which employs a *similarity function* for estimating similarities among sequences of events, while it aims at comparison of recurring patterns. We decided to adopt one-class approach, assuming only normal users' samples as labeled, due to the following reasons: a) in general, it is hard to obtain real-world malicious labeled data corresponding to user's behavior; and b) even if we would use synthetic malicious labeled data, there is no guarantee that they will cover all sorts of masquerade attacks.

A. *Similarity Function*

The *similarity function* that we conceive returns a score between *0* and *1*, and it compares:

1) the previous interactions of a legitimate user u (referred to as *history* H_u), with
2) any sequence of consecutive events occurred in a specific time window (ES_w).

Formally, it can be declared as:

$$similarity(H_u, ES_w) \rightarrow s_u, \tag{2}$$

where similarity value $s_u \in [0, 1]$ is a real number with the following interpretation:

(a) $\mathbf{s_u} \cong \mathbf{0}$ denotes a low similarity between the input sample ES_w and the history H_u of user u, which may indicate possible action by a masquerader.
(b) $\mathbf{s_u} \cong \mathbf{1}$ denotes a high similarity between the input sample ES_w and the history H_u, which indicate ES_w as a legitimate behavior of user u.

Note that it is possible to obtain a high similarity value when a user y perform the same tasks as user u, and his sequence of events is similar to the history of user u (*i.e.,* $similarity(H_u, ES_y) \cong 1$). This is acceptable as we deal with one-class problem that is equivalent to user authentication, not user identification, which is multi-class problem.

A per-user threshold $t_u \in [0, 1]$ is used for making a decision about acceptance of ES_w as behavior of legitimate user or rejection it as masquerade attack:

(a) if $\mathbf{s_u} \geq \mathbf{t_u}$, then accept ES_w as legitimate behavior,
(b) if $\mathbf{s_u} < \mathbf{t_u}$, then reject ES_w and consider it as attack.

In this approach, it is crucial to choose the suitable threshold in order to maximize TPR and at the same time keep FPR low. We will discuss some statistical approaches to face this problem in Section III, while in the rest of the current section, we will discuss in detail the adopted graph models, similarity functions for them and the way how we identify similar patterns in the history H_u.

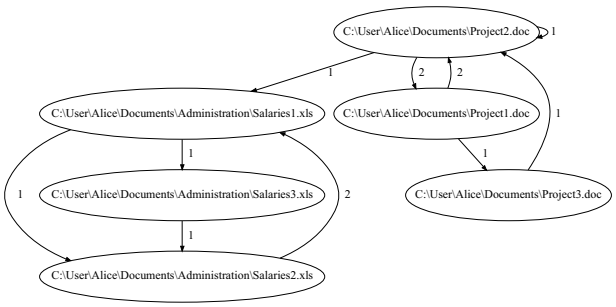

Fig. 1: An example of graph model.

a) ***Graph Model****:* We propose a semantic model in which we represent a sequence of user's events by cyclic oriented weighted graphs. Starting from a finite sequence of user's events (ES), we build an oriented graph such that its vertices are all unique events occurred in ES and each edge represents a transition between two consecutive events. Also, each edge has associated weight according to a number of repetitions of particular transition in ES. It is important to remark that each edge has a direction, therefore a transition from event e_1 to event e_2 is different than one from e_2 to e_1, and thus it will result into two edges with opposite directions. An example of graph model built on top of a sequence of file system access events is depicted in Figure 1.

b) ***Similarity Measurement between Graphs****:* Since similar tasks produce similar events, we expect to find these similarities also in the graphs built on top of them. In the literature, there are several approaches for measuring similarities between graphs [4], [10], however most of them are complex for application in real contexts.

In this work, we propose a novel technique for measuring the similarity between two graphs built on top of user's event sequences, that can be synthesized in two main points:

1) graph partitioning: at first, we cluster all vertices of each input graph by MCL algorithm in such a way that each cluster contains vertices that are strongly connected (*i.e.,* they contain a lot of edges among them) and refer to them as *vertex clusters*. At the output of this step, we obtain set of vertex clusters.
2) Computing of similarity score: We perform similarity measurements between the vertex clusters obtained at step 1) and the vertex clusters contained in the history H by several proposed similarity functions and for each of them obtain a normalized score in the interval $[0, 1]$.

Each point will be discussed in the following.

B. *Graph Partitioning*

The idea behind this approach is that a task tends to show similar interaction patterns every time it is performed. Those patterns will look like vertices strongly connected as there is a lot of transitions among them. For splitting the graph according to this idea, we applied the MCL [27]. The MCL algorithm takes as an input a graph and returns several sets of vertices that are strongly connected (*i.e.,* vertex clusters). With that in

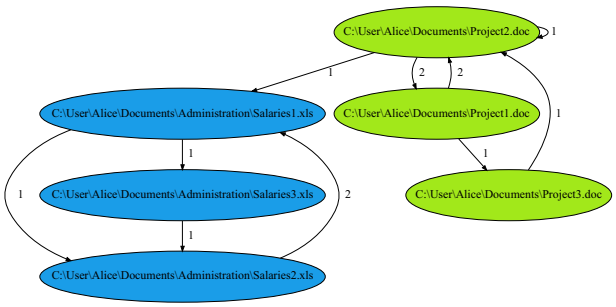

Fig. 2: An example of clustered graph model.

mind, let us return to the example with file system interaction, where we extract vertex clusters now (see Figure 2).

*a) **History Definition**:* Once graph partitioning is introduced, we can define the history of user's events. The history of events of user u is modeled as a set of vertex clusters consisting of vertex clusters V, edges E and function ω assigning weights to edges :

$$H_u = (V, E, \omega), \tag{3}$$

$$V = \{\{v_a, v_b, v_c\}, \{v_c, v_d, v_a\}, \{v_b, v_e\}, \dots\}, \tag{4}$$

$$E = \{(v_x, v_y) |\ if\ v_x\ has\ edge\ to\ v_y\}, \tag{5}$$

$$\omega(E) = \{e \leftarrow n \mid e \in E, n \in \mathbb{N}_0\}, \tag{6}$$

where each vertex cluster from V represents a task consisting of user's interactions. The process of constructing the history H_u is described in Algorithm 1. In general, it may include a number of sequences of user's events ES and there exist various options for the selection of the best list of ES (however a detailed discussion will take place in Section III).

Algorithm 1 Building the History of User's Events

```
1: procedure CREATEHISTORY(list of ES)
2:     H_u ← {}
3:     for all es ∈ list of ES do
4:         g ← makeGraph(es)
5:         set of vertex clusters ← clusterGraph(g)
6:         for all s ∈ set of vertex clusters do
7:             H_u ← H_u ∪ s
8:     return H_u
```

C. *Computing of Similarity Score*

In this section, we discuss how we measure the similarity between a sequence of user's events ES_w delimited by time window w and history H_u of user u. Because the sequence of events has been clustered using MCL, we need to compare vertex clusters. For this purpose, we start with basic operations from the set theory (equality, superset, and subset [14]) and will continue with their modifications and combinations.

The main idea of the similarity score computation of vertex clusters in ES_w is computing a ratio between the number of vertex clusters marked as *legitimate* and the total number of vertex clusters in ES_w. The vertex clusters are marked as legitimate according to match with the history H_u. We divide proposed similarity functions into two groups: 1) *Not-Weighted Similarity Functions*, which contains application of elementary set operations, and 2) *Weighted Similarity Functions*, which additionally to the previous group adopt weighting. Since the particular functions of each group share the similar structure, we describe all functions by one pseudo-code for each group.

*a) **Non-Weighted Similarity Functions**:* The functions that belong to this group apply a comparison operator between the vertex clusters extracted from ES_w and the vertex clusters contained in the history H_u, and then they return the ratio between the number of matches and the total number of vertex cluster in ES_w (see Algorithm 2). The *comparison* operators that we use are: *Equality*, *Subset*, *Superset*, and the *Logical OR* of Subset and Superset. It is straightforward to verify that if ES_w does not have any vertex cluster matching the history, then the function returns *zero*. Contrary, if all vertex clusters of ES_w match the history, it returns *one*. The particular algorithms which belong to this group are:

- Similarity by Equality,
- Similarity by Subset,
- Similarity by Superset,
- Similarity by Logical OR of Subset and Superset.

Algorithm 2 Non-Weighted Similarity Function Template

```
1: procedure SIMILARITYNOTWEIGHTED(H_u, ES_w)
2:     g ← makeGraph(ES_w)
3:     set of vertex clusters ← clusterGraph(g)
4:     m ← 0
5:     for all s ∈ set of vertex clusters do
6:         for all h ∈ H_u do
7:             if comparison(s, h) then
8:                 m ← m + 1
9:                 break
10:    return m/ |set of vertex clusters|
```

*b) **Weighted Similarity Functions**:* The functions belonging into this group are an extended version of some functions from the previous group. While in the previous group we check whether a vertex cluster matches at least one vertex cluster from the history by a *comparison* function, in this case we weight each match by measuring the ratio of common elements between each of the two vertices clusters (see Algorithm 3).

Finally, we return an average value of all weights normalized by the number of vertex clusters found in ES_w. If all vertex clusters do not match the history at all, then the function

Algorithm 3 Weighted Similarity Function Template

```
1: procedure SIMILARITYWEIGHTED(history H_u, ES_w)
2:     g ← makeGraph(ES_w)
3:     set of vertex clusters ← clusterGraph(g)
4:     m ← 0, n ← 0
5:     for all s ∈ set of vertex clusters do
6:         for all h ∈ H_u do
7:             if comparison(s, h) then
8:                 m ← m + ratioOfCommonElements(s, h)
9:                 n ← n + 1
10:    m_avg ← m/n
11:    return m_avg/ |set of vertex clusters|
```

returns *zero* (the worst case). On the other hand, if all vertex clusters match the history, then the function returns *one* (the best case). The functions of the current group are:

- Weighted Similarity by Subset,
- Weighted Similarity by Superset.
- Weighted Similarity by Logical OR of Subset and Superset.

D. Time Complexity Analysis

In order to deploy a detection system in a real environment, it is important that its time complexity is linear or at least polynomial. A generic time complexity schema that is common for all proposed similarity functions is present in Algorithm 4. More precisely, we model our detector as a function that takes as an input the history of a single user (H_u) and a list of events to classify (ES_w). First, the event list ES_w is transformed to a graph (line 2), which is further transformed to a *set of vertex cluster* (line 3). Finally, all vertex clusters are iteratively compared with all elements in the history H_u by a `compare()` function (line 4 to 6), whose implementation depends on a particular similarity function.

The first two steps are common to all similarity functions. In detail, line 2 refers to construction of a graph from a list of events, which can be achieved in linear time by using an adjacency list implemented as a dictionary. Formally, we can state that this step has a linear complexity in time *w.r.t.,* the number of events (*e.g.,* $O(|ES_w|)$). The step at line 3 refers to a graph partitioning, which is implemented by *Markov Chain Cluster* (MCL), having a time complexity of $O(N \times k^2)$ [27], where N is the number of nodes in the graph (namely $|g|$) and k is the parameter of the algorithm, which is a constant. The last part of the similarity function (lines 4 to 6) is a comparison between all elements of the history and the vertex clusters obtained from ES_w. Each comparison is an implementation of *equality*, *superset*, or *subset* operations, which all have linear complexity in time (more precisely $O(min(|s|, |h|))$). At this point, we want to find a relation between the set of vertex clusters and the input ES_w. By definition, a *set of vertex clusters* is an object that simply groups all vertices in g, therefore, the number of single vertices in a *set of vertex clusters* is the same as the number of vertices in the graph g. Intuitively, all elements of the history H_u are compared to all vertices of the graph g. The size of H_u does not change after the training phase; we also tried to build the history by using different amount of events and in all cases we observed that after a while the history reached a fixed size because users' actions were repeated. On the other hand, the size of $|g|$ can be equal to the size of the ES_w in the worst case when all events in ES_w are unique. In general, we can state that $|g| \leq |ES_w|$.

Algorithm 4 Time Complexity of a Similarity Function

```
1: procedure SIMILARITYF(history H_u, sequence of events ES_w)
2:     g ← makeGraph(ES_w)
3:     set of vertex clusters ← clusterGraph(g)
4:     for all s ∈ set of vertex clusters do
5:         for all h ∈ H_u do
6:             compare(s, h)
```

In summary, we argue that our technique has on average a linear time complexity, which depends by the size of the history (that is fixed after some time) and the number of input events ($|ES_w|$).

III. Evaluation

As we have already outlined, we have measured the performance of our approach on homogeneous user's events that are instantiated by file system access events. In the literature there are many approaches that aim at classification of user's behavior using file system access logs [5], [11], [20], [31]. These approaches consider domain specific properties of tree-based structure of file system paths (*e.g.,* distance, common path, frequency of accessing some paths, file types, previous access events etc.), while our approach is more generic and does not consider such contextual information, and thereby one may presume that it could potentially lead to worse performance results. However, we will show that we can achieve better results in comparison to existing one-class approaches that consider contextual information of a file system for computation of ad-hoc features, and moreover we will show that results obtained are comparable to more-informed two-class state-of-the-art approaches.

First, we briefly describe two datasets selected for our evaluation: WUIL dataset [6], and TWOS dataset [12]. Both datasets provide information about file system access events. In the second part of this section, we focus on evaluation of our approach on these datasets; we start with the analysis of applicability of our graph partitioning methods for user behavior modeling, and then continue with the evaluation itself. At the end, we discuss some practical aspects for deployment of our classifier into a real environment as well as some limitations. Note that throughout all of our experiments, we examined the sequences of file system access events considering three different time windows that are specific for each dataset (30 seconds, 1, and 2 minutes for WUIL; and 10, 20, and 30 minutes for TWOS). We selected those time windows based on the previous papers and properties of the datasets, which will be discussed later.

A. WUIL Dataset

The WUIL dataset has been designed and implemented by Camiña et al. [6], [7] in 2014 with the purpose of providing a valid support for studying masquerade insiders. In this work, we refer to the latest version of the dataset [5], which was kindly shared with us by the authors in December 2016.

The WUIL dataset contains records from 76 volunteer users, who had been monitored at different periods of time during routine daily activities. Therefore, some users may have around 1 hour of logs, while others several weeks. The data have been collected using an internal tool for file system audit on Windows machines [1]. Data collection ran on different versions of Windows OS (*e.g.,* XP, 7, 8, and 8.1). While the legitimate users' data had been collected from real users, the masquerade sessions had been simulated using batch scripts considering three skill levels of users: basic, intermediate, and advanced. Basic attackers are designed as users without technical background who manually look for files and perform exfiltration through common office-like software (*e.g.,* browser,

email client). Intermediate attackers represent those users who can look for files using internal Windows search file tool (*e.g.,* looking for a particular extension or key words) and use also thumb-drives to stole information. Advanced attackers can develop scripts that automatically look up for information and export them to a thumb drive or over the Internet. Per each type of masquerader, the authors compromised each legitimate user by 5 minutes long malicious sessions, which yielded in total 228 (3×76) masquerade sessions. The data of WUIL dataset are file system logs and each line of the dataset represents a generic file interaction regardless of its type (*e.g.,* open, write, read). Each line contains several items, while the most relevant to us are the *timestamp* and the *path*.

B. *TWOS Dataset*

The TWOS dataset has been designed and implemented by Harilal et al. [12] in 2017 with the purpose of providing realistic instances of two type of insider threats – masqueraders and traitors. Since we focus on masquerader detection only, we briefly describe and further use for evaluation only data related to masqueraders. The dataset is the outcome of a game designed to reproduce interactions in real companies, while stimulating existence of insider threat. The game involved 24 users, organized into 6 teams that played for an entire week. Each team was meant to simulate a sale department that was aimed at collecting points by dealing with virtual customers, which is considered as a legitimate activity. In contrast, masquerade sessions were performed by "*temporarily*" malicious users, who, once upon a time, received credentials of another users (victims) and were able to take control over victim's machines for a period of 90 minutes. During this time a victim lose control of his/her machine, while an attacker could steal victim's points or sabotage a victim's machine. In summary, 12 masquerader sessions were generated during the game, each affected a unique user, and lasted for 90 minutes.

From the implementation point of view, the authors used cloud environment and assigned virtual machine (VM) running Windows 10 to each participant. Each VM had installed Mozilla Firefox browser and several applications from MS Office suite (i.e., Word, Excel, Outlook). Using a few data collection agents, the authors collected miscellaneous types of datasets such as mouse, keyboard, network, and host monitor logs of system calls. For the purpose of this work, we use only host monitor logs that contain file system access events (i.e., open, read, write, close), involving access to application binaries when they are executed or terminated.[1] The difference between TWOS and WUIL is that malicious activities in TWOS were performed by human subjects, while in WUIL, they were synthetically simulated by batch scripts.

C. *Preliminary Analysis: Inverted Index*

The first experiment we performed was a test that examined whether clustering is capable of modeling user's behavior based on file system access events.

We decided to perform all of our experiments with three already mentioned time windows for each datasets. In the WUIL dataset, attacker session is around 5 minutes long, hence making the window equal or longer than 5 minutes would simplify the problem. However we preferred to keep the task challenging and comparable to other works [6], [5],[2] therefore we considered shorter time windows than 5 minutes. In the case of the TWOS dataset, attacker's sessions were 90 minutes long, and the attack was performed by a human subject who has non-uniform behavior that cannot be sufficiently captured by 1 or 2 minutes time windows. Also, we cannot assume that all actions within the masquerade sessions were malicious. With the previous in mind, we decided to use larger time windows for the TWOS dataset.

TABLE I: Average ratio of *gray* zone in WUIL and TWOS.

	Time Window	Gray-Zone
WUIL	30s	1.83%
	1m	1.82%
	2m	1.78%
TWOS	10m	1.26%
	20m	0.65%
	30m	0.74%

Firstly, we made an inverted index over the vertex clusters generated from both legitimate users and masqueraders of both datasets, and then we executed following steps for each user:

1) Split file system access events according to a time window (30sec, 1min, and 2min for WUIL; and 10min, 20min, and 30min for TWOS).
2) Build a graph for each sequence delimited by a specific time window.
3) Generate vertex clusters in each graph.
4) Make the inverted index: for each cluster, list all graphs that contain the cluster.[3]
5) Color all vertex clusters with three colors:
   a) **blue**: when a vertex cluster was present in legitimate graphs only,
   b) **red**: when a vertex cluster was present in attacker graphs only, and
   c) **gray**: when a vertex cluster was present in both legitimate and attacking graphs.

Our simplified assumption was that graphs built from attacking versus legitimate behaviors will contain different vertex clusters, which is sufficient but not necessary condition for distinguishing between these two behaviors. Therefore, we expected that the ratio of gray vertex clusters should be small, but not necessarily zero. Table I lists average ratios of gray vertex clusters that were obtained using three different time windows. We can see that the *gray zone* between legitimate and malicious sessions is very small on average. Also, we can observe a slight decrease with growing time window. This observation could be explained by the assumption that some important patterns need longer time period to capture them.

In this experiment, we demonstrated that our technique has a potential for distinguishing between malicious and legitimate behaviors in the majority of the cases, considering the assumption that *"graphs built from malicious and legitimate behaviors contain primarily disjoint vertex clusters."* This can

[1]Note that raw host monitor events were aggregated to reduce bulky read and write system calls.

[2]For comparison, Camiña et al. used only 30s as a time window [6].

[3]The step was performed by using *Similarity by Equality* function.

TABLE II: Average AUCs per each dataset.

Dataset	Time Window	Mean AUC	Std. Dev. AUC
WUIL	30s	0.916	0.066
	1m	0.937	0.058
	2m	0.944	0.050
TWOS	10m	0.771	0.126
	20m	0.805	0.177
	30m	0.851	0.132

be further improved by a soft comparison of two input graphs by a similarity function. In the following, we will measure the properties of the proposed similarity functions.

D. *Performance Evaluation*

The evaluation of our approach was performed using all described similarity functions in combination with three different time windows for each dataset: 30 seconds, 1, and 2 minutes for WUIL; and 10, 20, and 30 minutes for TWOS.

First, we analyze how different time windows influence the performance of our approach, then we perform various experiments aimed at performance evaluation of the proposed similarity functions, including ROC curves, AUCs, and comparison of several known strategies for selection of the best operational points on ROC curves.

*a) **Comparison of Time Windows**:* Table II shows average AUCs obtained over all users using the best performing similarity function – *Weighted Similarity by Logical OR of Subset and Superset*. The results were obtained using three different sizes of time windows per each dataset. The best values of AUC were achieved for the longest considered time windows in both datasets – 2 minutes in the case of WUIL and 30 minutes in the case of TWOS. In this experiment, we show the results only for a single similarity function, nevertheless the trend is the same for all proposed similarity functions.

*b) **Comparison of Similarity Functions**:* In this experiment, we compare the performances of all proposed similarity functions working with the best time windows found in the previous experiment (see Figure 3). In both datasets, the best performing similarity functions showed to be *Weighted Similarity by Superset* and *Weighted Similarity by Logical OR of Subset and Superset*. In the case of WUIL, these similarity functions achieved average AUC equal to 0.937 and 0.944, respectively, while in the case of TWOS, AUCs of those similarity functions were equal to 0.844 and 0.85, respectively. The best results for these two similarity functions can be explained due to the normal user's interaction with the system is not always the same over the time – once upon a time, a user may require to access a new file not seen in the training phase. Therefore, simple similarity functions, such as *Equality*, are too strict to capture such behaviors. As we will see in the further paragraphs, *a voting system* is needed in order to achieve a good balance between TPR and FPR; such voting system can be implemented by *e.g., Logical OR of Subset and Superset with Weigh* similarity function. Another interesting measurement displayed in the graph is the standard deviation, which highlights that the most stable results are obtained with *Logical OR of Subset and Superset with Weight* and *Superset with Weight*, while other similarity functions, such as *Superset*, look less stable in terms of per-user AUC.

*c) **ROC Curves**:* For the purpose of ROC curves generation, we selected *Logical OR of Subset and Superset with Weight* similarity function, and we varied per-user threshold through its full range. Obtained per-user ROC curves were then averaged across all users, for both datasets. The resulting mean ROC curves are depicted in Figure 4. Again, we can see that in the case of WUIL, we achieved better operational points across the full range of the detection threshold than in the case of TWOS. We attribute this to the easier detection of synthetically injected masquerader attacks of WUIL in contrast to attacks performed by real users in the case of TWOS.

*d) **System Tuning**:* As we mentioned in Section II, choosing the correct per-user threshold is crucial to detect the majority of malicious behaviors, while simultaneously keeping FPR low. Note that choosing these thresholds is not part of the training, but rather part of experimental evaluation intended to demonstrate the best performance a configuration of the approach can reach. In the literature, there exist several metrics for selection of the best configuration in ROC curve. In this work, we consider two metrics mostly used in medical applications [24]: *Younden index* and *the closest point*; and two metrics from machine learning [19]: *accuracy* and F_1-*measure*.

Before we introduce the metrics themselves, we define *sensitivity* and *specificity* for configuration c_t of the approach, where t represents the threshold:

$$\text{recall}(c_t) = \text{TPR}_{c_t} = \frac{\text{TP}_{c_t}}{P}, \quad (7)$$

$$\text{specificity}(c_t) = \text{TNR}_{c_t} = \frac{\text{TN}_{c_t}}{N}, \quad (8)$$

$$\text{precision}(c_t) = \frac{\text{TP}_{c_t}}{\text{TP}_{c_t} + \text{FP}_{c_t}}. \quad (9)$$

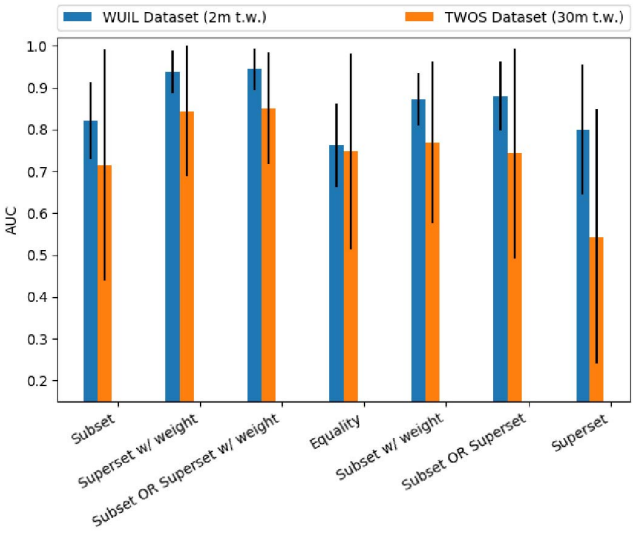

Fig. 3: Comparison of different similarity functions. The length of the lines at the top of the bars is equal to two times of standard deviation.

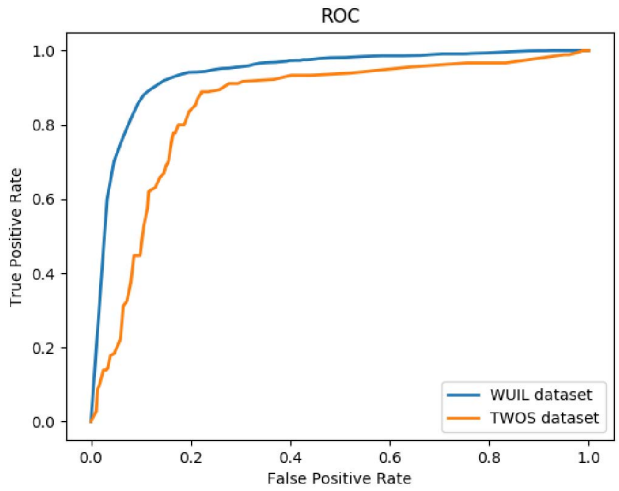

Fig. 4: Mean per-user ROC curves.

The *Youden index* is defined by the vertical distance between a point on the ROC and the diagonal connecting points $(0, 0)$ and $(1, 1)$. We are interested in configuration c_t parametrized by threshold t having the largest *Younden index*:

$$c_t = \arg\max_t \left[\text{recall}(c_t) + \text{specificity}(c_t) - 1\right]. \quad (10)$$

With the closest point technique, we compute the euclidean distance from the closest point on the ROC to the point $(0, 1)$, which is represented by finding the solution with the lowest distance from that point:

$$c_t = \arg\min_t \left[\sqrt{(1 - \text{recall}(c_t))^2 + (1 - \text{specificity}(c_t))^2}\right]. \quad (11)$$

Accuracy is defined as the sum of true positives and true negatives over the sum of all instances, while we seek for solution which maximizes it:

$$c_t = \arg\max_t \left[\frac{\text{TP}_{c_t} + \text{TN}_{c_t}}{\text{N} + \text{P}}\right]. \quad (12)$$

F_1-*measure* is a metric which balances the precision and recall of the malicious class. The sought configuration maximize F_1-*measure*:

$$c_t = \arg\max_t \left[\frac{2 \times \text{recall}(c_t) \times \text{precision}(c_t)}{\text{recall}(c_t) + \text{precision}(c_t)}\right]. \quad (13)$$

For each combination of time windows, validation strategies and similarity functions, we have computed defined metrics and chose the best configuration. Table III shows the best solutions found using particular performance metrics; the table displays average per-user values of the metric itself, TPR, and FPR. Note that accuracy is not suitable performance metric for unbalanced datasets as it favors the class with the superiority in numbers, which is in our case legitimate one. This experiment shows that our classifier performs similarly for both datasets. In the case of TWOS, TPR and FPR are slightly worse than in the case of WUIL. This can be explained by the nature of TWOS masquerade sessions that are performed by real users, and therefore they look more similar to legitimate actions, while in WUIL they are solely synthetic.

TABLE III: Various performance measures.

WUIL Dataset				
Metric	Configuration	Value	TPR	FPR
Younden i.	Logical OR w/ Weight	0.86	95.3%	9.4%
Closest p.	Logical OR w/ Weight	0.12	93.7%	8.5%
F_1-*measure*	Logical OR w/ Weight	0.30	74.7%	3.8%
Accuracy	Logical OR	0.98	19.6%	1.2%
TWOS Dataset				
Metric	Configuration	Val.	TPR	FPR
Younden i.	Logical OR w/ Weight	0.80	91.3%	11.7%
Closest p.	Logical OR w/ Weight	0.18	89.3%	10.2%
F_1-*measure*	Logical OR w/ Weight	0.45	68.7%	5.4%
Accuracy	Logical OR	0.98	42.7%	1.5%

TABLE IV: Time measurements of the MCL algorithm.

WUIL Dataset					
Time Window	**Mean**	**Median**	**Standard Deviation**	**Max**	**%samples > t.w.**
30s	0.013s	0.015s	0.21	91.0s	6.69×10^{-4}
1m	0.015s	0.015s	0.46	123.44s	1.79×10^{-3}
2m	0.020s	0.016s	0.72	160.55s	1.24×10^{-3}
TWOS Dataset					
Time Window	**Mean**	**Median**	**Standard Deviation**	**Max**	**%samples > t.w.**
10m	0.013s	0.012s	0.01	0.58s	0
20m	0.013s	0.012s	0.01	0.60s	0
30m	0.016s	0.011s	0.02	0.62s	0

E. *Toward a Practical Deployment*

In order to deploy our approach in the real-world, we need to study some practical aspects of the solution. In particular, we are interested in time profiling of the MCL algorithm that is the bottleneck of our approach.

*a) **Time Profiling**:* By profiling our detection mechanism, we realized that the bottleneck of our implementation lays in the MCL algorithm. Therefore, we focused on analysis of how this part slows down the detection in a run time scenario. The measured results are present in Table IV for both datasets. The data shown in the table are related to clustering time of time-window-delimited samples and include: mean, median, standard deviation, maximum, and the percentage of samples with a clustering time larger than a time window itself (for understanding the extreme cases). Looking at Table IV, it is possible to see that an average time that the MCL algorithm needs to extract vertex clusters from a graph is several orders of magnitude lower than time window itself in both datasets, and moreover we observe that only very few samples need long clustering time (see Max and Standard Deviation columns). In particular, those samples belong to WUIL dataset only, which is caused by the fact that synthetic attacks executed by batch scripts generated much more file system events than is common for human users. On the other hand, all events in the TWOS dataset were generated by human users, hence the clustering time is more stable (see low values of standard deviation in the table). Based on the above, it means that if we would consider distributed architecture deployed on each monitored endpoint, then it will be possible to perform graph partitioning

at almost[4] real-time together with the audit. This experiment was executed on a Windows 10 machine with an Intel Xeon CPU E5-1660 v4 @ 3.2GHz.

IV. Discussion

As we already mentioned, masqueraders include insiders as well as outsiders, while our proposed approach is not specifically intended for any of those cases, but is more general. However, for the evaluation we selected datasets that were collected with the intention of capturing inside masqueraders' behaviors. On the other hand, there are only a few aspects that differentiate insiders' behaviors in these datasets from potential outsiders' behaviors – such as using thumb drives (in WUIL), or knowing the format of file with secret information (in TWOS). Therefore, it would be interesting to evaluate our approach on more insider-oriented datasets and/or to compare it with experiments on outsider-oriented datasets.

In our experiments, we showed the necessity to reduce FPR of our approach, which is a common problem in one-class approaches. In our technique, false positives occur because legitimate vertex clusters are often too different from the history. We can explain it by two reasons: **1)** a vertex-cluster can include never-before-seen vertices – *e.g.,* new files are created or moved, **2)** history does not fit well – *e.g.,* a user "*switches his/her context*" to another task that is substantially different than the previous one (a.k.a., concept drift problem). It is possible to address each of these points using specific techniques: for **1)** we can adapt the vertices in the history once a file is moved or created; and for **2)** it is possible to train the system with only such vertex clusters that are necessary to accomplish a task. However, the former is not always possible to address because it depends on the nature of the dataset – for instance WUIL dataset does not provide enough information. In the case of TWOS, it would be possible but our intention was rather to evaluate the general implementation of our technique. The second FPR reduction technique cannot be performed neither in WUIL nor TWOS, because the authors of WUIL dataset did not provide detailed information about the tasks performed by particular users, while in the case of TWOS, participants were free in the way how they executed their tasks (even hacking their own machines was possible).

According to our attacker model, an attacker may attempt to perform an adversarial attack by mimicking a legitimate user. This can be achieved by crafting fake resources, for example the same filenames having different content. However, adversarial attacks are nontrivial to perform for two reasons: 1) an attacker has to know which resources are required for a specific task, and 2) successful attack requires a set of actions that follow a specific goal, therefore even if the resources look similar to legitimate ones, the attack itself will generate different graphs as the task is different as well.

Finally, we try to discuss the performance of our approach in a real deployment, considering for instance, configuration with 95% TPR and 10% FPR. In this case, an hour of legitimate user's work will contain around 6 minutes marked as malicious. On the other hand, one hour of masquerade attack will contain around 57 minutes of suspicious activities. For alleviation of FPR in the real world scenarios, there exist techniques that accumulate the amount of raised alarms during specified time interval that is longer than time window, and if this amount would reach a threshold, then an alarm notifying an operator would be raised. We showed how this technique helps almost to halve FPR from 11% to 6% in the case of TWOS. On the other hand, this technique makes our detection system more suitable for forensic analysis than online detection, since we need more time for rising an alarm.

[4]It is necessary to account for a size of the time window.

V. Related Work

We divide related work to studies that apply graph analysis for detection of malicious behavior and studies related to masquerade detection in file system access logs.

a) ***Graph Approaches****:* Several graph-theoretical techniques have been used in malware detection and classification. For instance, Anderson et al. [3] used features extracted by graph representation of system calls. Park et al. [22] used graph similarities for matching malware by sub-graphs, leveraging isomorphism algorithms. Other works [16], [32] propose several metrics on data-flow graphs extracted from malware and goodware for the classification purpose. In comparison to the above: 1.) we do not rely on labeled data for the training phase, since we use a one-class approach; 2.) we propose novel similarity matching algorithms in sub-graphs identified by the state-of-the-art clustering algorithm MCL [27].

b) ***Masqueraders & File System Behavior****:* When introducing WUIL, Camiña et al. [6] also showed some examples of detection using SVM and K-NN as one-class classifiers. Another approach was proposed by Camiña et al. in [7], where the authors use two kinds of abstraction: either considering the entire file path or the last folder that contains the file. The authors build n-grams according to time windows, and then they compute a similarity score using Markov Chain and Naive Bayes as one-class classifiers. The previous two works can be compared with our approach as they are one-class approaches (respecting anomaly detection), however we differ in the principles of the algorithm used, and moreover achieve better results. In the most recent work proposed by Camiña et al. [5], the authors introduced a new set of features leveraging "temporal" and "spacial locality." These are based on concepts such as distance between paths, frequency of access, and direction of path traversal. The authors experimented with TreeBagger classifier from MATLAB, which in comparison to our approach, works with two classes, hence it is more informed and does not respect anomaly detection principles in the training stage. Wang et al. [31] proposed a custom standard deviation computed over paths usually traversed by users. This customized standard deviation is based on distances between paths and common parts between paths, and is computed for a user's history. When it reaches a threshold an alarm is raised. Note that this approach is a one-class classifier that is based on domain-specific path features, and therefore differs from our generic technique. Another one-class classification approach was proposed by Gates et al. [11], and it is based on a score function that computes similarities between a file system access events and the history of a user. Designed similarity function is based on computing similarities between files using features such as location in a file system, type of file, etc. In contrast to [11], our approach: 1) does not consider file system hierarchy and any features extracted using it, which positively

affect privacy of anonymized input data, and 2) the prediction is made after longer user's interaction, not with each single file access. Salem and Stolfo [25] used one-class SVM to features extracted from file system as well as other sources such as registry, process creation/destruction, etc. Although the authors achieved promising results, we showed that it is possible to obtain comparable results with more generic approach, which has better privacy preserving properties.

VI. Conclusion

We proposed a one-class approach for detection of masquerade/anomalous user's behaviors in homogeneous user's event logs, which is based on graph partitioning and comparison of graph clusters, while considering that strongly connected nodes have to belong into the same cluster. The evaluation of our approach was performed on user's events represented by file system access logs of datasets called WUIL and TWOS. We achieved an average AUC equal to 0.94 (for WUIL) and 0.85 (for TWOS), while the best configurations obtained by Younden index metric yielded average TPR equal to 95.3% and 91% with FPR of 9.4% and 11.7% for WUIL and TWOS, respectively. This surpasses ad-hoc one-class approaches employed in the previous works [7], [6] that obtained an average TPR equal to 91.5% and an average FPR equal to 11.81% for the best configuration. Moreover, the proposed approach performed similarly to the more informed ad-hoc two-class approach – TreeBagger – employed in [5], which achieved average TPR of 94.4% and average FPR of 6%. Based on the results of the evaluation using the TWOS dataset, we conclude that our approach can be used to detect masquerade attacks performed by human subjects. The empirical time complexity of our approach is linear, which we confirmed by time profiling experiment. Therefore, the properties of our approach are suitable for real deployments either for the forensic purposes or for almost real-time detection (depending on the size of the time window). Finally, we emphasize that our approach ensures high privacy of anonymized input data, as no partial information about a directory structure is exposed after anonymization, in contrast when anonymization has to preserve the file system's hierarchy – this is common for the state-of-the-art masquerader detection approaches working with file system data. In future work, we plan to apply this technique to datasets containing other kinds of homogeneous events (*e.g.,* HTTP requests, SQL queries, etc.) as well as to datasets composed of heterogeneous events (e.g., SIEM logs).

ACKNOWLEDGEMENTS

This research was supported by ST Electronics and National Research Foundation, Prime Minister's Office Singapore, under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec - SUTD Corporate Laboratory).

References

[1] Audit file system. https://technet.microsoft.com/en-us/library/dn319068(v=ws.11).aspx.

[2] Faraz Ahmed and Muhammad Abulaish. A generic statistical approach for spam detection in online social networks. *Computer Communications*, 36(10–11):1120 – 1129, 2013.

[3] Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. Graph-based malware detection using dynamic analysis. *J. Comput. Virol.*, 7(4):247–258, November 2011.

[4] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689 – 694, 1997.

[5] J. B. Camiña, R. Monroy, L. A. Trejo, and M. A. Medina-Pérez. Temporal and spatial locality: An abstraction for masquerade detection. *IEEE Transactions on Information Forensics and Security*, 11(9):2036–2051, Sept 2016.

[6] J. Benito Camiña, Carlos Hernández-Gracidas, Raúl Monroy, and Luis Trejo. The windows-users and -intruder simulations logs dataset (wuil): An experimental framework for masquerade detection mechanisms. *Expert Systems with Applications*, 41(3):919 – 930, 2014. Methods and Applications of Artificial and Computational Intelligence.

[7] J Benito Camiña, Jorge Rodríguez, and Raúl Monroy. Towards a masquerade detection system based on user's tasks. In *International Workshop on Recent Advances in Intrusion Detection*, pages 447–465. Springer, 2014.

[8] Dawn M Cappelli, Andrew P Moore, and Randall F Trzeciak. *The CERT guide to insider threats: how to prevent, detect, and respond to information technology crimes (Theft, Sabotage, Fraud)*. Addison-Wesley, 2012.

[9] Anton J Enright, Stijn Van Dongen, and Christos A Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30(7):1575–1584, 2002.

[10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[11] Christopher Gates, Ninghui Li, Zenglin Xu, Suresh N. Chari, Ian Molloy, and Youngja Park. Detecting Insider Information Theft Using Features from File Access Logs. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *ESORICS - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 LNCS, pages 383–400, Wroclaw, Poland, 2014. Springer International Publishing.

[12] Athul Harilal, Flavio Toffalini, John Castellanos, Juan Guarnizo, Ivan Homoliak, and Martín Ochoa. Twos: A dataset of malicious insider threat behavior based on a gamified competition. 2017.

[13] Wafa Ben Jaballah, , and Nizar Kheir. A grey-box approach for detecting malicious user interactions in web applications. In *Proceedings of the 2016 International Workshop on Managing Insider Security Threats*, pages 1–12. ACM, 2016.

[14] T. Jech. *Set Theory: The Third Millennium Edition, revised and expanded*. Springer Monographs in Mathematics. Springer Berlin Heidelberg, 2006.

[15] Miltiadis Kandias, Vasilis Stavrou, Nick Bozovic, Lilian Mitrou, and Dimitris Gritzalis. Can we trust this user? predicting insider's attitude via youtube usage profiling. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 347–354. IEEE, 2013.

[16] Weixuan Mao, Zhongmin Cai, Xiaohong Guan, and Don Towsley. Centrality metrics of importance in access behaviors and malware detections. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 376–385. ACM, 2014.

[17] Ignacio J Martinez-Moyano, Eliot Rich, Stephen Conrad, David F Andersen, and Thomas R Stewart. A behavioral theory of insider-threat risks: A system dynamics approach. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 18(2):7, 2008.

[18] Michael Mayhew, Michael Atighetchi, Aaron Adler, and Rachel Greenstadt. Use of machine learning in big data analytics for insider threat detection. In *Military Communications Conference, MILCOM 2015-2015 IEEE*, pages 915–922. IEEE, 2015.

[19] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[20] Nam T Nguyen, Peter L Reiher, and Geoffrey H Kuenning. Detecting insider threats by monitoring system call activity. In *IAW*, pages 45–52. Citeseer, 2003.

[21] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, page 45. ACM, 2010.

[22] Younghee Park, Douglas S Reeves, and Mark Stamp. Deriving common

malware behavior through graph clustering. *computers & security*, 39:419–430, 2013.

[23] Jiajie Peng, Kun Bai, Xuequn Shang, Guohua Wang, Hansheng Xue, Shuilin Jin, Liang Cheng, Yadong Wang, and Jin Chen. Predicting disease-related genes using integrated biomedical networks. *BMC Genomics*, 18(1):1043, 2017.

[24] Marcus D. Ruopp, Neil J. Perkins, Brian W. Whitcomb, and Enrique F. Schisterman. Youden index and optimal cut-point estimated from observations affected by a lower limit of detection. *Biometrical Journal*, 50(3):419–430, 2008.

[25] Malek Ben Salem and Salvatore J Stolfo. Modeling user search behavior for masquerade detection. In *International Workshop on Recent Advances in Intrusion Detection*, pages 181–200. Springer, 2011.

[26] Boleslaw K Szymanski and Yongqiang Zhang. Recursive data mining for masquerade detection and author identification. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 424–431. IEEE, 2004.

[27] Stijn Marinus Van Dongen. *Graph clustering by flow simulation*. PhD thesis, 2001.

[28] Verizon. 2016 data breach investigations report. http://www.verizonenterprise.com/verizon-insights-lab/dbir/2016/, 2016.

[29] Gang Wang, Tristan Konolige, Haitao Zheng, Ben Y Zhao, Christo Wilson, and Xiao Wang. You Are How You Click: Clickstream Analysis for Sybil Detection. In *USENIX - 22nd USENIX Security Symposium*, pages 241–256. USENIX, 2013.

[30] Ke Wang and Salvatore J Stolfo. One-class training for masquerade detection. In *Workshop on Data Mining for Computer Security, Melbourne, Florida*, pages 10–19, 2003.

[31] Xiaobin Wang, Yonglin Sun, and Yongjun Wang. An abnormal file access behavior detection approach based on file path diversity. In *2014 International Conference on Information and Communications Technologies (ICT 2014)*, pages 1–5, May 2014.

[32] Tobias Wüchner, Martín Ochoa, and Alexander Pretschner. Robust and effective malware detection through quantitative data flow graph metrics. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 98–118. Springer, 2015.