Ghostbuster: A Fine-grained Approach for Anomaly Detection in File System Accesses

Shagufta Mehnaz
Dept. of Computer Science
Purdue University, West Lafayette, IN, USA
smehnaz@purdue.edu

Elisa Bertino
Dept. of Computer Science
Purdue University, West Lafayette, IN, USA
bertino@purdue.edu

ABSTRACT

Protecting sensitive data against malicious or compromised insiders is a challenging problem. Access control mechanisms are not always able to prevent authorized users from misusing or stealing sensitive data as insiders often have access permissions to the data. Also, security vulnerabilities and phishing attacks make it possible for external malicious parties to compromise identity credentials of users who have access to the data. Therefore, solutions for protection from insider threat require combining access control mechanisms and other security techniques, such as encryption, with techniques for detecting anomalies in data accesses. In this paper, we propose a novel approach to create fine-grained profiles of the users' normal file access behaviors. Our approach is based on the key observation that even if a user's access to a file seems legitimate, only a fine-grained analysis of the access (size of access, timestamp, etc.) can help understanding the original intention of the user. We exploit the users' file access information at block level and develop a feature-extraction method to model the users' normal file access patterns (user profiles). Such profiles are then used in the detection phase for identifying anomalous file system accesses. Finally, through performance evaluations we demonstrate that our approach has an accuracy of 98.64% in detecting anomalies and incurs an overhead of only 2%.

Keywords

Insider attacks; Anomaly detection; File system access

1. INTRODUCTION

Data stored in a file system can be compromised in various ways– by employees with malicious motivations inside an organization or by outsiders. An example of such insider attacks is the breach [1] at Sony Pictures Entertainment where at least one of the six attackers was a former system administrator with extensive technical background and knowledge of Sony's internal systems.

Organizations usually implement a combination of different techniques to protect data from unauthorized access. The main such techniques are user authentication and access control [2, 3]. Authentication is a process by which a system verifies the identity of a user. Access control determines whether an authenticated user has permission to access a particular resource. However, such security solutions are unable to protect data against malicious or compromised insiders [4]. Insiders generally have prior knowledge about the organization's internal procedures, location of sensitive files and weaknesses in systems' security. Hence, insider attacks pose serious threats to organizations with critical or sensitive information and thus have been investigated extensively [4, 5, 6, 7, 8].

As attempts to steal data by malicious or compromised insiders are often characterized by unusual access patterns [9, 10], anomaly detection for data accesses can be a useful technique that well complements other security techniques, such as authentication, access control, and encryption. Anomaly detection (AD) techniques have been proposed for relational databases [11, 12, 13, 14] and networked systems [15, 16, 17]. However, as in many applications data is not managed by a database management system, it is critical that data access anomaly detection techniques be also developed for file systems. Initial approaches to detect anomalies in file system usage leverage different file system features, e.g., file system hierarchy [18], file name, working directory and parent directory [19]. Some techniques [20] use a file system in userspace (FUSE) to capture run-time operations. Approaches have also been developed that focus on file system integrity [21, 22]. A significant common limitation of these approaches is that they are unable to support fine-grained monitoring of accesses to the files. For example, consider the case of an employee that for his daily tasks accesses only 20% of the records in a file and therefore has a read permission on this file. Now, if the employee all of a sudden accesses 100% records of the file, such access is certainly anomalous with respect to his daily task. Even though there could be legitimate reasons for this access, it is critical that such an anomalous access be flagged.

In this paper we propose an effective and practical approach, dubbed as *Ghostbuster*, to detect anomalous accesses to the file system by creating fine-grained user profiles. Our proposed approach comprises of two phases: the *Profile Creation (PC)* phase and the *Anomaly Detection (AD)* phase. In the first phase, we collect detailed information about the file accesses resulting from a user's normal file system activities and create the user profile by using a combination of techniques including data mining. The profiles are later used in the second phase to monitor the file sys-

CODASPY'17, March 22-24, 2017, Scottsdale, AZ, USA

DOI: http://dx.doi.org/10.1145/3029806.3029809

tem usage and to raise alerts upon identifying anomalous activities.

The development of an access AD technique for file systems is challenging. In contrast to database AD techniques where SQL queries provide a structure to learn normal access patterns [11, 12], lack of semantic information about accesses in the case of file systems makes the task of profile construction complex. Moreover, the profile of a user's normal behavior must be accurate. A large number of false positives, i.e., non-anomalous accesses classified as anomalous may slow down the entire system and disrupt the users' normal activities. On the other hand, a large number of false negatives, i.e., undetected anomalous accesses undermine the security protection. This trade-off between performance and security is a major challenge [23] in creating effective profiles. Moreover, an AD technique must add minimal overhead to the data access times. To address the performance issue, we extract a minimal set of interesting features from the users' block level file access information in the *PC* phase. As a next step, we use these features to build the user profiles in a fine-grained manner. Later in the *AD* phase we define and utilize a set of distance functions to measure the difference between a user's profile (i.e., expected behavior) and his file accesses (i.e., observed behavior) at runtime. In the performance evaluation section, we show that our approach is able to identify anomalous access size, anomalous frequency of access, and anomalous access patterns on the whole. We use a Linux OS kernel module named *blktrace* [24] to extract the file accesses at block level.

We list our contributions in the following:

- Our first contribution is the proposed block level access anomaly detection mechanism, which to the best of our knowledge is the first to model users' file access patterns at such a fine granularity.
- The second contribution is a set of efficient algorithms that (1) extract interesting features from the users' block level access information, (2) build effective and practical user profiles, and (3) identify anomalous accesses at runtime.
- As the third contribution, we present a taxonomy of anomalous file accesses and compare the accuracy of our approach with that of other existing approaches based on this taxonomy.
- The fourth contribution is an extensive evaluation of our fine-grained AD mechanism for file systems. We have collected data from a real organizations' file repository. The evaluation demonstrates that our approach achieves an accuracy of 98.64% in detecting anomalies while incurring an overhead of 2%.

The rest of the paper is organized as follows. Section 2 presents an overview of blktrace and other useful notions. Section 3 discusses the features of interest for profile creation and introduces a formal representation of user profiles. Section 4 presents a taxonomy of anomalous file accesses and provides a high-level comparison between our approach and other existing approaches. Section 5 and 6 present the *PC* and *AD* phases, respectively. We evaluate the performance of our approach in Section 7. Section 8 discusses the related work and Section 9 outlines conclusions and future work.

2. PRELIMINARIES

In this section we provide an overview of blktrace and other notions that are used throughout the paper.

2.1 Blktrace Overview

We use the blktrace utility implemented in the Linux kernel to trace block layer I/O operations where each block level access to a file is represented as an *event*. Another utility, *blkparse*, formats the events from the blktrace data. Blktrace has a very low overhead (2%) and can be configured to report a specific set of events, e.g. only read operations. An example of such an event from blktrace output is the following:

8,0	0	427	8.06743	57768	I	R	1729336	+32	[gedit]

427	sequence number of the event
8.06743	timestamp of the event
R	read operation ('W' represents write)
1729336	starting sector number for the read operation
+ 32	32 sectors (or, 4 blocks) are read including sector 1729336 (1 sector = 512 bytes, 1 block = 4096 bytes)

Table 1: Blktrace event components

Table 1 provides a brief idea of the highlighted event components that we use in user profiling.

2.2 Event Sequence and Episodes

Blktrace generates a sequence of events while collecting block level access information from the OS kernel. The events in the sequence are ordered, i.e., the timestamps and sequence numbers that are associated with the events are strictly increasing. Let $E = \{e_1, e_2, \ldots, e_n\}$ be a sequence of n events. An event consists of several attributes. For simplicity, we use only three attributes in this example: *en* (event name); *sr* (sequence number of the event); and *ts* (timestamp of the event) which, for instance, represents the ith event as $e_i = (en_i, sr_i, ts_i)$. Since the events are ordered, $sr_i \leq sr_{i+1}$, and $ts_i < ts_{i+1}$ for $i = 1, 2, \ldots, n-1$. Figure 1 shows an event sequence, E, that consists of $n = 15$ events.

In order to find patterns in a sequence of events we use the notion of episode [25], i.e., set of events that occur together but not necessarily in a consecutive manner. Hence, the episodes that occur frequently represent patterns of an event sequence. If the order among the events of an episode is total, the episode is a *serial* episode; otherwise, it is a *parallel* episode. In the above example, event d occurs after event b twice and thus the two events form a serial episode $\alpha = \{b, d\}$. The occurrences are: {(b, 1, 1), (d, 2, 2)} and {(b, 8, 11), (d, 10, 13)}. A parallel episode can be $\beta = \{c, f\}$ with occurrences: {(c, 4, 5), (f, 6, 7)} and {(f, 13, 19), (c, 14, 21)} where there is no constraint on the relative order of the events.

Episode α is a subepisode of a serial episode γ if the events in episode α form a *subsequence* of events in γ. For example, $\gamma = \{a, b, c, d\}$, and $\alpha = \{a, b, d\}$. Similarly, episode α is a subepisode of a parallel episode γ if the events in episode α form a *subset* of events in γ. For example, $\gamma = \{a, b, c, d\}$, and $\alpha = \{d, b, c\}$. We represent this relation as $\alpha \sqsubset \gamma$.

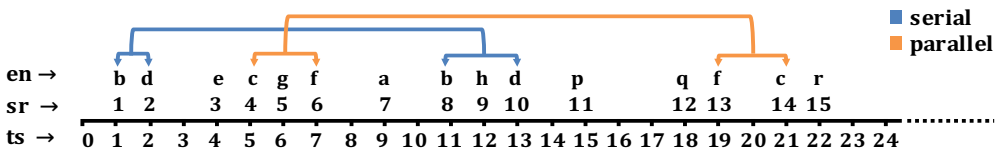


Figure 1: Event sequence E

3. PROFILE COMPONENTS

This section first presents a discussion about the features extracted from blktrace events and then introduces a formal representation of user profiles.

3.1 Features of Interest

- **User**: This feature represents the identity of the user originating a particular blktrace event, i.e., the id of the user reading (writing) from (into) a file.
- **File name**: A series of translations– sector to block, block to inode and inode to file name reveals this feature. This is used as the id of the file.
- **Type of access**: A blktrace event can either be a read ('R'), or a write ('W') operation.
- **Access size**: The number of blocks that are accessed in one blktrace event is represented by this feature.
- **Segment of file**: This feature identifies the relative position of the blocks that are accessed from a file. While profiling at file level only answers whether a user normally reads from a file, our approach leverages this segment information along with the 'access size' feature to precisely identify the portion of the file that a user normally reads. For example, if a file consists of 10 blocks and only the first block is read, we represent this feature as [0,0.1].
- **Timestamp**: The timestamp feature enables us to identify the temporal properties of accesses, e.g., temporal frequency of a file's access, temporal correlation between two files' accesses, etc.
- **Sequence number**: The sequence number feature provides us with the spatial properties of accesses, e.g., spatial frequency of a file's access, spatial correlation between two files' accesses, etc.

3.2 Profile Representation

1) Block level access: This profile component stores the maximum, average, and standard deviation of the size of accesses to each file by the user (in the unit of blocks), and a bit to denote whether the blocks accessed from the file are located in random segments. For a file $\mathbf{f} \in \mathbf{F}$ where $\mathbf{F}$ denotes the set of all files, we compute user $\mathbf{u}$'s block level access information as $B_{\mathbf{f}} = \{sz_{max}, sz_{avg}, sz_{sd}, R = 0/1\}$ and store it in the user $\mathbf{u}$'s profile. Finally, the profile component is denoted as $\mathcal{B} = \langle B_1, B_2, \ldots, B_{|\mathbf{F}|}\rangle$ where $|\mathbf{F}|$ is the cardinality of set $\mathbf{F}$.

2) Frequency of file access: This component accumulates the frequency of accesses to each file by the user. The temporal and spatial access frequencies of a file $\mathbf{f}$ are stored as $fr^t_{\mathbf{f}}$ and $fr^s_{\mathbf{f}}$, respectively. These frequencies are combined as $fr_{\mathbf{f}}$ to represent the profile component $\mathcal{F} = \langle fr_1, fr_2, \ldots, fr_{|\mathbf{F}|}\rangle$.

3) Cluster of accesses: The set of files that are frequently accessed together (represent a unit of task) by the user are considered as an access cluster. Episodes that are frequent in the blktrace event sequence identify these clusters. We store each such frequent episode ε as $C_\varepsilon = \{\varepsilon, cf, S/P\}$ where cf denotes cluster frequency, and S/P determines whether the episode is serial or parallel. Therefore, the access clusters for a user are stored as a vector $\mathcal{C} = \langle C_{\varepsilon 1}, C_{\varepsilon 2}, \ldots, C_{\varepsilon m}\rangle$ where m is the number of frequent episodes identified for the user.

Combining the above components, we represent a user profile as $\mathcal{P} = \langle \mathcal{B}, \mathcal{F}, \mathcal{C}\rangle$.

4. A TAXONOMY OF ANOMALOUS FILE ACCESSES

In this section, we present a taxonomy of anomalous file access cases and give a high-level comparison among our approach and other existing approaches based on the taxonomy. Table 2 summarizes different cases of anomalous file accesses and how different approaches respond to such anomalies. The columns *DC-1*, *DC-2*, and *DC-3* indicate the detection capability of access control mechanisms, a file level profiling approach (e.g., [18]) and our approach, respectively.

Case 1: Anomalous File Access w/o Permission. This case illustrates a masquerade attack where the attacker tries to read from a file to which the legitimate user does not have read permission. A masquerader who broke into the system with a legitimate user's credential but without the knowledge of file permissions may raise this kind of anomalies. This anomaly can be detected by all mechanisms in comparison.

Case 2: Anomalous Access Clusters. This case represents another masquerade attack where the attacker has enough intelligence about the file access permissions and thus does not try to access files to which the legitimate user does not have permissions. However, the accesses by the attacker do not comply with the learned access clusters. An outsider who has somehow gained a legitimate user's credential but does not have knowledge about the specific tasks of this compromised user raises this type of anomaly. While access control mechanisms fail to detect such anomalies, the approach by Gates et. al. [18] as well as our approach can detect such anomalies since both approaches learn access clusters from past history.

Case 3: Anomalous Frequency of Access Clusters. An insider with higher privilege or knowledge about the system's security vulnerabilities can masquerade as another user. This attacker also possesses knowledge about the compromised user's specific tasks, i.e., valid access clusters. However, while looking for sensitive data, the attacker repeats some set of access clusters (because he has background knowledge that the data he is searching for can be collected through this set of tasks). Such behavior by the attacker raises anomalous frequencies for the repeated access clusters and is identified by only our approach among the others in comparison.

Anomaly Cases		Attack Model	DC-1	DC-2	DC-3
Case 1:	Anomalous File Access w/o Permission	Masquerade Attack	✓	✓	✓
Case 2:	Anomalous Access Clusters	Masquerade Attack	×	✓	✓
Case 3:	Anomalous Frequency of Access Clusters	Masquerade & Insider Attacks	×	×	✓
Case 4:	Anomalous Size of File Access	Insider & Data Harvesting Attacks	×	×	✓
Case 5:	Anomalous Segment of File Access	Insider & Data Harvesting Attacks	×	×	✓
Case 6:	Anomalous Frequency of File Access	Insider & Data Harvesting Attacks	×	×	✓

Table 2: Taxonomy of anomalous file accesses and a comparison of detection capability among access control mechanism (DC-1), a file level profiling approach, e.g., [18](DC-2), and our fine-grained profiling approach (DC-3)

Case 4: Anomalous Size of File Accesses. An insider logged into his own account can access more blocks than normal from a file and thus gather larger amount of data from a sensitive file while complying with access clusters and their normal frequency. As shown in Table 2, identifying this data harvesting attack is impossible without a fine-grained approach and our approach is the only one able to detect such anomalous accesses.

Case 5: Anomalous Segment of File Accesses. An insider complying even with normal access size may harvest data by reading different segments of the file. Our fine-grained AD technique is able to detect such random accesses while the other methods fail.

Case 6: Anomalous Frequency of File Accesses. An insider complying even with normal access size may harvest data by reading the file with an elevated frequency. Again, our fine-grained AD technique detects such anomalous case while the other methods fail.

5. PROFILE CREATION PHASE (PC)

Our *PC* phase consists of a set of software modules (Figure 2). The blktrace tool traces block layer I/O operations from the kernel space during the user's normal file access activities. In the following we discuss the modules in detail.

5.1 Feature Extraction *(FE)*

The *FE* module transforms the output from the blktrace tool into an event sequence E. This module represents an event $e \in E$ as *(uid, fname, atype, sz, sg, ts, sr)*, where *uid* is the id of the user, *fname* is the name of the file, *atype* is access type, *sz* is the size of access, *sg* is the segment of the file accessed, *ts* and *sr* are the timestamp and sequence number of the event, respectively. A sequence of n events is thus represented as $E=\{e_1, e_2, \ldots, e_n\}$.

Algorithm 1 Discovering access clusters $\mathcal{C}$

1: **Input**: E, L_{ts}, L_{sr}, $minSp$
2: **Output**: $\mathcal{C}$
3: $m \leftarrow 0$
4: $Cand_1 \leftarrow$ set of distinct events
5: while $Cand_{m+1} \neq \emptyset$
6: $\quad Freq_{m+1} \leftarrow findFreq(E, Cand_{m+1}, L_{ts}, L_{sr}, minSp)$
7: $\quad m \leftarrow m + 1$
8: $\quad Cand_{m+1} \leftarrow genCand(Freq_m)$
9: $\mathcal{C} \leftarrow Freq_1 \cup Freq_2 \cup \ldots \cup Freq_m$
10: return $\mathcal{C}$

5.2 Block Level Profiling *(BLP)*

This module utilizes the *uid, fname, sz,* and *sg* features. In order to profile the accesses at block level, the *BLP* module implements a mapping from each user $\mathbf{u}$ to each file $\mathbf{f}$ and stores all of $\mathbf{u}$'s accesses to $\mathbf{f}$ in the map. Then it computes the values of sz_{max}, sz_{avg}, and sz_{sd} from the map entries. Furthermore, the *BLP* module compares the *sg* features of such access events. If e_1 and e_2 are two events representing accesses to file $\mathbf{f}$, *BLP* identifies whether sg_1 and sg_2 overlap or these segments represent random portions of the file. This analysis is useful, for example, in identifying that a user accesses the latest blocks of a file. If the accessed segments are random, R is set to 1; otherwise set to 0. The value of $B_{\mathbf{f}} = \{sz_{max}, sz_{avg}, sz_{sd}, R = 0/1\}$ is finally stored in the profile of user $\mathbf{u}$.

5.3 Frequency Profiling *(FP)*

The *FP* module utilizes the *uid, fname, ts,* and *sr* features. Two parameters– time interval, TI, and sequence interval, SI, are used in order to compute the temporal frequency ($fr^t_{\mathbf{f}}$) and spatial frequency ($fr^s_{\mathbf{f}}$), respectively. Here we explain how we compute $fr^t_{\mathbf{f}}$ for a file $\mathbf{f}$. Since the access frequency of a file can vary widely over time, we store the frequencies in different granularity levels- hourly (TI = one hour), daily (TI = one day), weekly (TI = one week), and monthly (TI = one month). For example, if a file is accessed a number of times per hour, its frequency is stored in all granularity levels. However, if a file is accessed twice a week its frequency is stored only in weekly and monthly granularity.

5.4 Access Cluster Profiling *(ACP)*

The *ACP* module uses two parameters, L_{ts} and L_{sr}, to denote the maximum time and sequence number differences between two events in order for these events to be considered as related to each other. For example, while determining the occurrence of an event sequence $\{a, b, c\}$, the constraints $ts_c - ts_a < L_{ts}$ and $sr_c - sr_a < L_{sr}$ must be satisfied. These parameters are set to discard uninteresting episodes while searching for the frequent ones. Another parameter, $minSp$, sets the minimum frequency required for an episode to be classified as frequent. The steps for computing profile component $\mathcal{C}$ are presented in Algorithm 1. It is an iterative procedure where in each iteration frequent episodes of progressively larger size are obtained. At the first iteration, all the distinct events are considered as candidates ($Cand_1$). Once the set of frequent episodes of size m, i.e.

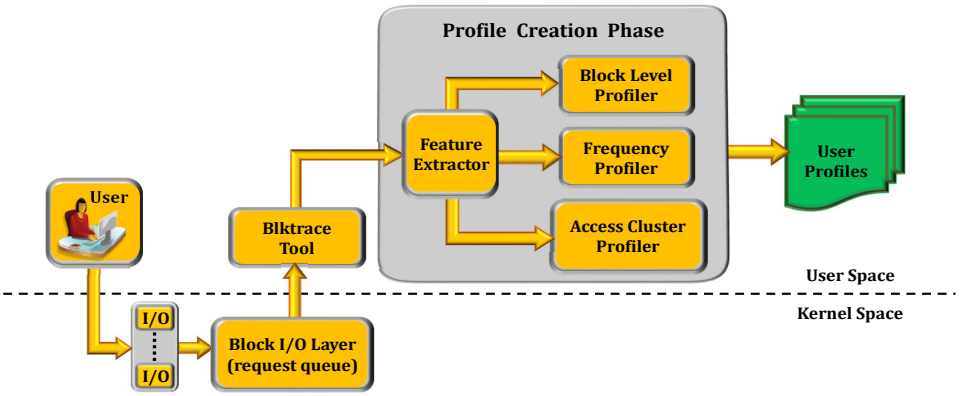


Figure 2: Profile Creation *(PC)* phase architecture

$Freq_m$, is computed from $Cand_m$ (using `findFreq`), the set of candidates for larger frequent episodes, i.e. $Cand_{m+1}$, is obtained from this $Freq_m$ (using `genCand`). The main idea is an Apriori [26] property as given in the following:

LEMMA 1. *Let γ and α be episodes s.t. $\alpha \sqsubset \gamma$. If γ is frequent, α is also frequent.*

According to this lemma, to generate candidate episodes of size $m+1$, we consider only the ones for which all of its subepisodes of size m are frequent. In this way, we discard a number of infrequent episodes and avoid computing their frequency. In what follows, we explain the `findFreq` and `genCand` functions.

Algorithm 2 Finding frequent episodes (`findFreq`)

1: **Input**: E, $Cand_i$, L_{ts}, L_{sr}, $minSp$
2: **Output**: $Freq_i$
3: $Freq_i \leftarrow \emptyset$
4: for each candidate episode $\varepsilon \in Cand_i$
5: $waits(\varepsilon[1]) \leftarrow waits(\varepsilon[1]) \cup (\varepsilon, 1, 0, 0)$ /*initialize $waits$ list*/
6: $\varepsilon.freq \leftarrow 0$ /* set frequency to 0 for all candidates */
7: for each event $e \in E$
8: for each cluster automaton $A_\varepsilon = (\varepsilon, j, ts_f, sr_f) \in waits(e)$
9: if $ts_f = 0$ & $sr_f = 0$ /* automaton at the first state */
10: $ts_f \leftarrow e.ts$ /* update first event timestamp */
11: $sr_f \leftarrow e.sr$ /* update first event sequence number */
12: $waits(e) \leftarrow waits(e) - A_\varepsilon$ /* remove the automaton from $waits(e)$ list */
13: if $j = L$ & $e.ts - ts_f \leq L_{ts}$ & $e.sr - sr_f \leq L_{sr}$
14: $j \leftarrow 1$ /* reached final state of automaton, reset to first state */
15: $\varepsilon.freq \leftarrow \varepsilon.freq + 1$ /* increment frequency */
16: else if $e.ts - ts_f \leq L_{ts}$ & $e.sr - sr_f \leq L_{sr}$
17: $j \leftarrow j + 1$ /* move to the next state */
18: else
19: $j \leftarrow 1$ /* e is outside of L_{ts} or L_{sr} range, reset to first state */
20: $waits(\varepsilon[j]) \leftarrow waits(\varepsilon[j]) \cup A_\varepsilon$ /* add A_ε to a new $waits$ list */
21: for each candidate episode $\varepsilon \in Cand_i$
22: if $\varepsilon.freq \geq minSp$
23: $Freq_i \leftarrow Freq_i \cup \varepsilon$
24: return $Freq_i$

Frequency Computation (`findFreq`): Although there are different definitions of frequency [25, 27], we consider a definition that is based on non-overlapping occurrences as this is the most practical in the context of file access. Two occurrences of an episode α, i.e., α_1 and α_2 are non-overlapping, if no event contained in α_1 appears among events contained in α_2 and vice versa. Let α be a serial episode $\{a, b, c\}$, and let $\alpha_1 = \{(a, sr_{a1}, ts_{a1}), (b, sr_{b1}, ts_{b1}), (c, sr_{c1}, ts_{c1})\}$ and $\alpha_2 = \{(a, sr_{a2}, ts_{a2}), (b, sr_{b2}, ts_{b2}), (c, sr_{c2}, ts_{c2})\}$ be two of its occurrences. We say that α_1 and α_2 are non-overlapping if for all events $x \in \alpha$, $sr_{x1} < sr_{a2}$ and $ts_{x1} < ts_{a2}$, or, for all events $x \in \alpha$, $sr_{x2} < sr_{a1}$ and $ts_{x2} < ts_{a1}$. For the following event sequence–

$$E = \{(b,1,2),(d,2,5),(a,3,6),(c,4,7),(f,5,10),$$
$$(c,6,12),(d,7,13),(c,8,14),(a,9,16),(f,10,20)\}$$

there are more than one overlapping occurrence of episode $\alpha = \{b, d, c\}$, e.g., $\{(b,1,2), (d,2,5), (c,4,7)\}$, and $\{(b,1,2), (d,7,13), (c,8,15)\}$. However, there can be only one non-overlapping occurrence, e.g., $\{(b,1,2), (d,2,5), (c,4,7)\}$ in E and choosing any of the other occurrences violates the given definition of frequency.

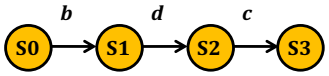


Figure 3: Finite state automata A_ε for $\varepsilon = \{b, d, c\}$

Frequency computation for serial episodes: We use finite state automata to recognize episodes in the event sequence. For example, to recognize the episode $\varepsilon = \{b, d, c\}$, an automaton A_ε transits to states $S1$, $S2$, and $S3$ after observing the events b, d, and c, respectively, as shown in Figure 3. Algorithm 2 shows the steps to count the frequency of serial episodes. It takes the event sequence E, the set of candidate episodes of length i, i.e. $Cand_i$, L_{ts}, L_{sr}, and $minSp$ as the input, and returns $Freq_i \subseteq Cand_i$ as frequent ones. Since we use finite state automata, counting the frequencies of all candidate episodes requires only one pass of the event sequence. Moreover, there is only one automaton for each candidate at a time. We use a $waits()$ list such that all automata waiting for an event e to transit to their next states are placed in the $waits(e)$ list. Therefore, upon scanning an event e from the given event sequence E, the automata waiting for this particular e event proceed to their next states. The automaton for candidate episode ε intending to transit to its jth state is represented as $A_\varepsilon = (\varepsilon, j, ts_f, sr_f)$, where ts_f and sr_f are the timestamp and sequence number of the first event in the automaton, respectively. An automaton

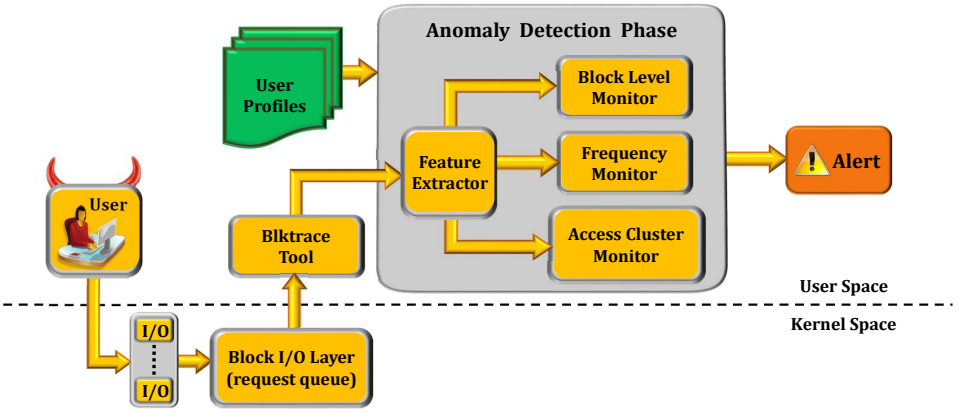


Figure 4: Anomaly Detection *(AD)* phase architecture

$A_\varepsilon = (\varepsilon, j, ts_f, sr_f) \in waits(e)$ means that the automaton for episode ε will transit to its jth state if it observes the event $e = (fname_e, atype_e, sg_e, sz_e, ts_e, sr_e)$ in the events' sequence for which $ts_e - ts_f \leq L_{ts}$ and $sr_e - sr_f \leq L_{sr}$.

Frequency computation for parallel episodes: A similar algorithm is used that we do not present in this paper for space constraint.

Algorithm 3 Candidate generation (`genCand`)

```
1: Input: Freq_m
2: Output: Cand_{m+1}
3: Cand_{m+1} ← ∅
4: S_m ← sort(Freq_m)
5: for each episode s_mi ∈ S_m
6:    for each episode s_mj ∈ S_m where j ≥ i
7:       if S_m.msg[i] = S_m.msg[j]
8:          for k = 1 to m
9:             ζ[k] ← s_mi[k] /* generate new candidate */
10:         ζ[m + 1] ← s_mj[k]
11:         for each α ⊏ ζ where |α| = m
12:            if α ∉ Freq_m /* if any α is not frequent */
13:               goto 6 /* ζ is also not frequent, start over */
14:         Cand_{m+1} ← Cand_{m+1} ∪ ζ
15: return Cand_{m+1}
```

Candidate Generation (`genCand`): Algorithm 3 shows the procedure for generating candidates for parallel episodes. Since the order of events is not important for parallel episodes, events in each episode $\varepsilon \in Freq_m$ are sorted by their $fname$ lexicographically and are then stored in S_m. For example, episode $\varepsilon = \{d, a, f\}$ is sorted as $\{a, d, f\}$. The episodes in S_m are also sorted lexicographically for efficiency purpose. For instance, an episode $\{a, d, e\}$ appears before the episode $\{a, d, f\}$ in S_m. The ith episode in the collection of episodes of length m is denoted as $S_m[i]$. Note that, if episodes $S_m[i]$ and $S_m[j]$ have first l events in common, then all episodes $S_m[k]$, where $i < k < j$, also contain these l events. However, if episodes $S_m[i]$ and $S_m[j]$ share the first $m-1$ events, the only difference between the episodes is the mth event and we denote the group of such episodes with maximal similarity as msg. For each episode $S_m[i]$, we store its msg's first episode index in $S_m.msg[i]$. Hence, the msg array efficiently points to the first episode in the maximal similarity group while generating candidates of size $m + 1$.

A simple modification of this algorithm generates candidates for serial episodes. For space constraint, we omit the algorithm for serial episodes in this paper.

6. ANOMALY DETECTION PHASE (AD)

The *Feature Extractor (FE)* module performs the same set of tasks as in the *PC* phase. Also, the *User Profiles* output from the *PC* phase are considered as input to this *AD* phase. The other three modules (see Figure 4) share a parameter W_{AF} for each user to weigh his anomalous actions. As soon as this parameter exceeds a threshold $minAnom$, the user's behavior is classified as anomalous. Depending on the user's file access activities, the modules raise different anomaly flags and thus update the value of this W_{AF} parameter. **Note that, the user's normal operations are not hindered until W_{AF} exceeds $minAnom$. Only after W_{AF} exceeds $minAnom$, a user is prevented from accessing the requested content.**

6.1 Block Level Monitoring *(BLM)*

The *BLM* module monitors whether the size of an access to a file in the *AD* phase is larger than sz_{max}, or significantly distant from the range $[sz_{avg} + \delta_1 * sz_{sd},\ sz_{avg} - \delta_1 * sz_{sd}]$ where δ_1 is a positive real number. In these cases, this module raises flags of anomaly types AF_1 and AF_2, respectively. The amount of effect an anomaly flag, e.g., AF_1 has on W_{AF} is determined by a distance function. For example, if the sz attribute of an event is larger than sz_{max}, the distance is measured as $dist(AF_1) = (sz - sz_{max})$. Furthermore, if the value of R is 0 in the user profile, and if the user accesses random segments of the file, another anomaly type AF_3 is flagged.

6.2 Frequency Monitoring *(FM)*

This module identifies anomalous frequencies, either temporal or spatial, that exceeds the stored normal frequencies by a threshold δ_2 * $fr_{\mathbf{f}}$. Hence, the *FM* module raises anomaly type AF_4 and stores the frequency anomalous events with detail information for further analysis.

6.3 Access Cluster Monitoring *(ACM)*

The *ACM* module keeps finite state automata for the access clusters identified in Section 5.4. Only the superepisodes are considered for efficiency purpose. For example, if $\{a, b, c, d\}$ is a frequent episode, only one automata is used to monitor this episode and its subepisodes. Assume that the user attempts to read a file **f** in the *AD* phase. If the user does not have permission to read that file, anomaly type AF_5 is raised. Otherwise, we exploit the fact that if this access by the user is legitimate according to the user profile,

there should be at least one automaton to accept the blktrace event resulted by this read operation and to transit to its next state. Otherwise, the *ACM* module flags this access as anomalous and an anomaly type AF_6 is raised. Note that there can be more than one automaton waiting for a single event in which case all of these automata transit to their next states upon the occurrence of that particular event. Anomalous frequencies of access clusters are also monitored and flagged as anomaly type AF_7.

Anomaly Cases	Anomaly Flags	Module (AD phase)
Case 1	AF_5	*ACM*
Case 2	AF_6	*ACM*
Case 3	AF_7	*ACM*
Case 4	AF_1, AF_2	*BLM*
Case 5	AF_3	*BLM*
Case 6	AF_4	*FM*

Table 3: Mapping between the anomaly cases and anomaly flags

Table 3 shows the mapping between the anomaly cases from Section 4 and the anomaly flags raised by the modules in the *AD* phase.

7. PERFORMANCE EVALUATION

In this section, we explain our experiment setup, present the evaluation metrics and the performance of our AD mechanism for different anomaly cases along with a comparison with the other existing approaches.

7.1 Experiment Setup

In order to setup the experiment environment, we collect blktrace data from accesses to a Wikipedia file repository for a duration of two months. A large directory containing 560 files is considered as the target directory. The blktrace data represents accesses to the files by 77 users with unique user IDs. Blktrace events that represent accesses outside the target directory are filtered out before profile creation. We use the data from the first four weeks as training data for profile creation in the *PC* phase whereas the data from the next four weeks is used as test data in the *AD* phase. The histogram in Figure 5(a) reports for different ranges of blktrace events the number of users making accesses within each range whereas Figure 5(b) reports such number for different ranges of distinct file accesses. Each file access generates 6 blktrace events in average. The average number of blktrace events resulting from a user's file accesses is $\sim 15K$ and each user accesses ~ 135 distinct files in average.

The accuracy of the user profiles depends on several parameters used in the *PC* phase, i.e., L_{ts}, L_{sr}, and $minSp$. During the experiment we vary these parameters to learn the users' normal file system activities effectively. According to the background knowledge about the users' tasks with the file system, each task takes 15-20 minutes in average. Figure 5(c) shows how the accuracy of discovered access clusters varies for different L_{ts} values. For instance, if L_{ts} is set to 1 working day (i.e., 8 hours, or 480 minutes), the algorithm interprets many irrelevant file accesses as relevant and thus results in a significant number of false positives in discovering access clusters. Conversely, setting L_{ts} to 1 minute fails to identify correlations in accesses, i.e., incurs high false negative rates and thus results in poor accuracy. Note that, these values of L_{ts} have been experimented only for demonstration purpose and setting such values to L_{ts} are not practical. Hence, in our experiments we set $L_{ts} = 20$ minutes, $L_{sr} = 10$ and use a variable $minSp$ for candidates of different lengths.

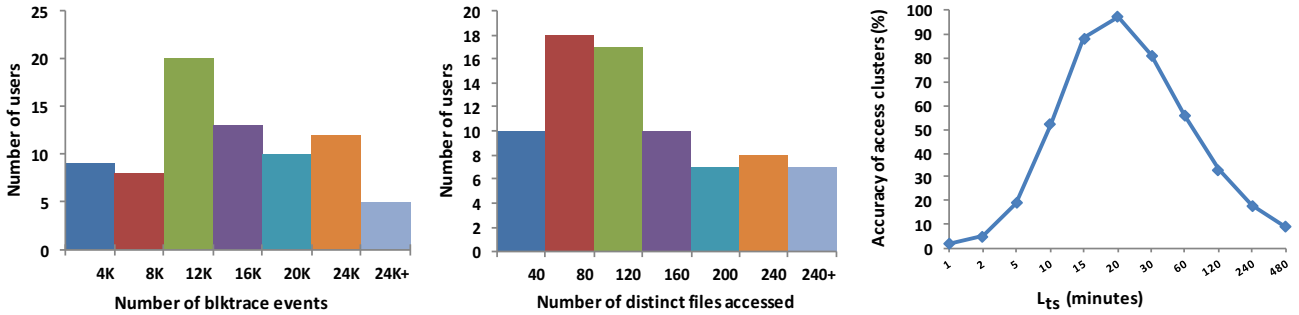
In order to evaluate how our approach performs in the presence of some malicious or compromised insiders, we generate 4 sets of test data, namely TS(I-IV). In TS-I, the data collected in weeks 5-8 remains unchanged. In TS-II, only 3% of TS-I is modified to include anomalous blktrace events which represent a smart attacker. In TS-III, 25% of TS-I is modified to represent a medium attacker scenario. Finally, we randomly access the target directory to generate test set TS-IV that represents a non expert attacker. In Section 7.3, we design three different attack scenarios that incorporate anomaly cases 1-3, 4-5, and 6, respectively, and evaluate the performance of our AD mechanism for these attack scenarios.

Our experiments have been performed on an Intel(R) Core (TM) i7-3770 CPU machine with two cores of speed 3.40 GHz each, using Ubuntu-14.04 operating system and 8GB of memory.

7.2 Evaluation Metrics

Considering *TP* as true positive, *TN* as true negative, *FP* as false positive, and *FN* as false negative, we analyze the performance of our approach using the following metrics:

- False positive rate (FPR) = $\frac{FP}{(FP+TN)}$
- False negative rate (FNR) = $\frac{FN}{(FN+TP)}$



(a) (b) (c)

Figure 5: Different ranges of (a) blktrace events, and (b) distinct file accesses by the users, (c) Accuracy of access clusters

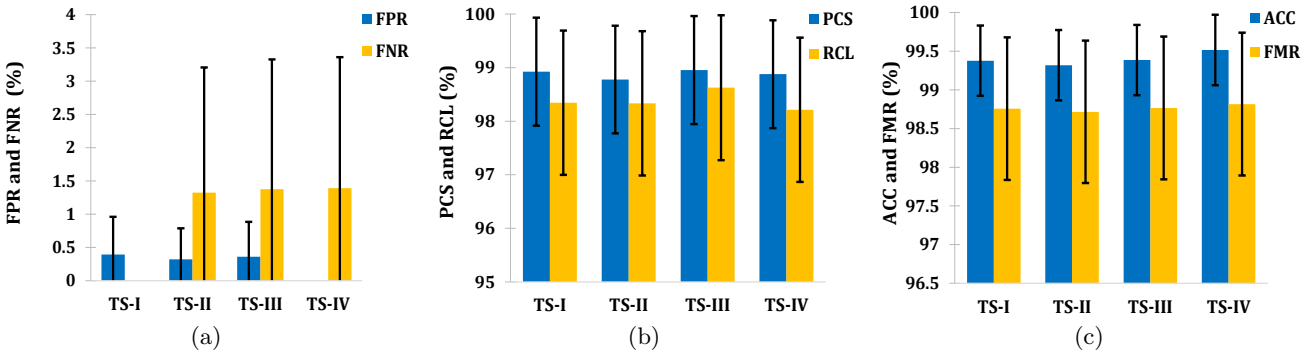


Figure 6: ACM module: mean (a) FPR and FNR, (b) PCS and RCL, (c) ACC and FMR values with confidence interval of standard deviation

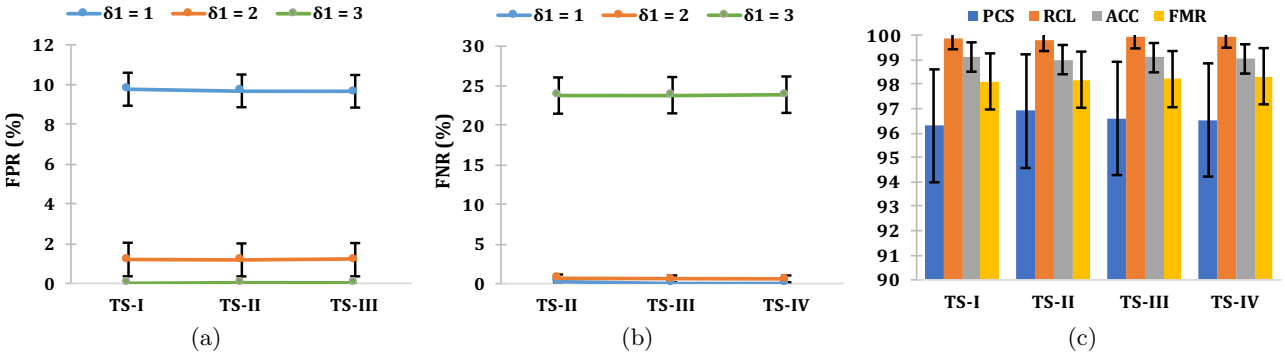


Figure 7: BLM module: mean (a) FPR, and (b) FNR for different δ_1, (c) PCS, RCL, ACC, and FMR for δ_1=2 with confidence interval of standard deviation

- Precision (PCS) = $\frac{TP}{(TP+FP)}$
- Recall (RCL) = $\frac{TP}{(TP+FN)}$
- Accuracy (ACC) = $\frac{(TP+TN)}{(TP+TN+FP+FN)}$
- F-measure (FMR) = $\frac{2TP}{(2TP+FP+FN)}$

7.3 Experiment Results

7.3.1 *Detecting Anomaly Cases 1-3 (Masquerade and Insider Attacks)*

The modifications in TS(II-IV) include blktrace events that represent accesses to files to which the user has no permission, consecutive accesses to files having zero or negligible correlation, and abnormal frequency of some sets of tasks.

Figure 6(a) shows the *FPR* and *FNR* of the *ACM* module for test sets TS(I-III) and TS(II-IV), respectively. Since TS-IV is generated randomly, all the accesses in this set are considered anomalous and therefore, we do not evaluate *FPR* for this test set. Similarly, since TS-I is directly taken from the user's accesses, we do not evaluate *FNR* for this test set. The negligible *FPR* values for all test sets demonstrate the effectiveness of the *ACP* module in discovering the access clusters. In very few cases, the automata for the access clusters reject a normal file access which results in a false positive. The *FNR* is slightly higher than the *FPR* for all test sets. The reason is that when a file is accessed, it requires acceptance by only one automaton to be considered as normal. However, there is a large number of automata in the *ACM* module that may accept the access. If the access is originally anomalous, the automata accepting this access eventually fail to reach their final states and identify the access as anomalous. However, in few cases, due to the complexity of users' interactions with the file system, the *ACM* module considers anomalous accesses as normal and results in false negatives. The average *FPR* and *FNR* values incurred by the *ACM* module are 0.36% and 1.37%, respectively. Note that the *FPR* and *FNR* values do not differ significantly for different test sets. The reason is that our AD approach analyzes each file access individually. Therefore, our AD approach is independent of different percentages of anomalous accesses. Figure 6(b) shows the *PCS* and *RCL* for the *ACM* module with average values of 98.95% and 98.62%, respectively, for all test sets TS(I-IV). Figure 6(c) shows *ACC* and *FMR* with average values of 99.38% and 98.77%, respectively.

7.3.2 *Detecting Anomaly Cases 4-5 (Insider and Data Harvesting Attacks)*

The test sets TS(II-IV) for these anomaly cases include blktrace events that represent abnormal access sizes, and abnormal access segments but comply with file permissions, access clusters and access cluster frequencies.

Figures 7(a) and 7(b) show the *FPR* and *FNR* of the

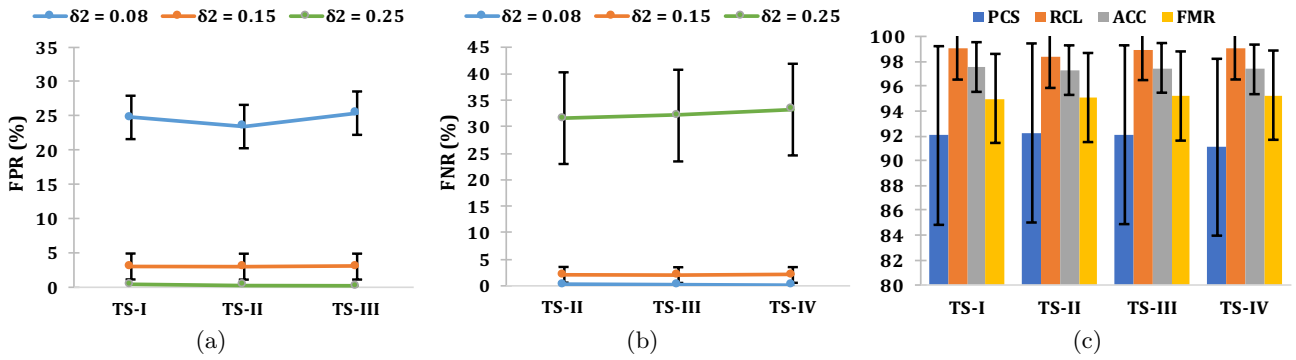


Figure 8: FM module: mean (a) FPR, and (b) FNR for different δ_2, (c) PCS, RCL, ACC, and FMR for $\delta_2 = 0.15$ with confidence interval of standard deviation

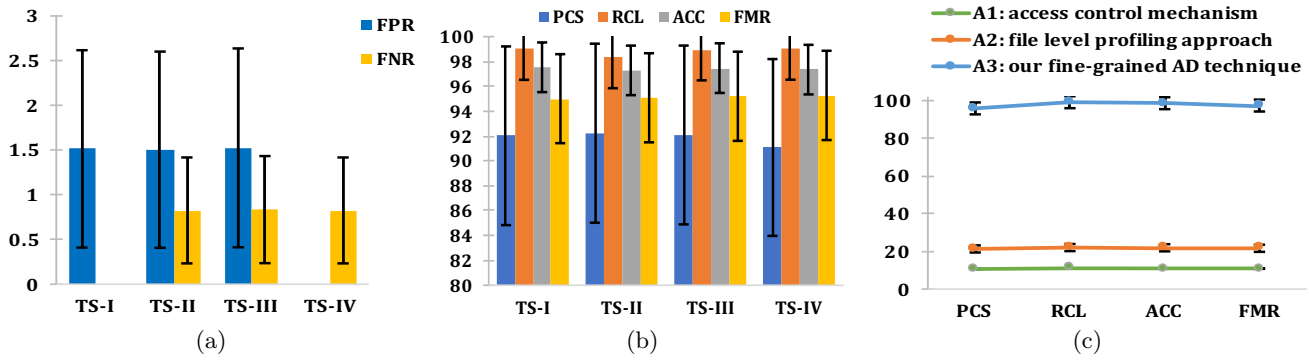


Figure 9: Combined (a) FPR, FNR, (b) PCS, RCL, ACC, FMR of our approach, (c) Comparison with access control mechanism and a file level AD technique

BLM module, respectively. For each test set we vary the δ_1 parameter with values 1, 2 and 3. The experiment shows that setting δ_1 to 1 incurs a high *FPR* of 9.66% in average as this value reduces the acceptable access size range to $[sz_{avg} + sz_{sd},\ sz_{avg} - sz_{sd}]$. On the other hand, a value of δ_1 equal to 3 accepts a wide range of access sizes and results into 23.81% *FNR*. Therefore, we choose δ_1 to be equal to 2 which reduces both the *FPR* and *FNR* to 1.19% and 0.11%, respectively. Since the *BLM* module analyzes each access at block level, it has a slightly higher *FPR*. However, this module is able to detect a wide range of attacks including intelligent insiders who have knowledge about normal access patterns and thus results in a negligible *FNR*. The *FNR* in TS-IV is attributed to the random accesses that comply with the average access sizes in the profile. However, since we consider all the accesses in this test set as anomalous, the accesses that comply with the user profile result in false negatives. Figure 7(c) shows the *PCS*, *RCL*, *ACC*, and *FMR* of the *BLM* module for all test sets TS(I-IV) with average values of 96.5%, 99.89%, 99.08%, and 98.2%, respectively. Note that we set $\delta_1 = 2$ while computing these metrics.

7.3.3 *Detecting Anomaly Case 6 (Insider and Data Harvesting Attacks)*

For this anomaly case, the test sets TS(II-IV) include blktrace events that represent abnormal access frequencies but comply with file permissions, access clusters and their frequencies, access sizes and segments.

The performance of the *FM* module is demonstrated in Figure 8. Figures 8(a) and 8(b) show the *FPR* and *FNR*, respectively. We evaluate this module by selecting different values for δ_2, i.e., 0.08, 0.15, and 0.25. As the value of δ_2 increases, this module increases the range of acceptable frequencies which results in a decrease of the *FPR*. The reason is that some of the files that are accessed with higher frequency than the frequency saved in the profile are not considered as anomalous. Conversely, the value of the *FNR* decreases for lower values of δ_2. However, setting $\delta_2 = 0.15$ results in the *FPR* and *FNR* to be 3.02% and 1.2%, respectively. The results for the *FNR* for TS-IV is due to the fact that some of the randomly accessed files in this test set have frequencies similar to the ones saved in the profile. Again, since we consider all the accesses in this test set to be anomalous, the files with expected frequency attribute to the false negatives. Figure 8(c) shows the *PCS*, *RCL*, *ACC*, and *FMR* of the *FM* module for all test sets TS(I-IV) that have average values of 92.09%, 98.97%, 97.47%, and 95.2%, respectively. Note that we set $\delta_2 = 0.15$ while computing these metrics.

7.4 Comparison with Existing Approaches

For comparison purpose, we combine the three attack scenarios from Section 7.3 where 1/3 of the anomalous accesses include anomaly cases 1-3; 1/3 include anomaly cases 4-5; and the remaining 1/3 include anomaly case 6. In Figure 9(a), we present the combined *FPR* and *FNR* with av-

erage values of 1.53% and 0.83%, respectively. Figure 9(b) represents the combined *PCS*, *RCL*, *ACC*, and *FMR* values with the average values of 95.88%, 99.17%, 98.64%, and 97.39%, respectively. Figure 9(c) reports a comparison among access control mechanism, a file level profiling approach (e.g., [18]) and our fine-grained AD technique for the combination of the attack scenarios above (denoted by $A1$, $A2$, and $A3$, respectively). The figure shows the *PCS*, *RCL*, *ACC*, and *FMR* values of the approaches in comparison. For example, the accuracy of these approaches are 98.64%, 21.92%, and 10.82%, respectively. The reason is that a file level profiling approach can detect only the anomaly cases 1 and 2 while an access control mechanism is able to detect only the anomaly case 1. From the evaluation, it is evident that an access control mechanism or a file level AD mechanism or even a combination of these two is inadequate to detect many anomalous accesses to file systems. Therefore, it is necessary that an approach be deployed to monitor file system activities in a fine-grained manner.

7.5 Overhead

Our AD mechanism operates as a passive component and, therefore, **does not disrupt normal file system activities of the users**. A separate host running our AD mechanism takes care of the computational cost of building user profiles and monitoring file system accesses at runtime. The only additional load that is added to the user's machine is the overhead of running the blktrace tool. However, blktrace has a very low overhead of only 2% [28].

The space requirement for computing and saving the user profiles depends on the duration of tracing blktrace events while collecting the training data, and also on the file system usage by the users. In our experiments, we observe the space requirement for a single user profile to be practical, i.e., less than 1 megabyte in average.

8. RELATED WORK

Most existing AD methodologies have been designed for relational databases [11, 12, 13, 14] and networked systems [15, 16, 17]. Nyalkalkar et al. [29] present a comparison of two network-based anomaly detection methods. Buschkes et al. [30] propose an anomaly detection technique based on profiling mobile users. Baracaldo et al. [31] extends the role-based access control (RBAC) model with a risk assessment process, and the trust the system has on its users. Görnitz et al. [32] devise a learning methodology for anomaly detection that requires less labeled data while achieving higher accuracy. Some research [33, 34] propose to build temporal user profiles in terms of multiple time granularities.

Bowen et al. [35] use believable decoys to detect malicious insiders and propose an automated decoy injection method [36]. In order to detect information theft by insiders Gates et al. [18] use the file system hierarchy for extracting information regarding the relevance of a resource with respect to the users. Due to the dependency on the file system hierarchy this approach cannot handle the case of dynamic file systems where files can be moved from one directory to another. Stolfo et al. [19] extract some file system features, e.g., file name, working directory and parent directory to detect abnormal accesses to the file system and therefore has similar limitations as Gates et al. [18]. On the contrary, our approach does not depend on the file system hierarchy and thus can be used for dynamic file systems.

Senator et al. [37] monitor user activity and combine different structural and semantic information to detect anomalies based on suspected scenarios of malicious insider behavior. Ray et al. [38] propose a framework that uses an attack tree to identify malicious activities from authorized insiders. Claycomb et al. [39] proposes an approach to monitor various systems across an enterprise in order to detect malicious insider activity by using directory virtualization. Huang et al. [20] use an unsupervised approach for detecting application's anomalous run-time operations. This approach collects the file access information of applications to create a baseline profile which is then used to score the file access requests at runtime. Camiña et al. [40] propose a task-based masquerader detector that avoids monitoring every single file system object. Compared to these approaches that work at a higher-level of abstraction, our approach leverages low-level access information (at block level) to detect a broader set of anomalies.

Moreover, some of the previous research work focus on file system integrity [21, 22] rather than confidentiality. I^3FS [22] is an on-access integrity checking file system that compares the checksums of files in real-time using cryptographic checksums to detect unauthorized modifications to files. These approaches are complementary to ours since their focus is on file system integrity.

9. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a technique to create fine-grained profiles of file system users and to use these profiles for detecting anomalous accesses to file systems. We consider that these anomalous accesses are due to the abuse of data by an insider or by an external attacker who can gain access to the files by exploiting the vulnerabilities of the software or by stealing credentials using different techniques, e.g., man-in-the-middle attack, key-logging, phishing, and so on. However, we learn the normal access patterns of the users by utilizing the block level access information from the OS kernel space and detect such malicious accesses with an accuracy of 98.64%.

We notice that our AD system can be easily integrated with anomaly response system [41] that automatically takes actions when an anomaly is detected, based also on contextual parameters. Examples of actions include: blocking access to the file, disconnecting the user, raising an alarm to a system administrator. We believe that effectively and efficiently managing detected anomalies is important for real-world deployment of AD techniques.

In order to prevent a malicious security administrator from abusing the profiles, we consider multiple administrators with separation of duty policy [42]. Notice that our AD mechanism can also be used to monitor accesses by system administrators to the files storing the profiles. Furthermore, an insider or even an external attacker can get access to the profiles in some cases by breaking the security properties which may undermine the AD techniques. However, profiles can be secured by using available security techniques, such as isolating them on secure storage and monitoring accesses to the files storing the user profiles.

Though collusion attacks are not popular among insiders, such attacks may allow each of the insiders to remain under the anomaly threshold but to steal information from the file systems. Identifying these attacks is challenging and thus is left as future work.

Acknowledgment

The work reported in this paper has been partially supported by the Schlumberger Foundation under Faculty For The Future (FFTF) Fellowship and the Purdue PLM Center.

10. REFERENCES

[1] Security breach at sony– here's what you need to know. *http://www.forbes.com/sites/josephsteinberg/2014/12/11/massive-security-breach-at-sony-heres-what-you-need -to-know/*, December 2014.

[2] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *Computer*, 29(2):38–47, February 1996.

[3] J. Park and R. Sandhu. Originator control in usage control. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, POLICY '02, pages 60–, Washington, DC, USA, 2002. IEEE Computer Society.

[4] Elisa Bertino. *Data Protection from Insider Threats*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael, 2012.

[5] Elisa Bertino and Gabriel Ghinita. Towards mechanisms for detection and prevention of data exfiltration by insiders: Keynote talk paper. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 10–19, New York, NY, USA, 2011. ACM.

[6] Cybersecurity watch survey: How bad is the insider threat? Technical report, Carnegie Mellon University, 2012. http://resources.sei.cmu.edu/asset_files/Presentation/2013_017_101_57766.pdf.

[7] Carly Huth and Robin Ruefle. Components and considerations in building an insider threat program. Technical report, Carnegie Mellon University, 2013. http://resources.sei.cmu.edu/asset_files/Webinar/2013_018_101_69083.pdf.

[8] Matthew Collins, Dawn M. Cappelli, Tom Caron, Randall F. Trzeciak, and Andrew P. Moore. Spotlight on: Programmers as malicious insiders (updated and revised). Technical report, Carnegie Mellon University, 2013. http://resources.sei.cmu.edu/asset_files/WhitePaper/2013_019_001_85232.pdf.

[9] David Mundie Andrew P. Moore, Michael Hanley. A pattern for increased monitoring for intellectual property theft by departing insiders. Technical report, Carnegie Mellon University, 2012. http://www.sei.cmu.edu/reports/12tr008.pdf.

[10] Andrew P. Moore, Matthew L. Collins, David A. Mundie, Robin M. Ruefle, and David M. McIntire. Pattern-based design of insider threat programs. Technical report, Carnegie Mellon University, 2014. http://resources.sei.cmu.edu/asset_files/technicalnote/2014_004_001_427430.pdf.

[11] Ashish Kamra, Evimaria Terzi, and Elisa Bertino. Detecting anomalous access patterns in relational databases. *The VLDB Journal*, 17(5):1063–1077, August 2008.

[12] Syed Rafiul Hussain, Asmaa M. Sallam, and Elisa Bertino. Detanom: Detecting anomalous database transactions by insiders. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, pages 25–35, New York, NY, USA, 2015. ACM.

[13] E. Bertino, A Kamra, and James P. Early. Profiling database application to detect sql injection attacks. In *IEEE International Performance, Computing, and Communications Conference, IPCCC 2007*, pages 449–458, April 2007.

[14] Sunu Mathew, Michalis Petropoulos, Hung Q. Ngo, and Shambhu Upadhyaya. A data-centric approach to insider attack detection in database systems. In *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, RAID'10, pages 382–401, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, pages 18 – 28, 2009.

[16] Matthew V. Mahoney and Philip K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 376–385, New York, NY, USA, 2002. ACM.

[17] M. Thottan and Chuanyi Ji. Anomaly detection in ip networks. *Signal Processing, IEEE Transactions on*, 51(8):2191–2204, Aug 2003.

[18] Christopher Gates, Ninghui Li, Zenglin Xu, SureshN. Chari, Ian Molloy, and Youngja Park. Detecting insider information theft using features from file access logs. In *Computer Security - ESORICS 2014*, volume 8713 of *Lecture Notes in Computer Science*, pages 383–400. Springer International Publishing, 2014.

[19] SalvatoreJ. Stolfo, Shlomo Hershkop, LinhH. Bui, Ryan Ferster, and Ke Wang. Anomaly detection in computer security and an application to file system accesses. In Mohand-Said Hacid, NeilV. Murray, ZbigniewW. RaÅŻ, and Shusaku Tsumoto, editors, *Foundations of Intelligent Systems*, volume 3488 of *Lecture Notes in Computer Science*, pages 14–28. Springer Berlin Heidelberg, 2005.

[20] Liang Huang and Kenny Wong. Anomaly detection by monitoring filesystem activities. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 221–222, Washington, DC, USA, 2011. IEEE Computer Society.

[21] *ZFS End-to-End Data Integrity*. https://blogs.oracle.com/bonwick/entry/zfs_end_to_end_data.

[22] Swapnil Patil, Anand Kashyap, Gopalan Sivathanu, and Erez Zadok. Fs: An in-kernel integrity checker and intrusion detection file system. In *Proceedings of the 18th USENIX Conference on System Administration*, LISA '04, pages 67–78, Berkeley, CA, USA, 2004. USENIX Association.

[23] Brendan Juba, Christopher Musco, Fan Long, Stelios Sidiroglou-douskos, and Martin Rinard. Principled

sampling for anomaly detection. In *Proceedings of the Network and Distributed System Security Symposium*, 2015.

[24] *Blktrace*. http://linux.die.net/man/8/blktrace/.

[25] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, January 1997.

[26] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[27] Srivatsan Laxman, P. S. Sastry, and K. P. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 410–419, New York, NY, USA, 2007. ACM.

[28] *Block I/O Layer Tracing using blktrace*. http://smackerelofopinion.blogspot.com/2009/10/block-io-layer-tracing-using-blktrace.html.

[29] Kaustubh Nyalkalkar, Sushant Sinha, Michael Bailey, and Farnam Jahanian. A comparative study of two network-based anomaly detection methods, 2011.

[30] R. Buschkes, D. Kesdogan, and P. Reichl. How to increase security in mobile networks by anomaly detection. In *Computer Security Applications Conference, 1998. Proceedings. 14th Annual*, pages 3–12, Dec 1998.

[31] Nathalie Baracaldo and James Joshi. A trust-and-risk aware rbac framework: Tackling insider threat. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 167–176, New York, NY, USA, 2012. ACM.

[32] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *J. Artif. Int. Res.*, 46(1):235–262, January 2013.

[33] Yingjiu Li, Ningning Wu, Sean Wang, and Sushil Jajodia. Enhancing profiles for anomaly detection using time granularities. *J. Comput. Secur.*, 10(1-2):137–157, July 2002.

[34] Shagufta Mehnaz and Elisa Bertino. Building robust temporal user profiles for anomaly detection in file system accesses. In *Proceedings of the Fourteenth IEEE International Conference on Privacy, Security and Trust (PST)*, 2016.

[35] Brian M. Bowen, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo. *Baiting Inside Attackers Using Decoy Documents*, pages 51–70. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[36] Brian M. Bowen, Vasileios P. Kemerlis, Pratap Prabhu, Angelos D. Keromytis, and Salvatore J. Stolfo. Automating the injection of believable decoys to detect snooping. In *Proceedings of the Third ACM Conference on Wireless Network Security*, WiSec '10, pages 81–86, New York, NY, USA, 2010. ACM.

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

[38] Indrajit Ray and Nayot Poolsapassit. Using attack trees to identify malicious attacks from authorized insiders. In *Proceedings of the 10th European Conference on Research in Computer Security*, ESORICS'05, pages 231–246, Berlin, Heidelberg, 2005. Springer-Verlag.

[39] William Claycomb, Dongwan Shin, and Gail-Joon Ahn. Enhancing directory virtualization to detect insider activity. *Security and Communication Networks*, 5(8):873–886, 2012.

[40] J. Benito Camiña, Jorge Rodríguez, and Raúl Monroy. *Towards a Masquerade Detection System Based on User's Tasks*, pages 447–465. Springer International Publishing, Cham, 2014.

[41] A. Kamra and E. Bertino. Design and implementation of an intrusion response system for relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):875–888, June 2011.

[42] Richard Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations*, CSFW '97, pages 183–, Washington, DC, USA, 1997. IEEE Computer Society.