Detecting Intruders by User File Access Patterns

Shou-Hsuan S. Huang, Zechun Cao(✉), Calvin E. Raines, Mai N. Yang, and Camille Simon

University of Houston, Houston, TX 77204, USA
shuang@cs.uh.com, zcao3@uh.edu

Abstract. Our society is facing a growing threat from data breaches where confidential information is stolen from computer servers. In order to steal data, hackers must first gain entry into the targeted systems. Commercial off-the-shelf intrusion detection systems are unable to defend against the intruders effectively. This research uses cyber behavior analytics to study and report how anomalies compare to normal behavior. In this paper, we present methods based on machine learning algorithms to detect intruders based on the file access patterns within a user file directory. We proposed a set of behavioral features of the user's file access patterns in a file system. We validate the effectiveness of the features by conducting experiments on an existing file system dataset with four classification algorithms. To limit the false alarms, we trained and tested the classifiers by optimizing the performance within the lower range of the false positive rate. The results from our experiments show that our approach was able to detect intruders with a 0.94 F1 score and false positive rate of less than 3%.

Keywords: Intrusion detection · Cybersecurity · Cyber behavior analytics · File accessing pattern · Machine learning

1 Introduction

One purpose of computer security is to limit information access to a certain authorized group of people. Although current computer security methodology keeps out some intruders, it does not deter the most persistent ones. Through backdoors, brute force, or with stolen credentials, intruders can gain access to valuable data and abilities normally reserved for trusted individuals. Once inside, intruders can create a multitude of problems. They could install malware, deface websites or delete documents. These actions would visibly affect the system, requiring damage control or likely an audit. In 2014, the U.S. Office of Personnel Management [20] lost the personal information of millions of people. These individuals were put at risk of blackmail or identity theft. More recently, Equifax disclosed that a massive data breach in 2017 may have impacted 143 million consumers, which is nearly 44% of the US population. In this incident, attackers got their hands on names, SSN, birth date, addresses, some driver license numbers, and about 209,000 credit card numbers, causing the impact of this breach to last for years to come [24]. In both instances, data theft put many people at risk and damaged the reputations of the organizations stolen from. For these reasons, designing an effective

J. K. Liu and X. Huang (Eds.): NSS 2019, LNCS 11928, pp. 320–335, 2019.
https://doi.org/10.1007/978-3-030-36938-5_19

intrusion detection system (IDS) that is capable of quickly raising alarms to security breaches is extremely important.

However, many of the existing IDSs do not provide enough protection as we experience an increasing number of data breaches. Some of the existing IDSs are adopting the misuse-based detecting approach, such as network IDSs Snort and Bro [36]. They rely on pre-defined signatures to identify known attacks. The main problem with this approach is that it fails to properly detect unknown threats because the high number of new vulnerabilities that are discovered every day are not available to the experts for analysis and inclusion in the signature database. Additionally, detecting attacks based on prior knowledge often fails to develop effective signatures as there are usually numerous ways to exploit the same vulnerability [36]. As a result, the misuse-based approach raises too many false alarms by including as many attack signatures as possible, while some attacks can easily evade detection by a slight variance. Additionally, more and more attacks are launched by insiders who are abusing their privileges. The misuse-based IDS cannot detect the attack from an insider, because the intruder can deliberately avoid attacking against vulnerabilities that have been previously exploited.

In this paper, we hypothesize that the deviation of the intruder's cyber behavior from the normal one can be used to detect attempted malicious exploitations. For example, an intruder possessing valid admin credentials is not going to alert existing security systems while logging into the system. Since the objective of the intruders is to scan the data, identify the information of value to them, and quickly exfiltrate the data, the behavior of the intruders will be different from that of the normal users. Differences in intruders' behavior caused by their malicious nature leave traces that can be used for detection. Desiring the ability to detect intruders based on their behavior, we propose an approach that describes the user's behavior through effective features. The objective of this research is to use the differences in the behavioral features in detecting intruders.

As the computer system has been more and more involved in activities in our lives, the file becomes an essential carrier of important and sensitive information, which is often the ultimate target of intruders. Since the file system is the permanent place to store all the files of the host, any malicious execution intended to steal information or damage a host will eventually set its sight upon the file system [31]. Therefore, in this paper, we propose a new intrusion detection approach that uses behavioral features to detect intruders in the file system.

The remainder of this paper is organized as follows. In Sect. 2, we will discuss the existing research on behavior modeling and intrusion detection. Section 3 will expand on presenting and evaluating our behavioral features in modeling user behavior in the file system. In Sect. 4, we will build multiple classifiers with selected features to validate our hypothesis that the user's behavioral difference is sufficient to identify the intruders. Lastly, we will conclude our paper with a summary of our contribution and briefly mention how our approach may be able to further improve intrusion detection in Sect. 5.

2 Related Work

In one of the earliest researches about IDS, Anderson [2] described that audit trails contain valuable information and could be utilized for the purpose of misuse detection

by identifying the anomaly. The proposed misuse-based detecting approach focuses on modeling what is normal, instead of what is anomalous. Since then, we have seen research [12, 22] following this direction with success to a certain extent. But the misuse-based approach is prone to generate many false alarms. Additionally, it has always been a difficult task for researchers to know what to monitor in the system [36]. In an attempt to model normal behavior, Denning [11] presented a host-based IDS that is capable of detecting anomaly by computing statistics (login frequency, password fails, etc.) of system events. Javitz et al. [17] later developed a system called IDES, which issues alerts by comparing the new event's parameters to the thresholds established in the statistical models built on past events, such as a set number of mean values, standard deviations, etc.

In comparison to the host-based IDS, researchers have also studied network-based IDS that detects anomalies by monitoring network packets. Based on existing research [34], many intruders route their SSH connections through a series of computers to make backtracing more difficult. This results in a slower response time, which can be used in stepping-stone intrusion detection. Yang et al. [34] used a clustering-partitioning algorithm to calculate round-trip times of send/echo packet pairs in the network. In doing so they could successfully determine if an intruder was routing their traffic through multiple hosts. Similarly, Kuo et al. [21] proposed an algorithm based on association rule mining to detect stepping-stone SSH attacks. These works focused on detecting intruders by monitoring network activities and shown robustness even under certain types of evasion techniques.

There is relatively little work has been completed in detecting intrusion to the file system. Stolfo et al. [31] developed a File Wrapper Anomaly Detector (FWRAP) which monitors the file system and extracted static features, such as user ID, current directory, file name, etc. from the records. Then Probabilistic Anomaly Detection (PAD) algorithm was applied to detect abnormal processes. However, this approach did not utilize dynamic features and occupied system memory with a large number of files. Other file system intrusion detection methods, such as honeyfiles [35] and decoy documents [4], had proven to be effective under certain assumptions. But there are many limitations, for example, it is expensive to deploy bait files in every user's computer, and the false alarms are overwhelming for the users who are searching intensively in their daily tasks.

We realize that many related studies to our work are attempting to solve masquerader detection problems. Unlike intrusion attack, masquerade attack is a class of attacks in which a user of a system poses as, or assumes the identity of another legitimate user [28]. To detect a masquerader by behavior, existing research tried to model user's cyber behavior based on various information, for example, operating system command [30, 33], mouse usage [26], keyboard usage [19], etc. As Schonlau et al. [30] being the pioneer to design a dataset in an attempt to detect masqueraders, their dataset (SEA) recorded a series of UNIX commands for each user. Although this dataset was widely utilized, it had the limitations of being restricted to UNIX commands and a lack of real "intrusion" behavior. Recently, Camina et al. [7, 8] focused on file system objects and proposed a new feature abstraction model called locality to characterize users and detect masqueraders. Compared to the action-based approach, their locality-based features showed better performance in capturing user behavior for masquerader detection. In

summary, although the research in modeling user's cyber behavior is prolific, there is very limited research has been conducted in the behavior-based intrusion detection method.

3 Modeling User Behavior in File System

Since the file system is often the endpoint of the intrusion chain, it is important to provide protection to it. Therefore, we set the focus of this paper on detecting malicious behavior in the file system. Our working hypothesis is that the intruders behave differently compared to normal users due to their malicious nature. To capture such behavioral differences, we need to define features to model users' behavior in the file system. In this section, we will present in detail how our behavioral features are designed and computed based on the file access logs. We will then evaluate their effectiveness in detecting intruders in a file system dataset extracted from the public repository.

3.1 Behavioral Feature Space in File Access

To profile the user's behavior in the file system, one should continuously monitor the events generated by the system. These events are usually triggered by the OS, applications or users' actions, containing file system objects access history. Modern system logging tools, such as audit [6] in the Windows system, are able to record not only the file system objects that have been accessed, e.g. files and directories, but also the timing information for each record. Without loss of generality, we assume the logs recorded by the system during a time span t can be denoted by a segment of consecutive access entries $s = e_1, e_2, \ldots, e_n$. Each entry e_i, $(1 \leq i \leq n)$ in s describes an access record in the file system with a timestamp. Note that we partition the logs into segments based on the time window of length t because it is not feasible to profile behavior by a single access entry. The behavioral feature space in the file system can be denoted as F, from which another higher dimensional feature space F' can be mapped by a function ϕ

$$\phi(F) \rightarrow F'. \tag{1}$$

A user's behavior in the file system can be viewed as how the file system objects are accessed during a period of time. Thus, we consider temporal and spatial features are essential in deriving other feature spaces by function ϕ. We now propose three basic temporal- and spatial-based features in the behavioral feature space F.

- $\boldsymbol{time}(e_i)$: Timestamp in seconds since the system epoch of an entry e_i logged by the system.
- $\boldsymbol{file}(e_i)$: The file accessed by entry e_i. In the file system log, it is typically represented by the file ID or name of the accessed file.
- $\boldsymbol{path}(e_i)$: Full access path of the file identified by $file(e_i)$. In a tree-structured file system, $path(e_i)$ provides spatial information for the entry e_i.

We expect other behavioral features can be mapped from the feature space F in order to model various behaviors in the file system.

3.2 File Access Behavioral Features

In this section, we introduce two sets of behavioral features that can be mapped from F to detect intruders in the file system. Recall that given a sequence of events $s = e_1, e_2, \ldots, e_n$ during a time window of length t, let $\|s\| = n$ be the size of the file system log segment. The behavioral feature space F can be defined as

$$F = \{time(e_i), file(e_i), path(e_i)\}. \tag{2}$$

Non-frequency-Based Feature Set. These features derived from F give the overall measures of the users within a given time window of length t. Note that the features in this set are intuitively designed aiming to identify the difference in the intruder's behavior from the normal ones.

Entries Per Second (EPS). EPS is the number of log entries within a one-second period. We define

$$EPS = \frac{\|s\|}{time(e_n) - time(e_1)}, \tag{3}$$

where $\|s\|$ denotes the size of the log segment, which is the number of recorded entries of log within the segment. EPS measures how quickly entries are created, which gives an idea of how quickly files are being accessed. A normal user would assumedly spend ample time on a single file, while an intruder would move through many files quickly.

Fraction Unique Path (FUP). This feature shows how many of the paths in the sequence are distinct, measures the diversity of the accessed file. FUP is defined as

$$FUP = \frac{\|\{path(e_i)\}\|}{\|s\|}. \tag{4}$$

A normal user would likely use the same paths repeatedly, but an intruder would rarely use the same path twice.

Fraction Unique Location (FUL). This feature more accurately represents repeated use of the same file. We define FUL as

$$FUL = \frac{\|\{file(e_i)\}\|}{\|s\|}. \tag{5}$$

Different files necessarily have different paths, but two paths may end up in the same file because of the shortcuts. Again, a normal user would be expected to end up at the same file often, but an intruder would access many different files.

Fraction Multiway (FM). This metric combines FUP and FUL. The number of unique files is necessarily smaller than the number of unique paths, thus we have $FM \leq 1$. The smaller FM is, the more often different chosen paths ended at the same location. FM is defined as

$$FM = \frac{\|\{file(e_i)\}\|}{\|\{path(e_i)\}\|}. \tag{6}$$

A normal user should have a favorite way of getting to a known file. An intruder doing an uninformed search would find the same file in every path it can be found.

Fraction Single Path (FSP). FSP measures how many of the paths were only used once in the given sequence of entries. We have FSP as

$$FSP = \frac{\left\| \{e_i : path(e_i) \neq path(e_j) \forall i \neq j\} \right\|}{\|\{path(e_i)\}\|}. \tag{7}$$

Intruders don't need to repeatedly revisit files. In contrast, a normal user would likely use the same path multiple times.

Average Location Time (ALT). ALT averages the difference in time between the first and last entry of each unique location in the given sequence of entries. Let $L = \{file(e_i)\}$, for $k \in L$, let $k_{first} = e_x$: $file(e_x) = k \neq file(e_y) \forall y < x$, and $k_{last} = e_z$: $file(e_z) = k \neq file(e_y) \forall y > z$. Then,

$$ALT = \frac{\sum_L time(k_{last}) - time(k_{first})}{\|L\|}. \tag{8}$$

A normal user might return to a file previously used a few minutes later, while an intruder would avoid looking at the same file over time.

Average Consecutive Time (ACT). ACT measures how long on average a user is working on the same file. The denominator reduces sequence s by removing entries whose path is the same as the previous entry. ACT is defined as

$$ACT = \frac{time(e_n) - time(e_1) + 1}{\|\{e_i : path(e_i) \neq path(e_{i-1})\}\|}. \tag{9}$$

A normal user would be expected to spend longer on a single file than an intruder would.

Average Consecutive Repetitions (ACR). Instead of looking at how much time a user spends before changing files, ACR looks at how many entries are created in succession before moving. Therefore,

$$ACR = \frac{\|s\|}{\|\{e_i : path(e_i) \neq path(e_{i-1})\}\|}. \tag{10}$$

A user repeatedly saving their work would create multiple consecutive entries, while an intruder should do very few actions on a single file.

Average Depth (AD). AD provides the average number of depths in s. We have AD as

$$AD = \frac{\sum_{e_i} (\|path(e_i)\| - 1)}{\|s\|}, \tag{11}$$

where $\|path(e_i)\| - 1$ is the number of directories in an entry's path, which is equivalent to the depth. A normal user would likely use shortcuts or put their files in easy to access locations, while an intruder doing a depth-first search would reach the deepest parts of a file system.

Average Movement Distance (AMD). AMD calculates how many steps the user moves from one entry to its next on average. We define AMD as

$$AMD = \frac{\sum_{e_i} \| path(e_i) \neq path(e_{i-1}) \|}{\|s\|}, \tag{12}$$

where the numerator calculates the distance between two entries. It does so by summing how many different directories or locations there are between two consecutive entries. For example, if event entry 0/1/3/7 is followed by 0/1/2/9/12, the distance would be 5. This signifies moving from 7→3, 3→1, 1→2, 2→9, and 9→12: a total of five steps. A normal user would jump to a very different area when deciding to work on a different task. An intruder, in contrast, would simply check the nearest file.

Frequency-based Feature Set. Comparing to the features proposed above, which are the overall metrics relative to the time window, frequency-based features profile user's behavior based on defined events' frequencies. The event is defined by feature function m, by which intruders and normal users are likely to have different measures. We expect the following 6 features functions to capture the differences in file access patterns.

File Visit Duration (FVD). For each new file accessed that was different than the file that was previously accessed, FVD is measured as the total duration of the user's access time. Thus we have,

$$m(FVD) = \big(time(e_j) - time(e_i) + 1\big), \tag{13}$$

where $i < j$, $e_i = e_j$, $e_{i-1} \neq e_i$ and $e_{j+1} \neq e_j$. The idea in designing this feature is that we expect normal users have a bigger portion of the files that have long access time, while the majority of the files accessed by intruders only being accessed briefly.

Change in Depth Per Second (CDPS). Given a one-second sequence of entries $e_1, e_2, \ldots, e_n$, feature CDPS measures the largest depth difference in the sequence as

$$m(CDPS) = (max(\|path(e_i)\| - 1) - min(\|path(e_i)\| - 1)), \tag{14}$$

where $\|path(e_i)\| - 1$ is the entry's path as we described in feature AD previously. Following a similar intuition with FVD, we expect intruders to be more active in searching for valuable information. Consequently, large changes in depth occur more frequently than in normal users.

Change in Files Per Minute (CFPM). We measure the number of file switches within pairs of consecutive entries during the time window of one minute. Assume we have a one-minute sequence $e_1, e_2, \ldots, e_n$, CFPM is computed as in below

$$m(CFPM) = (\|file(e_i) \neq file(e_{i+1})\|). \tag{15}$$

In this feature, we expect to see fewer file location switches within one minute for normal users, who are performing their daily tasks.

Change in Files Per Second (CFPS). Following the similar idea of CFPM, we shorten the time window to one second in CFPS. For automated file system activities, such as searching, batch operations, etc. it is common to generate numerous location switches within one second. We skip the formula of CFPS here due to its similarity with CFPM.

Number of Times Each File Visited (NTFV). For each unique file in a sequence $e_1, e_2, \ldots, e_n$, NTFV is computed by counting the number of visits for each unique file. Note that we count consecutive entries that are accessing the same file as one visit. To formally define NTFV, we first define the equivalence condition $e_i \equiv e_j$ if $file(e_i) = file(e_j)$ and $file(e_i) \neq file(e_k) \forall 1 \leq i < k < j \leq n$. Based on the equivalence condition, the entries sequence can be partitioned into disjoint sets $G = \{g_1, g_2, \ldots, g_m\}$. For any equivalent classes set $g_z \in G$, we can compute NTFV as in below

$$m(NTFV) = (\|g_z\|). \tag{16}$$

Unlike feature ALT describes how long a user spends time on each file, NTFV focuses on the access frequency of each file. We expect that normal users may visit some files repeatedly, but intruders are less likely to do so.

Changes in Depth in Consecutive Entries (CDCE). For each pair of consecutive entries that are different, we measure the depths difference between entries. Assume we have a sequence of entries $e_1, e_2, \ldots, e_n$, CDCE is computed by the following formula

$$m(CDCE) = (|\|path(e_i)\| - \|path(e_{i+1})\||) \forall e_i \neq e_{i+1}. \tag{17}$$

We simplified the CDCE in Eq. 17 because it computes the difference between depths.

With the 6 feature functions defined above, we have a sequence of measures $m_1, m_2, \ldots, m_n$ for a given block of entries. By applying a predefined threshold value v, threshold-based frequency $f(v)$ can be computed by the formula below

$$f(v) = \frac{\|\{i | m_i \geq v\}\|}{n}, (1 \leq i \leq n). \tag{18}$$

Derived from the behavioral feature space F, we have presented the definitions of 16 behavioral features in order to distinguish the access patterns of the normal users and the intruders. Next, we will evaluate the proposed features with the existing Windows file system logs dataset for their effectiveness in detecting intruders.

3.3 Feature Evaluation

Most of the prior behavior modeling research was intended to solve the masquerader detection problem. Among them, Camina et al. [6] focused on the Windows file system and used the native Windows event logger to record data on each user over five to ten weeks of working days. The normal logs were preprocessed to filter for actions on user objects only, as opposed to system or application objects. The logs were then sanitized

for confidentiality. This dataset was named WUIL [6], stands for Windows-Users and -Intruder simulations Logs, aimed to generate more faithful normal-attack patterns by simulating data exfiltration attacks on the users' machines. By the time we obtained the dataset, the WUIL dataset has been updated to contain a set of normal activity from 76 users and three types of simulated attacker logs for each of the users. The users were volunteers of varying ages, positions, and computer familiarity. Windows system users were chosen because of their abundance. The attacks were restricted to five minutes each, and each of the three attack logs was simulated with a different method.

While each dataset had its weaknesses, the WUIL dataset overcame many pitfalls of its predecessors. It did not use the one-versus-the-others approach found in the SEA [30] dataset. It had the attack and normal data collected from the same machine, which is suitable for intrusion detection analysis. For these reasons, we decide to use the WUIL dataset in this paper to evaluate our intrusion detection method. A few log entries from the WUIL dataset are shown in Table 1. The dataset contains 6 columns of information: entry number, date, clock time, seconds since 12:00 AM on January 1st, 2011 (SS11), depth, and path.

Table 1. Sample entries from the WUIL repository

Entry	Date	Clock Time	SS11	Depth	Path
70	14/02/2012	12:50:59 p.m.	35405459	5	0/1/2/7/15/12
71	14/02/2012	12:50:59 p.m.	35405459	5	0/1/2/7/15/12
72	14/02/2012	12:51:01 p.m.	35405461	4	0/1/2/7/17
73	14/02/2012	12:51:01 p.m.	35405461	4	0/1/2/7/17
74	14/02/2012	12:51:01 p.m.	35405461	4	0/1/2/7/12

The entry number is an indexing element and thus is not very informative for our experiments. SS11 column provides an easy-to-use reckoning of time, such that both date and clock time columns could be derived from SS11. For this reason, we refer to SS11 for the timestamps of the log entries. Depth measures how many levels in the directory of an accessed file's path, e.g., 0/1/3/9/14 has a depth of four. Depth was used in Camina et al.'s original experimentation [6] and was also used by the authors to break user activity into tasks [9]. In this paper, we decide to use the path column for our experiments because depth could be easily derived from the path.

Since the intruders' logs were simulated as five minutes of the malicious activities, it is important to have a comparable size in normal users' data. To achieve this, the normal log entries are split into distinct five-minute blocks, discarding those that aren't suitably active. A block is considered inactive if it contains a gap lasting more than two minutes. To keep as many normal blocks as possible, the first entry after a gap lasts longer than two minutes starts a new block. This could potentially result in a block that lasts merely over three minutes. Therefore, we discarded the blocks that last four minutes or less to ensure they span a comparable length of time with intruders. Reducing the entire log entries down to these active time blocks throws out a sizable amount of normal data, but this is necessary as the intruders' logs included in the dataset are highly active

and time-restricted. If all of the inactive blocks had been kept, the normal data would have been overwhelmed by inactivity, which introduces noise into our experiment. To create a balanced dataset with both classes, we only use the first three five-minute blocks extracted from the normal users. Consequently, the resulting experiment dataset contains 442 data blocks, in which 220 blocks are from normal users, the rest of them are from intruders.

Table 2. Behavioral features ranked by permutation importance

Behavioral Feature	Expected		Importance
	Normal	Intruder	
Change in Files Per Second (CFPS)	Low	High	60.98
Change in Files Per Minute (CFPM)	Low	High	39.45
Average Consecutive Time (ACT)	High	Low	36.84
Entries Per Second (EPS)	Low	High	34.23
Change in Depth Per Second (CDPS)	Low	High	33.96
Average Depth (AD)	Low	High	29.56
Average Movement Distance (AMD)	High	Low	25.66
Fraction Unique Path (FUP)	≈ 0	≈ 1	23.81
Average Location Time (ALT)	High	Low	21.99
Fraction Unique Location (FUL)	≈ 0	≈ 1	21.19
Average Consecutive Repetitions (ACR)	High	Low	21.05
Fraction Single Path (FSP)	≈ 0	≈ 1	17.66
Fraction Multiway (FM)	≈ 1	< 1	16.25
File Visit Duration (FVD)	High	Low	16.08
Number of Times Each File Visited (NTFV)	High	Low	14.58
Change in Depth in Consecutive Entries (CDCE)	Low	High	11.14

We argue that intrusion detection is a binary classification problem between intruders and normal users. To evaluate the effectiveness of the behavioral features in classification models, we compute features' permutation importance by the experiment dataset. The permutation importance is an intuitive, model-agnostic method to estimate the feature importance for classifier and regression models [1]. The procedure to compute permutation importance is straightforward: we take a classification model that is fit to the dataset, and measure its predictive accuracy as baseline performance; for each feature, we then permute its values in the dataset, and compute the decreased performance after permutation; the feature's importance can be computed as the difference with the baseline performance. If a feature with high importance value shows a large performance decrease after permutation, the indicates a strong influence on the classification model, while an irrelevant feature has the value close to 0. In Table 2, we list all 16 behavioral

features with our intuitive expectations for their ranges and ranked by the permutation importance with the Random Forest (RF) model.

For the classification problem of intrusion detection, Table 2 shows that our features are all influential, but not all the features equally contribute to the classification performance. The design similarity in CFPS and CFPM results in a high correlation coefficient of 0.76 between these two features. To our surprise, both of them are the top 2 important features to the classification performance, with CFPS has far more superior importance value than the rest of the group. In Sect. 4, we will select a subset of the features to train the classifiers, and evaluate their performances in detecting intruders in the dataset.

4 Experiments and Results

Our experiments attempt to validate our working hypothesis, that the proposed features can capture the behavior difference to detect intruders in the file system. In this section, we select and use behavioral features with four classification algorithms, namely Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM) and Neural Network (NN) with the dataset extracted from the WUIL repository. We evaluate their performances by a metric that is customized for the intrusion detection problem.

4.1 Performance Measure

One way to analyze the classification performance is by plotting the classifier's Receiver Operating Characteristic (ROC) curves [25]. The ROC curve evaluates the classification performance in a two-dimensional space. It can be understood as the relation of the ratios of the correctly identified positive samples (true positive rate, TPR) and the incorrectly classified negative samples (false positive rate, FPR) at various threshold settings. One of the frequently used metrics extracted from the ROC curve is the value of area under the entire curve, commonly denoted as AUC. With a correctly chosen threshold, a two-class classifier has an AUC value of 1 achieves perfect accuracy, and the classifier that predicts the class at random has an AUC value of 0.5.

In general, acquiring a classifier with large AUC value is desirable, one of the major drawbacks of relying on the AUC metric, however, is that it summarizes the entire curve, including regions that may not be relevant to the security problems (e.g. the regions with high FPR) [23]. Comparing to other classification applications, IDS has much lower tolerance on the false alarm. Dealing with false alarm is not only extremely time- and labor-intensive, but also decreases the chance of a system administrator to capture the real alarms fired by the intruders, which makes the system essentially useless. To remedy this limitation, the Partial Area Under the Curve (pAUC) [18] can be used as a summary index of detecting accuracy over a range of FPR that is of security interest, i.e.

$$pAUC(t_0) = \int_0^{t_0} ROC(t)dt, \tag{19}$$

where $[0, t_0]$ is the range of interest that needs to be specified before using the pAUC metric. In our experiments, we particularly focus on reducing FPR instead of treating TPR and FPR as equally important. In the training process of our experiments, we specify

$t_0 = 0.05$, which means we train and optimize the classifiers by maximizing the pAUC value within the range of 0–5% in FPR.

Although the pAUC value provides an overall evaluation of the classification performance within the range of interest, we need a set of metrics to describe a classifier's performance provided a chosen classification threshold. Accuracy is not an ideal indicator because, in a real organization, the number of normal users is several orders of magnitude greater than the intruders. Hence, all the accuracy values are close to 1 and these results prevent capturing the true effectiveness of a classifier. On the contrary, the F1 score combines precision and recall into a single value, in which the precision indicates how much a given approach is likely to provide a correct result, and recall is used to measure the detection rate. F1 score reaches its best value at 1 with perfect precision and recall, and the worst possible value at 0.

4.2 Feature Selection

Whereas permutation importance is generally considered as an efficient technique that works well in practice, we should not select the features by directly referring to the ranking. In fact, one drawback of the permutation importance is that the importance of correlated features may be overestimated [32]. To reduce the effect of the correlation on the importance measure, e.g. highly correlated features CFPS and CFPM, the Recursive Feature Elimination (RFE) algorithm should be used to make the feature selection [14]. RFE algorithm is inspired by [16] and implemented in this paper with the random forest algorithm. In RFE, features are eliminated iteratively to examine the classifier's performance change. The algorithm is recursive as it updates the ranking based on features' importance after each iteration, then the least important feature is eliminated until no further features remain [14]. Applying the RFE in our experiments eliminated feature CDCE, and achieved the best classification performance with the remaining 15 features. Therefore, we use this 15-feature dataset to train and evaluate the classifiers in the rest of this section.

4.3 Experimental Settings

All our experiments in this paper are conducted with *R*. For all 442 samples in the dataset, 80% of them are for training and 20% are reserved for testing. We employ the 10-fold cross-validation approach, enabled by *R*'s *caret* package, to train the classifiers. In the training process, parameters are chosen by maximizing the classifier's pAUC(0.05) value. Then the performance is evaluated by applying the classifiers to the testing dataset. DT is based on the decision tree algorithm [5, 27, 29]. In this experiment, we used the *rpart* package of *R* to implement the recursive partitioning procedure in growing the decision tree. Using the *rpart* package allows the training function to use Gini Index-based measurement [3] in performing recursive partitioning for modeling. Comparing to DT, an RF classifier contains many decision trees, each of which is constructed by instances with randomly sampled features and produces a response when a set of predictor values are given as input [10, 15]. We construct the RF classifier by using *R*'s *randomforest* package. Existing researches indicate that SVM-based algorithms show noticeable performance gains in anomaly detection system [13, 28,

37] over other machine learning algorithms. In our experiments, we evaluate the SVM classifier by *R*'s package *e1071* with different kernels and report the best performance results from the linear kernel. In the NN classifier, the outcome values come from the input data passing through the multiple successive neural network layers in between. Each layer learns a specific feature of input data and contributes to the final decision. We select a three-layer neural network with the backpropagation algorithm, in which the hyperparameters of the network are optimized by the grid search process to maximize the pAUC(0.05) value. We implemented the NN classifier by the *mxnet* package in *R*, which is a powerful tool to construct and customize the state-of-art deep learning models.

4.4 Experimental Results

Figure 1 plots four ROC curves by applying the classifiers to the testing dataset. We removed the data points that have TPR lower than 70% on the y-axis from the graph, to reduce the redundancy and improve the readability. The x-axis denotes the FPR in percentage, is in the logarithm scale to set the focus on the range of interest of 0–5%.

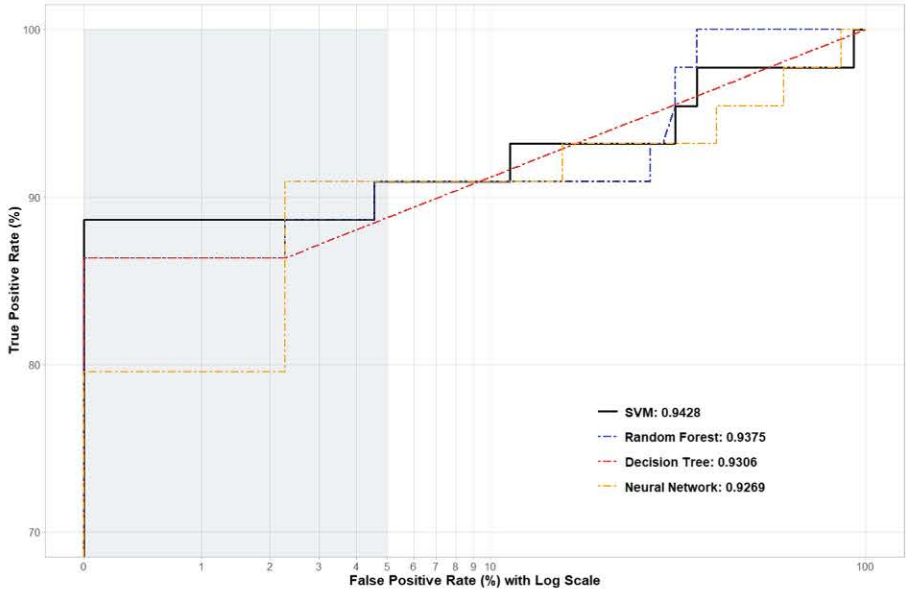

Fig. 1. Comparison of the ROC curves with partial AUC ($\leq$5%).

As we can see in Fig. 1, classifiers perform differently by varying FPR on the x-axis. The RF has a higher true positive rate than other classifiers with higher FPR, however, SVM has the edge if only 2% FPR or less is allowed. In Fig. 1, pAUC(0.05) value is computed by measuring the size of the region under the ROC curve with FPR smaller or equal than 5%. Although the NN classifier has the lowest pAUC(0.05) value of 0.9269, it reaches the highest true positive rate at the 3% FPR mark. In order to compare four

classifiers with the same FPR upper bound, we summarize their best performances respect to 5%, 3% and 1% FPR.

Table 3. Performance of four classifiers with different FPR tolerance values

	DT			RF			SVM			NN		
	5%	3%	1%	5%	3%	1%	5%	3%	1%	5%	3%	1%
Precision	1.000	1.000	1.000	0.952	0.975	1.000	0.952	1.000	1.000	0.976	0.976	1.000
Recall	0.864	0.864	0.864	0.909	0.886	0.864	0.909	0.886	0.886	0.909	0.909	0.795
F1 Score	0.927	0.927	0.927	0.930	0.929	0.927	0.930	0.940	0.940	0.941	0.941	0.886

In Table 3, with every FPR upper bound value evaluated, we list the precision, recall and F1 score for each classifier. We notice that DT retains the same performance for all three upper bound values. This is due to its sparse threshold data points on the ROC curve, and the best performance is achieved at the point where FPR is 0. For the FPR upper bound values of 3% and 5%, the NN classifier performs the best with the highest F1 scores of 0.941. However, SVM surpasses all other classifiers in the F1 score with a 1% FPR upper bound value. These findings from Table 3 are coherent with what we observed in Fig. 1, and validate our hypothesis that the proposed behavioral features are effective in discovering different behaviors to detect intruders in the dataset.

5 Conclusion

In this paper, we hypothesize that intruders have different cyber behavior patterns compared to that of the normal users in a system, which can then be used to detect intrusions. Instead of focusing on local and static features as in existing research, we described a behavior-based model that can be used to detect intruders based on the user file access patterns. The result of our experiments is very encouraging. It supported our hypothesis that normal user behavior in a file system is significantly different from the intruder's, and our proposed behavioral-based model is effective in detecting such behavior deviation with an F1 score of 0.94 and false positive rate upper bound of 3%.

Future work will focus on exploring other possible behavioral features in order to further improve performance. We plan to explore more temporal and spatial features of the time series, including the access order of the file system objects, elapse time before revisiting the same file, etc. Although our method shows a high detection rate with the dataset extracted from the WUIL repository, we are interested in validating this approach with other types of the dataset to further validate this work.

Acknowledgment. We would like to thank Raúl Monroy for creating and sharing the WUIL dataset [6]. This work was supported in part by the National Science Foundation (NSF) under grants NSF-1659755, NSF-1433817, and NSF-1356705.

References

1. Altmann, A., Tolosi, L., Sander, O., Lengauer, T.: Permutation importance: a corrected feature importance measure. Bioinformatics **26**(10), 1340–1347 (2010). ISSN 1460-2059, 1367–4803, p. 395
2. Anderson, J.P.: Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania (1980)
3. Atkinson, E.J., Therneau, T.M.: An Introduction to Recursive Partitioning Using the Rpart Routines. Mayo Foundation, Rochester (2000)
4. Bowen, B.M., Hershkop, S., Keromytis, A.D., Stolfo, S.J.: Baiting inside attackers using decoy documents. In: Chen, Y., Dimitriou, T.D., Zhou, J. (eds.) SecureComm 2009. LNICST, vol. 19, pp. 51–70. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05284-2_4
5. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth & Brooks, Monterey (1984)
6. Camiña, J.B., Hernández-Gracidas, C., Monroy, R., Trejo, L.: The Windows-users and-intruder simulations logs dataset (WUIL): an experimental framework for masquerade detection mechanisms. Expert Syst. Appl. **41**, 919–930 (2014)
7. Camiña, J.B., Monroy, R., Trejo, L.A., Medina-Perez, M.A.: Temporal and spatial locality: an abstraction for masquerade detection. IEEE Trans. Inf. Forensics Secur. **11**(9), 2036–2051 (2016)
8. Camiña, B., Monroy, R., Trejo, L.A., Sánchez, E.: Towards building a masquerade detection method based on user file system navigation. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011. LNCS (LNAI), vol. 7094, pp. 174–186. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25324-9_15
9. Camiña, J.B., Rodríguez, J., Monroy, R.: Towards a masquerade detection system based on user's tasks. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 447–465. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_22
10. Chen, Y.W., Lin, C.J.: Combining SVMs with various feature selection strategies. In: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L.A. (eds.) Feature Extraction. STUDFUZZ, vol. 207, pp. 315–324. Springer, Berlin (2006). https://doi.org/10.1007/978-3-540-35488-8_13
11. Denning, D.E.: An intrusion-detection model. IEEE Trans. Softw. Eng. **13**(SE-2), 222–232 (1987)
12. D'haeseleer, P., Forrest, S., Helman, P.: An immunological approach to change detection: algorithms, analysis, and implications. In: IEEE Symposium on Security and Privacy (1996)
13. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection. In: Barbará, D., Jajodia, S. (eds.) Applications of Data Mining in Computer Security. ADIS, vol. 6, pp. 77–101. Springer, Boston, MA (2002). https://doi.org/10.1007/978-1-4615-0953-0_4
14. Gregorutti, B., Michel, B., Saint-Pierre, P.: Correlation and variable importance in random forests. Stat. Comput. **27**(3), 659–678 (2017)
15. Gupta, B., Rawat, A., Jain, A., Arora, A., Dhami, N.: Analysis of various decision tree algorithms for classification in data mining. Int. J. Comput. Appl. **163**(8), 15–19 (2017)
16. Guyon, I., Weston, J., Barnhill, S., Vapnik, V.: Gene selection for cancer classification using support vector machines. Mach. Learn. **46**, 389–422 (2002)
17. Javitz, H.S., Valdes, A.: The SRI IDES statistical anomaly detector. In: Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, pp. 316–326 (1991)
18. Jiang, Y., Metz, C.E., Nishikawa, R.M.: A receiver operating characteristic partial area index for highly sensitive diagnostic tests. Radiology **201**(3), 745–750 (1996)

19. Killourhy, K., Maxion, R.: Why did my detector do *That*?! In: Jha, S., Sommer, R., Kreibich, C. (eds.) RAID 2010. LNCS, vol. 6307, pp. 256–276. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15512-3_14
20. Koerner, B.I.: Inside the cyberattack that shocked the US government, October 2016. https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/. Accessed 21 Mar 2018
21. Kuo, Y., Huang, S.S.: Detecting stepping-stone connection using association rule mining. In: Proceedings of International Conference on Availability, Reliability, and Security, Fukuoka, pp. 90–97 (2009)
22. Lunt, T.F.: A survey of intrusion detection techniques. Comput. Secur. **12**, 405–418 (1993)
23. Ma, H., Bandos, A.I., Gur, D.: On the use of partial area under the ROC curve for comparison of two diagnostic tests. Biometrical J. **57**, 304–320 (2015)
24. Newman, L.H.: How to protect yourself from that massive Equifax breach, September 2017. https://www.wired.com/story/how-to-protect-yourself-from-that-massive-equifax-breach/. Accessed 21 Mar 2018 (2017)
25. Provost, F., Fawcett, T., Kohavi, R.: The case against accuracy estimation for comparing induction algorithms. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 445–453 (1998)
26. Pusara, M., Brodley, C.E.: User re-authentication via mouse movements. In: Proceedings of ACM Workshop on Visualization and Data Mining Computer Security (VizSEC/DMSEC), pp. 1–8 (2004)
27. Quinlan, J.R.: Introduction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
28. Salem, M.B., Stolfo, S.J.: Modeling user search behavior for masquerade detection. In: Sommer, R., Balzarotti, D., Maier, G. (eds.) RAID 2011. LNCS, vol. 6961, pp. 181–200. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23644-0_10
29. Salzberg, S.L.: C4.5: programs for machine learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. Mach. Learn. **16**(3), 235–240 (1994)
30. Schonlau, M., DuMouchel, W., Ju, W.-H., Karr, A.F., Theus, M., Vardi, Y.: Computer intrusion: detecting masquerades. Statistic. Science **16**(1), 58–74 (2001)
31. Stolfo, S.J., Hershkop, S., Bui, L.H., Ferster, R., Wang, K.: Anomaly detection in computer security and an application to file system accesses. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 14–28. Springer, Heidelberg (2005). https://doi.org/10.1007/11425274_2
32. Strobl, C., Boulesteix, A.L., Kneib, T., Augustin, T., Zeileis, A.: Conditional variable importance for random forests. BMC Bioinf. **9**(1), 307 (2008)
33. Wu, H., Huang, S.S.: User behavior analysis in masquerade detection using principal component analysis. In: Proceedings of 8th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 201–206 (2008)
34. Yang, J., Huang, S.S.: Mining TCP/IP packets to detect stepping-stone intrusion. Comput. Secur. **26**(7), 479–484 (2007)
35. Yuill, J., Zappe, M., Denning, D., Feer, F.: Honeyfiles: deceptive files for intrusion detection. In: Proceedings of the 5th Annual IEEE SMC Information Assurance Workshop (IAW 2004), pp. 116–122 (2004)
36. Zanero, S.: Behavioral intrusion detection. In: Aykanat, C., Dayar, T., Körpeoğlu, İ. (eds.) ISCIS 2004. LNCS, vol. 3280, pp. 657–666. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30182-0_66
37. Zhang, F., Wang, Y., Wang, H.: Gradient correlation: are ensemble classifiers more robust against evasion attacks in practical settings? In: Hacid, H., Cellary, W., Wang, H., Paik, H.-Y., Zhou, R. (eds.) WISE 2018. LNCS, vol. 11233, pp. 96–110. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02922-7_7