




Bagging-RandomMiner: a one-class classifier for file access-based masquerade detection

José Benito Camiña¹ · Miguel Angel Medina-Pérez¹ · Raúl Monroy¹ · Octavio Loyola-González¹  · Luis Angel Pereyra Villanueva² · Luis Carlos González Gurrola²

Received: 29 January 2018 / Revised: 11 May 2018 / Accepted: 15 June 2018 / Published online: 12 July 2018
© Springer-Verlag GmbH Germany, part of Springer Nature 2018

Abstract

Dependence on personal computers has required the development of security mechanisms to protect the information stored in these devices. There have been different approaches to profile user behavior to protect information from a masquerade attack; one such recent approach is based on user file-access patterns. In this paper, we propose a novel classification ensemble for file access-based masquerade detection. We have successfully validated the hypothesis that a one-class classification approach to file access-based masquerade detection outperforms a multi-class one. In particular, our proposed one-class classifier significantly outperforms several state-of-the-art multi-class classifiers. Our results indicate that one-class classification attains better classification results, even when unknown attacks arise. Additionally, we introduce three new repositories of datasets for the identification of the three main types of attacks reported in the literature, where each training dataset contains no object belonging to the type of attack to be identified. These repositories can be used for testing future classifiers, simulating attacks carried out in a real scenario.

Keywords One-class classification · Masquerade detection · information security · User behavior · File access

Abbreviations

AUC	Area under the receiver operating characteristic curve
CD	Critical difference
FP	False positive

FPrate	False positive detection rate
FS	File system
FSN	File-system navigation
GUI	Graphical user interface
HCI	Human–computer interaction
MDS	Masquerade detection system
MTeS	Masquerade testing set
MTrS	Masquerade training set
MRO	Most representative object
PC	Personal computer
PCA	Principal component analysis
ROC	Receiver operating characteristic
TeS	Testing set
TP	True positive
TPrate	True positive detection rate
TrS	Training set
UTeS	User testing set
UTrS	User training set
ZFP	Zero-false positives
1vA	One versus all
5-FCV	Fivefold cross-validation

✉ Miguel Angel Medina-Pérez
migue@itesm.mx

José Benito Camiña
sacaquija@hotmail.com

Raúl Monroy
raulm@itesm.mx

Octavio Loyola-González
octavioloyola@itesm.mx

Luis Angel Pereyra Villanueva
wuicho.pereyra92@gmail.com

Luis Carlos González Gurrola
lgonzalez@uach.mx

¹ School of Science and Engineering, Tecnológico de Monterrey, Carretera al Lago de Guadalupe Km. 3.5, 52926 Atizapán, Estado de México, Mexico

² Facultad de Ingeniería, Universidad Autónoma de Chihuahua, C. Escorza 900, Col. Centro, 31000 Chihuahua, Chih., Mexico

1 Introduction

We live in a technology-dependent society where the use of a personal computer (PC) is common in daily life. A PC stores a large amount of a user's sensitive information, and more often than not has either poor protection against cybercrime or none at all. A *masquerade* is a cyberattack whereby an intruder (or masquerader) impersonates a legitimate user; it is of interest as it is commonly perpetrated by an insider. To mitigate this problem, a *masquerade detection system* (MDS) typically first extracts a profile from a history of user behavior and then uses that profile to raise an alarm whenever a new behavior observation differs significantly from the users profile. Example behavior patterns used for profile construction include command usage [41], keyboard usage [23,24,33,34], mouse usage [36,48], and file-system navigation (FSN) [4–7,38].

In the FSN approach to masquerade detection, a user is profiled based on the files that the user has accessed and how the user has accessed them. The main advantages of FSN are twofold: (i) It is operating system independent; and (ii) it can be used to protect personal profiles in cloud-based file repositories [7]. The FSN approach has been successfully validated using *WUIL* [7], a repository that contains logs of day-to-day user activity, and performs faithful attack simulations on each subject participant's computer. *WUIL* is currently composed of 73 different users, each with logs containing dozens of ordinary daily activities, and considers three distinct types of attacks.

FSN also provides a rich medium for discovering abstract, high-level features for masquerade detection. Accordingly, most research on it has centered on feature selection. For example, Camiña et al. [5] built a detection model out of features regarding file usage (such as new access occurrence and file inter-access time), as well as system-dependent file features (such as file depth and inter-file path distance). In contrast, Camiña et al. [6] proposed a set of features using a key concept of virtual and cache memory to characterize user behavior, namely *locality*. Camiña et al. [6] included a direction feature vector, which captures the position toward which a target user often moves. We call the feature space of [5,6] the *structure* and *locality*, respectively.

Masquerade detection using FSN has been addressed using both multi-class classification [6,7] and one-class classification [32]. In addition to a log of ordinary behaviors (the so-called negative class), the construction of a multi-class classifier requires examples of the positive class, in our case, masquerades. Because it is very difficult to obtain example attacks, it is no surprise that masquerade detection is often structured as a one-class classification problem. However, no experimental comparison has been conducted to determine whether a one-class classifier is as accurate as a multi-class

one, at least in the context of the FSN approach to masquerade detection.

The main contribution of this paper is a novel one-class classifier which is trained using only a sample of allegedly negative examples (ordinary user behavior), which outperforms a number of multi-class classifiers for masquerade detection using FSN. In the design of our classifier, we have drawn inspiration from another one-class classifier, the Bagging-TPMiner algorithm [32], for strengthening classification construction and masquerade identification. Our experimental results using the *WUIL* repository show that our classifier surpasses several multi-class classifiers reported in the literature. These results indicate that practical applications can be built using a one-class classifier without requiring data on real attacks or simulated ones, the latter of which are often argued as not being representative of real-world intruders.

The remainder of this paper is organized as follows. First, in Sect. 2, we overview different types of user behaviors that have been studied for masquerade detection. Then, in Sect. 3, we present an outline of the *WUIL* repository. In Sect. 4, we introduce our proposal for file access-based masquerade detection, which is based on a one-class approach. Next, in Sect. 5, we outline the different classifiers used in our experimentation and discuss our experimental methodology. In Sect. 6, we present the results of our experiments and compare them against those of previous works. Finally, in Sect. 7, we report on the conclusions drawn from this study and provide guidelines for further work.

2 Masquerade detection review

There are several types of user behaviors that have been used for masquerade detection. In this section, we outline some of the most prominent works and the information employed for user profiling.

2.1 UNIX commands

One of the most influential works in masquerade detection was proposed by Schonlau et al. [41]. The authors suggested that users should be profiled regarding their history of computer commands. They developed a repository of datasets, called *SEA*,¹ which became the de facto standard for developing and comparing masquerade detection mechanisms.

SEA includes the activity of 70 users. For each user, it contains a record of 15,000 commands (without options or arguments) executed in a regular UNIX session. The command sequence of each user is divided into 100-command

¹ Available at www.schonlau.net/intrusion.html.

blocks, each of which is called a *session*. Fifty users, randomly chosen, were designated to be *honest*. For each legitimate user, the first 50 sessions are clean, whereas the remaining ones may or may not have been replaced with a session from an illegitimate user. The problem posed by Schonlau et al. [41] is as follows: For each legitimate user, build a mechanism able to distinguish which sessions have been contaminated and which have not. Note that, in passing, SEA does not include faithful masquerades, as it adopts a one versus all (1vA) approach, where other, ordinary users are considered to have attempted to masquerade against an intended user.

SEA has been used with different approaches and classifiers. Two examples are the work of Maxion [30] and the work of Maxion [31], who used naïve Bayes to evaluate a test command sequence in which user u has allegedly participated. The authors first computed the probability that a command sequence has been typed by u and then compared it against a sequence having been generated by not u (evaluated in the same way as the former one, except using legitimate users other than u). In another work, Wang and Stolfo [45] showed that using one-class naïve Bayes based only on the user history of commands is enough to obtain similar masquerader detection performance.

In [25], the authors performed an empirical study with the aim of investigating the effectiveness of a support vector machine (SVM) for detecting masquerade activities using two different UNIX command sets, which were used by Maxion [30] in previous studies. The authors introduced the concept of “*common commands*,” where *names* and *arguments* were included in experiments, and their SVM achieved improvement of masquerader detection from 82.1 to 87.3%.

In [20], the authors argued that models created by SVM are invisible and uninterpretable by security administrators and consequently, new models containing a language more similar to that used by experts in the application domain are needed. The authors designed an experimental study where a rule-based approach for masquerader detection is analyzed using n -grams of the command sequences used by Maxion [30] in previous studies. The authors showed that by using a boosting decision stumps method, classification results were improved from 80.1 to 89.2%.

Recently, bioinformatics genetic algorithms have been used for masquerade detection using SEA, for example, the work of Vidal et al. [44]. In [44], the authors used sequence alignment, which is commonly used to measure the similarity between two sequences of proteins, DNA, or RNA. They measured a similarity score between a user command sequence, commonly called a signature, and the sequence being evaluated. The similarity score is determined through analysis of the number of mutations between sequences. A mutation can be a substitution, an insertion, or an elimination of commands; when fewer mutations occur between

sequences, a greater similarity score is obtained. If the similarity score is high, it is very likely that the sequence was performed by the user; if there are many mutations, the sequence may be a masquerade sequence.

SEA is the most popular repository for masquerader detection; unfortunately, it has contributed more to classification (from a machine learning perspective) than to actual masquerader detection. The main reason is that SEA includes an awkward masquerade scenario, where an intruder has to type in 100 commands because it involves no masquerade attempts, adopting the 1vA validation approach mentioned earlier. This drawback can be witnessed even in recent papers, for example, [22,27], where the key question is how to apply powerful classification techniques rather than how to detect masquerade attacks.

2.2 Mouse usage

Human–computer interaction (HCI) has also been used to approach masquerader detection. A profile is extracted by observing how a user interacts with a computer, usually through a specific I/O device, such as a mouse and keyboard. Because it provides significant control of a graphical user interface (GUI), the mouse has been a popular target in user profiling for masquerader detection.

Pusara and Brodley [36] published a dataset that contains information on mouse movements in a 2D screen, namely the final coordinates of the mouse pointer as well as angle, time, and distance traveled from the last pointer position. The dataset captured information from 18 users who were asked to browse the Internet over a time interval; the model for masquerader detection is hence application dependent. For an MDS, Pusara and Brodley [36] proposed a decision tree classifier, namely C5.0; thus, their MDS is not a one-class classifier and they used a 1vA validation approach.

Garg et al. [15] also provided a dataset based on mouse usage information, the main difference being that the dataset also includes mouse clicks. For an MDS, they suggested using an SVM, but as a binary classifier. Using this approach implies that the model is built using the ordinary behavior of both the user being protected and the remaining community, again using a 1vA validation approach. The dataset proposed by Garg et al. [15] is far from thorough, as it involves only three users.

Shen et al. [42] reported on an experimental study of a range of anomaly-detection algorithms in mouse dynamics. The authors created a dataset containing 17,400 samples from 58 users. They evaluated 17 algorithms designed for detecting mouse dynamics reported in the literature. The comparison measured detection accuracy, usability with respect to sample length, sensitivity to training sample size, and scalability regarding the number of users. The experimental results indicated that the six top-performing detectors

achieved equal error rates ranging from 8.81 to 11.63% with a detection time of 6.1 s.

In conclusion, mouse usage for masquerader detection enables the possibility of contrasting users to one another. However, the masquerade scenarios that have so far been considered are of little interest, as they are constrained to one specific application. Moreover, creating a masquerade dataset is still required because the existing ones include only a few users. Further, the methods do not follow the one-class approach, and consequently, they adopt a 1vA validation approach.

2.3 Keyboard usage

Keyboards remain the most popular device for HCI, and accordingly, they have been a popular subject of study in the context of masquerader detection. For user profiling, keyboard dynamics can be either static (if users are required to type the same key sequence), or free text. Key to the static-text approach is the work of Killourhy and Maxion [23], who studied a number of MDSs reported as state-of-the-art methods. In this case, every MDS under consideration is based on a one-class classifier, and most depend on the notion of distance. Each MDS detects masqueraders based on how the users type their passwords. The authors developed a dataset to conduct a fair comparison of these MDSs so that every MDS is constructed using the same collection of feature vectors. The dataset includes the activities from 51 users, each of which participated in eight sessions. In each session, every user correctly typed a fixed, common password 50 times. Recently, Morales et al. [34] used this dataset, but with the goal of predicting which users are more likely to be masquerade-protected using keystroke dynamics.

In the context of free text, Messerman et al. [33] developed a dataset that contains logs of 55 users working in a webmail application. The dataset mainly involves key presses and time stamps. Again, the MDS proposed by Messerman et al. [33] is not a one-class approach. Furthermore, both [23] and [33] adopted the 1vA validation approach.

Keyboard-based user profiling may not be suitable for a general masquerader detection scenario. On the one hand, MDS construction may span an unacceptably long time interval. This is especially true, for example, in the static-text approach if a change-password policy is mandatory. On the other hand, for successful masquerader detection, this approach currently requires the masquerader to interact with a specific, designated application, which is not realistic for this type of intrusion.

2.4 System-wide MDSs

An alternate view to masquerade detection states that, apart from user behavior, one needs to capture the state of the

underlying working system. For example, (N)IDES [10], one of the earliest attempts at masquerade detection, is an expert system that takes into account CPU usage, the number of attempts to access password-protected directories, and other session information. It also involves user-level features, such as working with commands, and especially working with certain categories of commands, such as compilers or editors.

In a similar approach, Salem and Stolfo [39] structured masquerader detection using a mixture of features. For the user level, they considered features such as browsing, communication, information gathering, among others, and for the system level, registry modification, process creation/destruction, file system (FS) access, DLL usage, etc. In a follow-up paper proposed by Song et al. [43], the authors attempted to identify which of these features best capture user patterns, at least in their dataset, called *RUU*. For that purpose, Song et al. [43] applied Fisher's method of feature selection for multi-class learning (which, as the name suggests, is *not* a one-class approach). They concluded that the most discriminative feature is the number of processes run by each user. While *RUU* includes separate attack logs, these attacks were simulated in an external computer, not in the users computer. This approach could be a serious limitation, because a users pattern may drastically differ from one computer to another, which is attributable to issues such as computer architecture and FS organization.

2.5 FS navigation for masquerader detection

The existing masquerader detection systems, although successful, suffer from two main limitations. First, MDS evaluation has not been thorough, as it does not involve faithfully simulated attacks, adopting a 1vA approach. Second, MDS construction is sometimes limited to the information output by a single application, overlooking the entire picture. We have also seen that a user profile for masquerader detection is usually built out of a record of user actions (in the form of either I/O events or running commands).

Prompted by this situation, Camiña et al. [4] argued that not only is it the action but also the object upon which the action is executed that might be used to distinguish users. They introduced a novel MDS based on the ways in which users navigate the structure of their FS. Furthermore, they developed *WUIL*² Camiña et al. [5], a repository that collects FS navigation information from several users, but more importantly, it collects several faithfully simulated masquerade attempts. They showed that by using a definition of FS object distance, which is not applicable in the context of actions, it is possible to build detection models that outperform those based on commands only.

² <http://homepage.cem.itesm.mx/raulm/wuil-ds/>.

WUIL can be used for different high-level abstractions. An example is *task abstraction* [7], where related files are linked to a user task, relying on an ancestor directory that holds the files. Task abstraction has been recently improved by Rodríguez et al. [38], allowing better user characterization with an increase in efficiency and faster detection time. Another approach, *locality abstraction*, based on the memory-based locality concepts, was developed by Camiña et al. [6] with high-accuracy results. In Sect. 3, an explanation of the WUIL repository information gathering is presented along with further WUIL applications of previous works related to the present work.

In [46], the authors introduced a method for detecting malicious processes that was then extended in Wang et al. [47]. The method uses an anomalous measure, called file path diversity (FPD) that considers the path diversity involved in a process's FS access. Broadly, Wang et al. [46] started by defining an FPD function that takes a file f , and a most popular path, p ; if f is a descendant of p , then FPD equals 0; otherwise, it equals $9/3^{\text{lca}(f,p)}$, where $\text{lca}(f,p)$ stands for the path of the *least common ancestor* of both f and p , and length has its standard interpretation, returning the length (or depth) of a given path. $\text{fpd}(f,P)$ is the extension of $\text{fpd}(f,p)$ over a set of most popular paths, P , in the expected manner, and yields the minimum $\text{fpd}(f,p)$, for some $p \in P$. Now, given both a file access window w of the form f_0, \dots, f_n , and P , a set of most popular paths, Wang et al. [46] mark w as abnormal if:

$$\frac{1}{s+1} \left(\text{fpd}^2(f_0, P) + \text{fpd}^2(f_1, P) + \dots + \text{fpd}^2(f_s, P) \right) > u \quad (1)$$

where u is a predefined threshold. Finally, a process is considered malicious if the FPD determines that at least k windows are abnormal.

In [16], the authors attempted to detect insider malicious behaviors, such as stealing files. The authors claim that it should be possible to exploit information between the files that are accessed by user j in the current period A_j^T , against files accessed during the users history, A_j^H . Gates et al. [16] worked under a hypothesis based on two conditions: (i) If files that one user currently accesses have been accessed in the recent past by the same user, then this is likely to be a legitimate action; and (ii) if files are similar to files previously accessed, then this is less likely to be malicious theft. To measure similarity, the authors resort to two observations: First, files under the same directory may be considered as more similar than files that are far apart in the FS; and second, files accessed by essentially the same set of users may be viewed as more similar than files accessed by a disjoint set of users. The authors successfully applied the techniques to profile identification and anomaly detection; however, there is not a

single combination of techniques that dominates among the best results.

In [40], the authors performed a masquerade detection experiment that combines different behaviors. They collected information on GUI coordinates, mouse data, keyboard data, command line data, and file accesses under the Linux OS. For file accesses, they collected the number of successful and unsuccessful attempts to access specific files or folders including the password file, log folder, and bin folder, among others. Data gathering was performed on a shared computer used by 16 different users, where there was an absence of masqueraders and the 1vA approach was used.

3 Repository based on structure and locality

WUIL [5] is a repository of datasets developed by its authors, who were motivated by the necessity for a masquerade file access test dataset. WUIL is comprised of different users' file access logs and a set of three types of simulated attacks performed on each of the users' computers. In this section, we present an overview of the most important aspects of WUIL.

In its first version, WUIL included 20 users [5], but over the time it has been enriched, ending with 73 users [7]. For this work, we have added one more user, for a total of 74 users. The WUIL logs were gathered using the Windows tool named *audit*. Each entry in a log holds different types of information, the most important being time, date, and the path to the file being accessed. For each user, file usage was collected for the amount of time that the user allowed; some users provided only one-day logs, but others provided more than 60-day logs.

WUIL also includes three types of simulated attacks. The attacks are called basic, intermediate, and advanced. In the basic attack, the masquerader is not prepared to perform the attack and he can only see the files. In the intermediate attack, the masquerader is prepared with a USB flash drive, and using the Windows search utility, looks for a specific set of files and manually copies them to the USB flash drive. Finally, in the advanced attack, the masquerader equips the USB flash drive with a script that automatically searches for files and copies them to the USB flash drive.

WUIL has been shown to be suitable for use in different types of experiments. For example, WUIL has been used with different types of windows (event windows [7,38] and time windows [5,6]). In addition, different types of features have been extracted from WUIL logs, including navigation statistics [5], user tasks [7,38], and locality [6].

3.1 Feature extraction on WUIL

In Camiña et al. [5], a set of 25 features were extracted from WUIL, and we will refer to them as *structure features*.

WUIL logs were divided into 30-s time windows, and structure features were based mainly on statistics of: number of file accesses, time between each file access, the path distance between two adjacent accesses, navigation structure, and depth. We refer the reader to Camiña et al. [5] for the complete details of these features.

Classification results obtained using structure features were appealing but not conclusive. Therefore, in Camiña et al. [6], the authors introduced a new set of 16 features based on an abstraction of the cache memory term of locality; henceforth, we will refer to them as *locality features*. Here, WUIL logs were divided into 30-s windows.

Locality features can be classified into three groups: *spatial* (four features), *temporal* (eight features), and *direction* (four features). The spatial locality features are based on the distance between two consecutive file accesses assuming that whenever the user has accessed a file, then it is highly likely he/she will access files close to it in the near future. The temporal locality features are based on the time elapsed since the last time a file was accessed, assuming that whenever a file has been accessed, it is highly likely to be accessed again in the near future. The direction features are based on where the user is moving to on the subsystem tree within the window, assuming that the user should exhibit common direction patterns. We refer the reader to Camiña et al. [6] for the complete details of these features.

Camiña et al. [6] performed an experiment comparing the performance results of structure features against locality features using TreeBagging, a multi-class classifier. The results showed that masquerade detection based on locality features outperforms that based on structure features. As the authors in [6] aimed to show the strength of using locality features, they tested various feature selections, but they considered only TreeBagger in their experiments. This gap in the authors' research methodology has motivated us to examine our hypothesis that using a one-class classification approach obtains similar or better classification results using locality features to results obtained using other classification approaches for file access-based masquerade detection.

4 Proposal for file access-based masquerade detection

Bagging-TPMiner [32] aims to identify typical objects to represent a genuine users behavior in the context of masquerade detection. To this end, the algorithm calculates a distance between all pairs of a subset of objects, which in the worst-case scenario is estimated to require up to $O(n^2)$ comparisons. As a consequence, for large datasets, training the algorithm is of very limited practical use. Thus, we have extended Bagging-TPMiner so that it both requires significantly less time for building a model and yields classification

results without a significant statistical difference with respect to Bagging-TPMiner.

Instance-based classification has been proven to obtain competitive classification scores in a wide range of problems [1]. This idea has motivated us to develop an extended version of Bagging-TPMiner that focuses on identifying objects that could represent prototypes. A way to select these objects is to randomly sample a given user dataset. The rationale behind this idea is to take into account zones that represent common user behavior and place less importance on isolated objects, i.e., rare user behavior. Then, the obtained sample can resemble the set of typical objects (as computed in Bagging-TPMiner) and no inter-prototype distance calculation is required. Note that training and classification in our extended version are similar to those used in the original version of the Bagging-TPMiner algorithm, except that our training set (TrS) has a larger number of objects than the original version.

4.1 Bagging-RandomMiner

Just as Bagging-TPMiner, Bagging-RandomMiner is an ensemble consisting of multiple instances of the same classifier, RandomMiner in our case. RandomMiner works by randomly selecting a percentage of the objects of a given training dataset. The output of this step consists of a set of objects that we call *the most representative objects* (MROs). Each member in the MRO set is similar in context to a so-called *typical object* in TPMiner, except that typical objects are found using an exhaustive search. In a practical sense, RandomMiner differs from the original TPMiner only during the training of each individual classifier.

As a visual aid of Bagging-RandomMiner, consider Fig. 1 where a flow diagram of the training phase is presented. This phase receives as input: the training dataset (T), the number of classifiers to use in the ensemble (N), and the fraction of objects (F) from the training dataset to bootstrap to build a classifier. It generates as output an ensemble P of N classifiers.

In detail, to construct a particular classifier i , we first bootstrap a fraction F of objects from T , the TrS, and we place those objects in T' . Then, we sample $RS_{\text{percent}}\%$ of the objects in T' and name them MRO. The next step is to compute the covariance matrix over the objects in the bootstrap T' and calculate the average object pair distance, which will serve as a threshold to dictate, in the classification phase, if a new object should be deemed as a normal user behavior or not. The product of each iteration is a classifier that is specified by a triplet: the MRO set, the average object pair distance in the MRO set, and the covariance matrix, which in turn are added to the ensemble P . The ensemble P is inputted in the classification stage, which, as in TPMiner, evaluates

whether a new object belongs to the negative class or should be annotated as a masquerade.

4.1.1 Classification phase

The classification phase receives as input the ensemble P of classifiers that was generated in the training phase and an object (O) to classify. Consider Fig. 2, where a flow diagram of the classification phase is presented. First, we initialize S , which is a queue that will be used to store the vote that each classifier assigns to the object O . For each classifier, the minimum distance between an object in the MRO set and the object O is computed and saved in $dmin$. This distance is used by the classifier to calculate a similarity value which will fall within the interval $[0,1]$. Note that a high similarity value (> 0.6) indicates that the object O is likely to belong to the cluster represented by MRO, corresponding to regular user behavior. Once all the classifiers have registered a vote, an average of all votes in S is calculated, which represents the similarity value calculated by the ensemble. Based on the idea that consecutive objects correspond to the same user behavior, another queue, Q , is also constructed and is used to calculate the average final score, which is the output of the classification phase.

4.2 Classification region for Bagging-RandomMiner

To evaluate the performance of Bagging-RandomMiner, we now discuss the merit of its classification region, especially when compared against that of Bagging-TPMiner. To illustrate our point, we use the (training) dataset of *User 1* from WUIL [5] as a running example. Figure 3 portrays a 2D representation of this dataset. To compute this representation, we apply principal component analysis (PCA) [11] over the dataset and then select the first two principal components from it. Figures 4 and 5, respectively, portray the MRO and the classification region for Bagging-RandomMiner; Figs. 6 and 7 do so for Bagging-TPMiner.

From Figs. 4, 5, 6 and 7, our proposal (Bagging-RandomMiner) uses a smaller classification region than Bagging-TPMiner, because it identifies areas with a high density of data points. However, this classification region is included in that of Bagging-TPMiner, thus explaining why the former classifier attains results as good as those obtained by the latter. A further analysis of this situation leads us to hypothesize that although Bagging-TPMiner rightly considers typical objects whose influence area spans all possible genuine user data points, the real pattern for a given user is better captured by the more common behaviors than by rarely executed actions (isolated data points). The strength of Bagging-TPMiner for identifying typical objects that correspond to uncommon, although genuine, user behavior

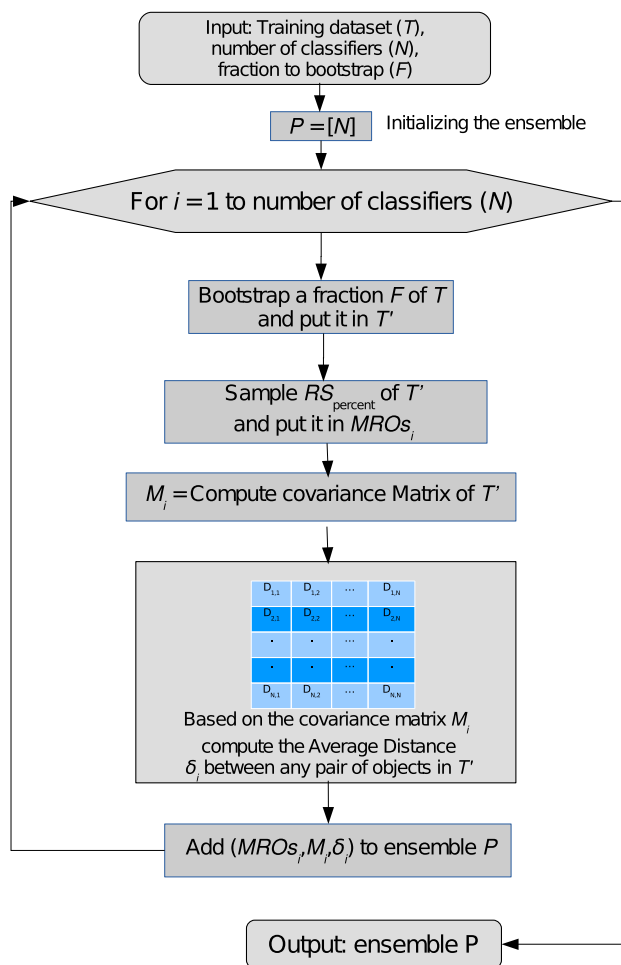


Fig. 1 Flow diagram of the training phase of Bagging-RandomMiner

appears to hinder its ability to fully discriminate some data points of the minority class (masquerade attempts).

4.3 Parameter tuning for Bagging-RandomMiner

As the reader may now suspect, the performance of Bagging-RandomMiner essentially depends on the percentage of the input dataset to sample. This parameter value actually yields a trade-off between classification performance and the time required for classifier construction. To identify what value(s) one should use, we compute classification measures using the range $RS_{\text{percent}} \in \{5\%, 10\%, 15\%, 20\%\}$; this is because, beyond 20%, our experiments show no classification improvement. We have followed the validation protocol introduced by Medina-Pérez et al. [32] for parameter tuning; however, instead of using the WUIL repository [5], we used the repository of datasets and the same data preprocessing technique introduced in Camiña et al. [6].

Figure 8 portrays a summary of all performance values for Bagging-RandomMiner, as measured by the area under the receiver operating characteristic (ROC) curve (AUC) [12].

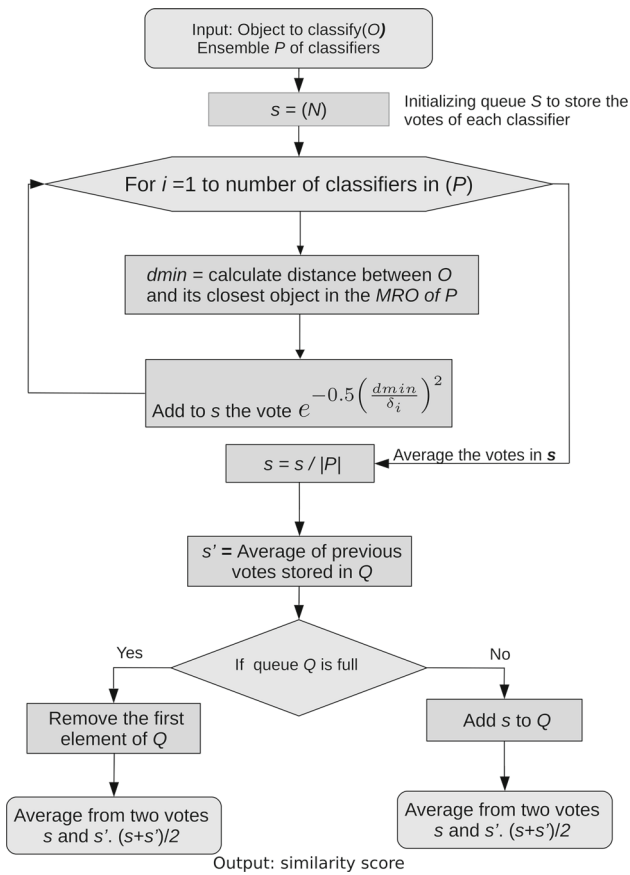


Fig. 2 Flow diagram of the testing phase of Bagging-RandomMiner

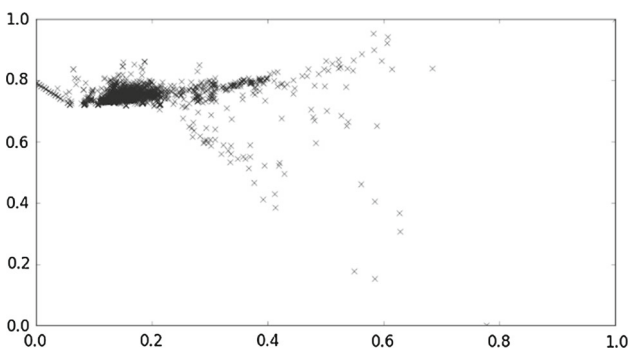


Fig. 3 Graphical representation of running example consisting of the two main principal components as output by applying PCA to the dataset of User 1 in WUIL

The figure presents the results using a *critical difference (CD) diagram* [9], because it succinctly presents the rank of an algorithm with respect to the performance indicator (the AUC in this case). The CD diagram also shows both the magnitude and the significance of any differences among the algorithms' performance [9]. Note that, in particular, the best algorithm appears rightmost, and statistically similar algorithms are joined by a thick horizontal line. More specifically, a CD diagram depicts the results of the Friedman test [9] and

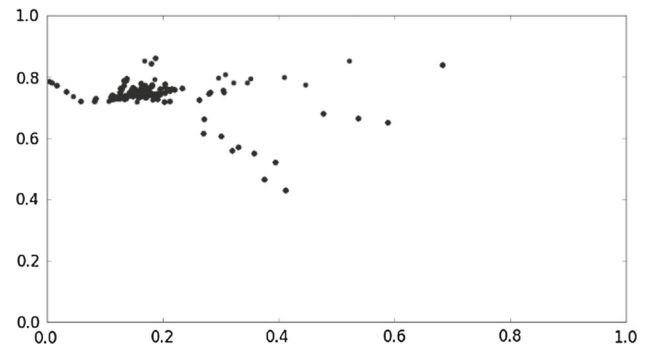


Fig. 4 Most representative objects (MROs) identified by running Bagging-RandomMiner over the running example

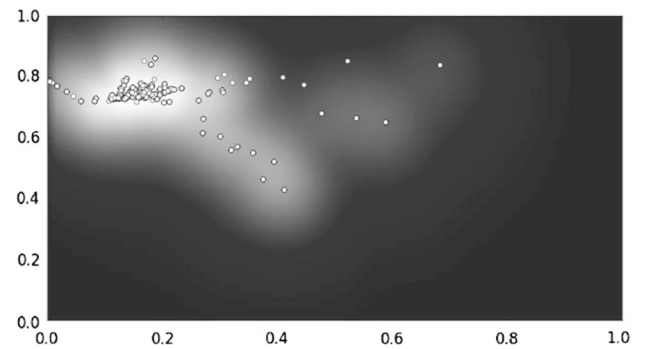


Fig. 5 Classification region of Bagging-RandomMiner associated with the running example

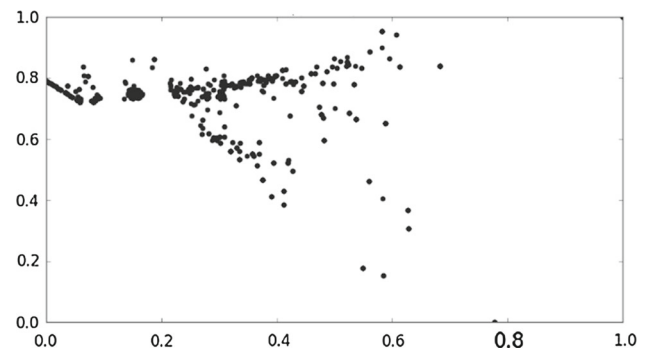


Fig. 6 Typical objects identified by Bagging-TPMiner for the running example

the Shaffer dynamic post hoc analysis [14], with a level of significance of $\alpha = 0.05$.

Figure 8 indicates that the classifier for the $RS_{\text{percent}} = 5\%$ test is significantly worse than the other tested options ($RS_{\text{percent}} \in \{10\%, 15\%, 20\%\}$). Moreover, there is no significant statistical difference among the following options: $RS_{\text{percent}} = 10\%$, $RS_{\text{percent}} = 15\%$, and $RS_{\text{percent}} = 20\%$. Based on these results, we recommend using our proposed Bagging-RandomMiner with $RS_{\text{percent}} = 20\%$ because this yielded the highest AUC among those with low values of RS_{percent} .

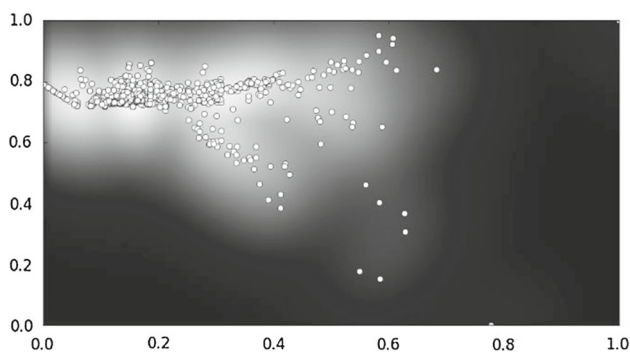


Fig. 7 Classification region of Bagging-TPMiner associated with the running example

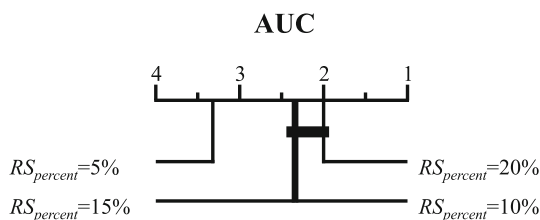


Fig. 8 CD diagram showing the statistical comparison of performance (in terms of AUC) of Bagging-RandomMiner for different values of the parameter $RS_{percent}$

5 Materials and methods

In this section, we present the materials and methods used throughout this paper. To demonstrate that our proposal obtains similar classification results for file access-based masquerade detection as the ones obtained using other classifiers, we compare it with several supervised multi-class and one-class classifiers (involving different paradigms). First, we compare our proposal against Bagging-TPMiner [32] because this method is a one-class classifier which has been reported to outperform others in masquerade detection. Thus, if our model outperforms Bagging-TPMiner, we then compare it against several state-of-the-art classifiers.

In Sect. 5.1, we detail the repository of datasets used and the cross-validation procedure executed for these datasets. In Sect. 5.2, we present the selected classifiers and the rationale behind the design decisions. In Sect. 5.2, we discuss the selected measures for assessing the performance of the tested classifiers. Finally, the statistical tests used for comparing classification results are described in Sect. 5.3.

5.1 Datasets and data preprocessing

For our experiments, we used the repository of datasets and the same data preprocessing technique introduced in Camiña et al. [6]. For preprocessing the datasets, first, each TrS is comprised of a user TrS (UTrS) and a masquerade TrS (MTrS), whereas each testing set (TeS) is comprised of a user

testing set (UTeS) and a masquerade testing set (MTeS). To build the different sets, we first divided the feature user logs for fivefold cross-validation (5-FCV), using for each cross-validation four subsamples to constitute the UTrS and the remaining ones as UTeS. Then, we balanced the classes to build the MTrS by selecting at random with replacement, processed windows from other user attacks until we reached the size of the UTrS used for training during the 5-FCV process. It is important to mention that we do not balance the datasets for the PBC4cip [28] method because this algorithm was designed for class-imbalance problems, as was Bagging-TPMiner. The MVS is built with the processed feature logs of the three types of attacks performed on each user computer (more details are available in Camiña et al. [6]).

After balancing the datasets, we performed the student’s t-statistic normalization and MinMax scaling of the features.

In summary, our repository contains 74 users which are described by 16 numerical features (locality features). For each user, on average, there are approximately 10,000 instances belonging to the *user* class (UTrS = 8000; UTeS = 2000) and 3000 instances belonging to the *attack* class (MTrS = 2000; MTeS = 1000).

5.2 Supervised classifiers and evaluation methodology

In our experiments, we test the following well-known classifiers as implemented in the Weka data-mining software tool [17]: KNN [1], AdaBoost.M1 [13], J48 [37], LogRegression [8], MLP [18], NaiveBayes [21], RandomForest [3], and SVM [35].

Additionally, we test the following classifiers:

- *PBC4cip* is a contrast pattern-based classifier designed for class-imbalance problems [28]. PBC4cip first extracts a collection of contrast patterns from several decision trees. Then, using the class-imbalance level of the training dataset, it weights the support of each pattern. Finally, a query instance is classified to the class with the highest sum of supports.
- *Bagging-TPMiner* is a novel one-class classifier ensemble that has achieved interesting results with the WUIL repository [32]. Bagging-TPMiner uses a bootstrap approach in the training phase, such as TreeBagger, paying equal attention to dense and sparse regions, thus capturing ordinary user behavior that is not commonly used in other techniques. Bagging-TPMiner has been reported to outperform other well-known one-class classifiers [32].
- *TreeBagger* is a bootstrap aggregating classifier [29] that generates n random decision trees using a subset of the training dataset. This subset is obtained by sampling with replacement. Each tree contains approx-

imately 63.2% of instances from the training dataset, whereas the remainder of a tree is filled with repeated instances. For classification, TreeBagger returns the highest voted class among the n decision trees regarding a query instance. In [6], the authors showed that TreeBagger achieved better results than the implementation of decision tree bagging included in the Weka data-mining tool [17].

To evaluate the performance of the classifiers, we use the following measures.

- AUC** *Area under the ROC curve* AUC evaluates the true positive (true masquerader) detection rate (TP_{rate}) versus the false positive (false masquerader) detection rate (FP_{rate}). This measure is commonly used for masquerade detection because it, unlike G-mean [26] and F-Measure [2], is an objective measure which is not affected by subjective factors and is insensitive to changes in the distribution of the training dataset [19].
- ZFP** *Zero-false positives* measures the number of true masqueraders (TP) when no false masquerader (FP) is detected; i.e., the value of TP when $FP = 0$. The higher the ZFP value, the more reliable the MDS. The main reason is that users justifiably find it disturbing to continuously receive false masquerader alerts in a short time interval.

We compute these performance indicators from ROC curves according to Fawcett [12] and will average the results of the 5-FCV performance indicators for each user.

5.3 Statistical tests

In supervised classification, commonly, the results produced by different classifiers are corroborated through statistical tests to determine whether these results are statistically different. Some nonparametric tests and post hoc procedures have been suggested by Demšar [9] to perform a comparison among different classification results. In this paper, we apply Friedman's test (as a nonparametric test), as suggested in Demšar [9]. Friedman's test reveals whether or not there are significant statistical differences among the compared classifiers. However, Friedman's test cannot determine which classifiers have statistical differences among them. Hence, we perform Shaffer's statistic procedure (as a post hoc procedure) as suggested by García [14]. We selected Shaffer's statistic procedure because it is more powerful than the classical Nemenyi and Holm procedures and it is less computationally expensive than Bergmann-Hommel's dynamic procedure García [14]. All statistical tests are executed with a level of significance of $\alpha = 0.05$ as proposed by García [14] and Demšar [9].

6 Experimental results and discussion

For our experiments, we first compare our proposal and Bagging-TPMiner to select the best one. Then, the classification results of the best classifier selected from the first comparison are contrasted with the classification results obtained by other state-of-the-art classifiers (see Sect. 5.2) by using the *Locality* datasets described in Sect. 5.1. To simplify the presentation, a supplementary material website³ has been created for this paper, which contains all experimental results, statistical test results, and all necessary information for downloading the repositories of datasets used in our experimentation.

6.1 Comparing Bagging-RandomMiner versus Bagging-TPMiner

Tables 1 and 2 show the compared methods (see the column comparison), the sum of ranks for the problems where Bagging-RandomMiner outperformed Bagging-TPMiner (see the column R^+), the sum of ranks for the opposite (see the column R^-), the result of the null hypothesis (see the column Hypothesis), and the p value computed by the Wilcoxon signed-rank test.

From Table 1, note that the null hypothesis was rejected. Consequently, concerning the AUC metric, our proposal obtains statistically better classification results than Bagging-TPMiner. Moreover, from Table 2, note that our proposal obtains classification results (regarding ZFP values) similar to those obtained by Bagging-TPMiner.

It is important to highlight that although our proposal significantly outperforms Bagging-TPMiner regarding the AUC but not the ZFP, our proposal has a computational complexity equal to $O(n)$ in the training phase, whereas Bagging-TPMiner has a computational complexity equal to $O(n^2)$. As a consequence, our proposal is significantly faster in the training phase and obtains better classification results regarding AUC than Bagging-TPMiner.

6.2 Comparing all selected classifiers against Bagging-RandomMiner

It is important to highlight that some of the tested classifiers produced errors regarding some users in the repository. For such experiments, we decided to remove the results from all the tested classifiers regarding these users to obtain a fair comparison among the tested classifiers. As a consequence, we show the classification results for 69 users from the 74 available in the repository described in Sect. 5.1.

Figure 9 shows a scatter plot of the average Friedman ranks according to AUC and ZFP for all the tested clas-

³ drive.google.com/file/d/0B9jNhJSXlx7GNTJKUHFqb0Rtdjg/view.

Table 1 Wilcoxon signed-rank test comparing the average AUC of Bagging-RandomMiner to the average AUC of Bagging-TPMiner, using all the tested datasets

Comparison	R^+	R^-	Hypothesis ($\alpha = 0.05$)	p value
Bagging-RandomMiner versus Bagging-TPMiner	1715.0	631.0	Rejected	0.000917

Table 2 Wilcoxon signed-rank test comparing the average ZFP of Bagging-RandomMiner to the average ZFP of Bagging-TPMiner, using all the tested datasets

Comparison	R^+	R^-	Hypothesis ($\alpha = 0.05$)	p value
Bagging-RandomMiner versus Bagging-TPMiner	1049.0	1297.0	Not Rejected	0.442268

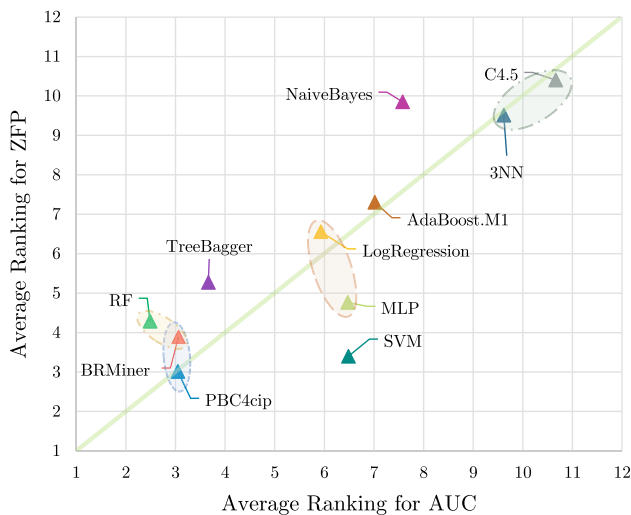


Fig. 9 Average ranking for the tested classifiers according to AUC versus ZFP

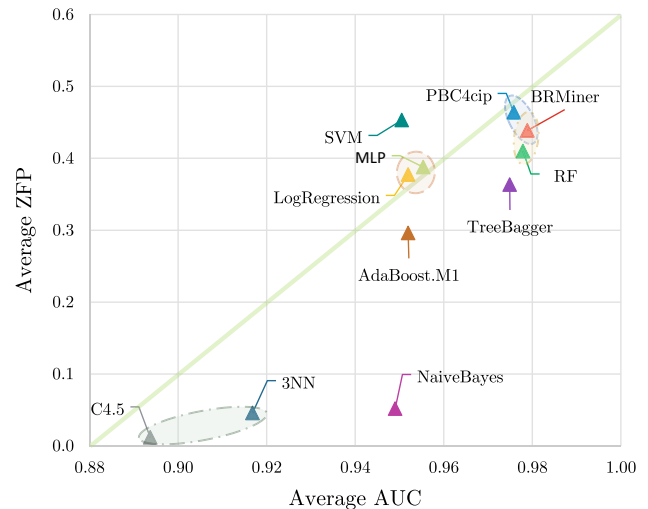


Fig. 10 Average AUC versus average ZFP for the tested classifiers

sifiers. In this figure, the best classifiers according to the AUC appear to the left, and the best classifiers according to ZFP appear at the bottom. Hence, the classifier closest to the coordinate origin (1,1) is the best one considering both performance metrics. Moreover, in this figure, those classifiers enclosed by an ellipse have no significant statistical differences among them regarding both evaluated measures. In Fig. 9, we can observe that our proposal (Bagging-RandomMiner), RandomForest, and PBC4cip obtain the best ranking results for both measures (AUC and ZFP). Further, note that Bagging-RandomMiner is not statistically different from RandomForest and PBC4cip models.

Figure 10 shows a scatter plot of the average classification results according to AUC and ZFP for all the tested classifiers. In this figure, the best classifiers according to the AUC appear to the right, and the best classifiers according to ZFP appear at the top. Hence, the classifier closest to the upper right corner is the best one considering both performance metrics. Additionally, in this figure, those classifiers

enclosed by an ellipse have no statistical difference among them regarding both evaluated measures. In Fig. 10, once again Bagging-RandomMiner, RandomForest, and PBC4cip models obtain the best results for the two metrics (AUC and ZFP), with Bagging-RandomMiner being the best classifier when calculating AUC, although it shows no statistical difference with respect to PBC4cip and RandomForest.

From Figs. 9 and 10, we can conclude that Bagging-RandomMiner, RandomForest, and PBC4cip have no statistical differences among them and that they outperform the results obtained for AUC and ZFP by other well-known supervised classifiers in file access-based masquerade detection problems.

PBC4cip obtains a collection of contrast patterns that describe the masquerade detection problem in terms of an expert’s language; however, in a real scenario, it is impossible to obtain valid samples of every possible attack, and as a consequence, PBC4cip cannot be applied. For this type of real scenario, the best choice is to use a one-class classifier because it does not require any prior knowledge of

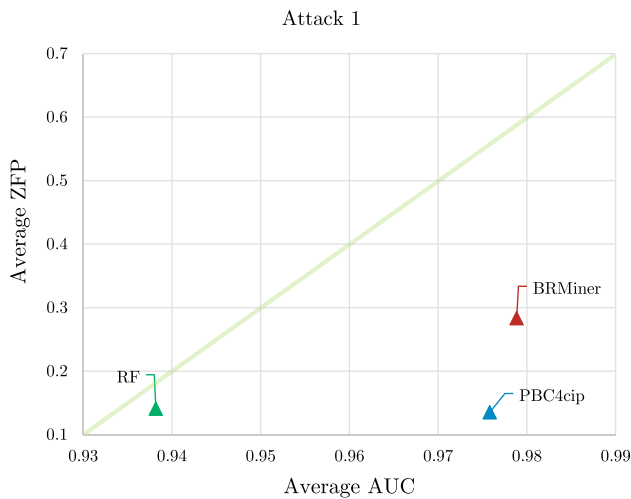


Fig. 11 Average AUC versus average ZFP of the classifiers Bagging-RandomMiner, RandomForest, and PBC4cip for predicting attack 1 types

attack types, as it is trained with only genuine user information. To evaluate the performance of Bagging-RandomMiner, RandomForest, and PBC4cip in a more realistic scenario, in Sect. 6.3, we conduct a comparison among these three classifiers by using a modification of the repository of datasets presented in Sect. 5.1.

6.3 Comparing Bagging-RandomMiner, RandomForest, and PBC4cip

This section aims to evaluate the performance of Bagging-RandomMiner against PBC4cip and RandomForest in a more realistic scenario. Consequently, we can determine which of these three classifiers obtain the best classification result on file access-based masquerade detection. To accomplish this, we first modify the datasets presented in Sect. 5.1 and perform an experiment for classifying query instances belonging to a type of attack for which the classifier was not trained. Finally, these classification results are corroborated by using Friedman's nonparametric statistical test proposed in Demšar [9].

We modified the repository presented in Sect. 5.1 as follows: For each training dataset, we removed all instances belonging to a type of attack, and for the corresponding testing dataset, we removed those instances belonging to the remaining types of attacks in the training dataset. Consequently, we created three repositories (one for each type of attack) from the original locality repository presented in Sect. 5.1 but each training dataset contains no information on the three types of attacks to be identified.

Figure 11, shows a scatter plot of the average classification results according to AUC and ZFP for Bagging-RandomMiner, RandomForest, and PBC4cip. In this figure, the best classifiers according to the AUC appear to the right,

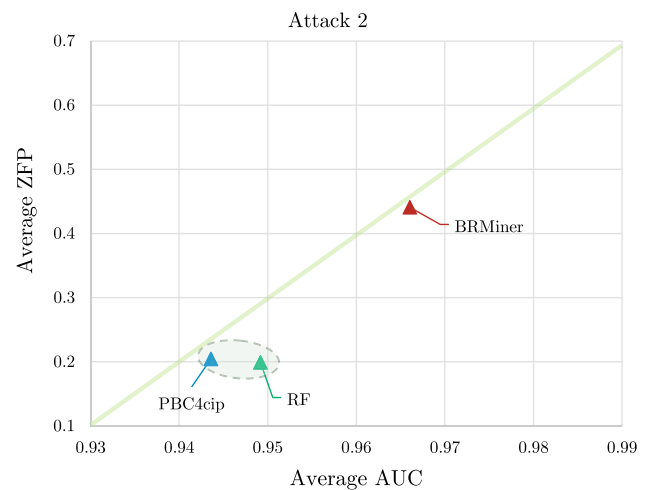


Fig. 12 Average AUC versus average ZFP of the classifiers Bagging-RandomMiner, RandomForest, and PBC4cip for predicting attack 2 types

and the best classifiers according to ZFP appear at the top. Hence, the classifier closest to the upper right corner is the best one considering both performance metrics. From Fig. 11 and Friedman's test, we can state that Bagging-RandomMiner exhibits better performance and is statistically different to PBC4cip and RandomForest for both AUC and ZFP for type 1 (basic) attacks.

Figure 12 presents a scatter plot of the average classification results according to AUC and ZFP for Bagging-RandomMiner, RandomForest, and PBC4cip, similar to Fig. 11, but for type 2 (intermediate) attacks. From this figure and Friedman's test, we can say that Bagging-RandomMiner significantly outperforms PBC4cip and RandomForest for both AUC and ZFP. Note also how the difference according to ZFP is larger than the difference shown in Fig. 11. The main reason is that the type 2 attack is more interactive than the type 1 attack because some specific files are found and copied to a USB flash drive, whereas for type 1 attacks, files are not copied. In addition, according to Friedman's nonparametric statistical test, there is no statistical difference between RandomForest and PBC4cip.

Figure 13 presents a scatter plot of the average classification results according to AUC and ZFP for Bagging-RandomMiner, RandomForest, and PBC4cip, similar to Figs. 11 and 12 but for type 3 (advanced) attacks. From this figure and the statistical test used, we can state that Bagging-RandomMiner outperforms and is statistically different to PBC4cip and RandomForest for both ZFP and AUC measurements. Although Bagging-RandomMiner had statistical differences in the ZFP and AUC metrics to RandomForest and PBC4cip, it can be seen that for the AUC there is not as much difference as there is for ZFP. The main reason for this result is that some automated scripts were executed

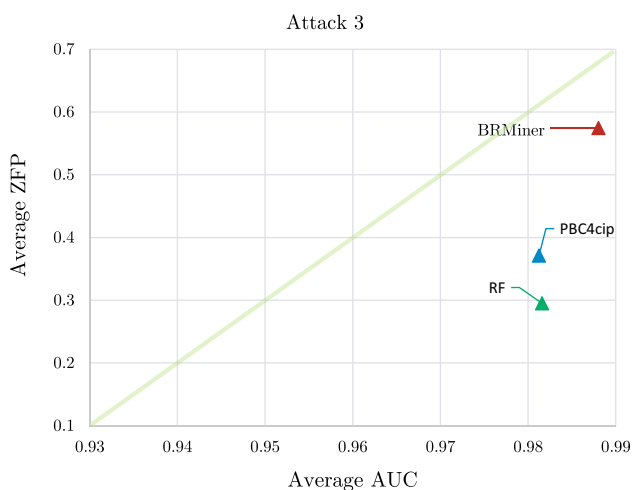


Fig. 13 Average AUC versus average ZFP for Bagging-RandomMiner, RandomForest, and PBC4cip for predicting attack 3 types

on each users computer in type 3 attacks and consequently, these attacks could produce similar behaviors. Compared to automated scripts, human behaviors are more diverse, and therefore, they are more difficult to generalize by a classifier designed for multi-class problems.

Finally, based on all the experimental results, we conclude that the Bagging-RandomMiner classifier is the best classifier for file access-based masquerade detection among the tested classifiers.

7 Conclusions and further work

The need for protecting information has prompted research on MDSs aiming to detect an attack given a user behavior profile. Existing approaches to user profiling have focused mainly on one type of activity, usually user actions (command usage, keyboard usage, and so on). In this case, the access-patterns approach has reported better classification results for masquerade detection than other approaches designed for this type of problem.

In this paper, we introduced a one-class classifier for file access-based masquerade detection, which obtains significantly better classification results than several state-of-the-art classifiers. We corroborated our hypothesis that, in file access-based masquerade detection, a one-class classification approach outperforms the multi-class approach. This hypothesis is supported by an in-depth experimental analysis for file access-based masquerade detection taking advantage of the one-class classification approach. Furthermore, unlike classifiers not based on one-class classification, these results indicate that one-class classification obtains high-quality classification results when unknown attacks arise.

Additionally, we introduced three repositories of datasets for identifying each of the three types of attacks introduced in the WUIL repository without instances in the training dataset belonging to the type of attack to be identified in the testing dataset. These new repositories could be used for testing future classifiers simulating attacks perpetrated in a real scenario.

In the future, we will focus on the correlation analysis between users' performance and their features' values, with the aim of finding user behavior rules that can be followed to better protect users. Furthermore, modern data storage is moving to cloud services and to repositories where different users can access the information of other users. Therefore, in a future study, we plan to develop a new FSN repository based on cloud services and repositories of data storage file accesses.

Acknowledgements We wish to express our gratitude to the members of the GIEE-ML group at Tecnológico de Monterrey for providing useful suggestions and advice on earlier versions of this paper.

References

1. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. *Mach. Learn.* **6**(1), 37–66 (1991). <https://doi.org/10.1007/BF00153759>
2. Baeza-Yates, R.A., Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
3. Breiman, L.: Random forests. *Mach. Learn.* **45**(1), 5–32 (2001). <https://doi.org/10.1023/A:1010933404324>
4. 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.) *Proceedings of the 10th Mexican International Conference on Artificial Intelligence (MICAI 2011)*, pp. 174–186. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-25324-9_15
5. 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**(3), 919–930 (2014). <https://doi.org/10.1016/j.eswa.2013.08.022>. *Methods and Applications of Artificial and Computational Intelligence*
6. Camiña, J.B., Monroy, R., Trejo, L.A., Medina-Pérez, M.A.: Temporal and spatial locality: an abstraction for masquerade detection. *IEEE Trans. Inf. Forensics Secur.* **11**(9), 2036–2051 (2016). <https://doi.org/10.1109/TIFS.2016.2571679>
7. 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.) *Proceedings of the 17th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2014)*, pp. 447–465. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_22
8. Cessie, S.L., Houwelingen, J.C.V.: Ridge estimators in logistic regression. *J. R. Stat. Soc. Ser. C (Appl. Stat.)* **41**(1), 191–201 (1992)
9. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.* **7**, 1–30 (2006)

10. Denning, D.E.: An intrusion-detection model. *IEEE Trans. Softw. Eng.* **SE-13**(2), 222–232 (1987). <https://doi.org/10.1109/TSE.1987.232894>
11. Duda, R.O., Hart, P.E., Stork, D.G.: *Pattern Classification*, 7th edn. Wiley-Interscience, Hoboken (2012)
12. Fawcett, T.: An introduction to ROC analysis. *Pattern Recognit. Lett.* **27**(8), 861–874 (2006). <https://doi.org/10.1016/j.patrec.2005.10.010>. ROC Analysis in Pattern Recognition
13. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148–156 (1996)
14. García, S., Herrera, F.: An extension on “statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *J. Mach. Learn. Res.* **9**, 2677–2694 (2008)
15. Garg, A., Rahalkar, R., Upadhyaya, S., Kwiat, K.: Profiling users in GUI based systems for masquerade detection. In: *IEEE Information Assurance Workshop*, pp. 48–54 (2006). <https://doi.org/10.1109/IAW.2006.1652076>
16. Gates, C., Li, N., Xu, Z., Chari, S.N., Molloy, I., Park, Y.: Detecting insider information theft using features from file access logs. In: *Kutyłowski, M., Vaidya, J. (eds.) Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, pp. 383–400. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_22
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. *SIGKDD Explor. News.* **11**(1), 10–18 (2009). <https://doi.org/10.1145/1656274.1656278>
18. Haykin, S.S.: *Neural Networks: A Comprehensive Foundation*, 2nd edn. Tsinghua University Press, Beijing (2001)
19. Japkowicz, N.: Assessment Metrics for Imbalanced Learning. In: He, H., Ma, Y. (eds.) *Imbalanced Learning: Foundations, Algorithms, and Applications*, Chap. 8, pp. 187–206. Wiley, New York (2013). <https://doi.org/10.1002/9781118646106.ch8>
20. Jian, Z., Shirai, H., Takahashi, I., Kuroiwa, J., Odaka, T., Ogura, H.: Masquerade detection by boosting decision stumps using unix commands. *Comput. Secur.* **26**(4), 311–318 (2007). <https://doi.org/10.1016/j.cose.2006.11.008>
21. John, G.H., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, UAI'95*, pp. 338–345. Morgan Kaufmann Publishers Inc., San Francisco (1995). <http://dl.acm.org/citation.cfm?id=2074158.2074196>
22. Kholidy, H.A., Baiardi, F., Hariri, S.: DDSGA: a data-driven semi-global alignment approach for detecting masquerade attacks. *IEEE Trans. Dependable Secure Comput.* **12**(2), 164–178 (2015). <https://doi.org/10.1109/TDSC.2014.2327966>
23. Killourhy, K., Maxion, R.: Why did my detector do that?!. In: Jha, S., Sommer, R., Kreibich, C. (eds.) *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 256–276. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-15512-3_14
24. Killourhy, K.S., Maxion, R.A.: Comparing anomaly-detection algorithms for keystroke dynamics. In: *International Conference on Dependable Systems Networks (IFIP)*, pp. 125–134 (2009). <https://doi.org/10.1109/DSN.2009.5270346>
25. Kim, H.S., Cha, S.D.: Empirical evaluation of SVM-based masquerade detection using unix commands. *Comput. Secur.* **24**(2), 160–168 (2005). <https://doi.org/10.1016/j.cose.2004.08.007>
26. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: one-sided selection. In: *14th International Conference on Machine Learning (ICML97)*, pp. 179–186 (1997)
27. Kudłacik, P., Porwik, P., Wesolowski, T.: Fuzzy approach for intrusion detection based on user’s commands. *Soft. Comput.* **20**(7), 2705–2719 (2016). <https://doi.org/10.1007/s00500-015-1669-6>
28. Loyola-González, O., Medina-Pérez, M.A., Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Monroy, R., García-Borroto, M.: PBC4cip: a new contrast pattern-based classifier for class imbalance problems. *Knowl. Based Syst.* **115**, 100–109 (2017). <https://doi.org/10.1016/j.knsys.2016.10.018>
29. MathWorks, Inc.: Treebagger (2015). <http://www.mathworks.com/help/toolbox/stats/treebagger.html>
30. Maxion, R.A.: Masquerade detection using enriched command lines. In: *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 5–14 (2003). <https://doi.org/10.1109/DSN.2003.1209911>
31. Maxion, R.A., Townsend, T.N.: Masquerade detection using truncated command lines. In: *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 219–228 (2002). <https://doi.org/10.1109/DSN.2002.1028903>
32. Medina-Pérez, M.A., Monroy, R., Camiña, J.B., García-Borroto, M.: Bagging-TPMiner: a classifier ensemble for masquerader detection based on typical objects. *Soft. Comput.* **21**(3), 557–569 (2017). <https://doi.org/10.1007/s00500-016-2278-8>
33. Messerman, A., Mustafi, T., Camtepe, S.A., Albayrak, S.: Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics. In: *International Joint Conference on Biometrics (IJCB)*, pp. 1–8 (2011). <https://doi.org/10.1109/IJCB.2011.6117552>
34. Morales, A., Fierrez, J., Ortega-García, J.: Towards predicting good users for biometric recognition based on keystroke dynamics. In: *Agapito, L., Bronstein, M.M., Rother, C. (eds.) Proceedings of the Workshop on Computer Vision (ECCV 2014)*, pp. 711–724. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-16181-5_54
35. Platt, J.C.: Fast training of support vector machines using sequential minimal optimization. In: *Schölkopf, B., Burges, C.J.C., Smola, A.J. (eds.) Advances in Kernel Methods*, pp. 185–208. MIT, Cambridge, MA, USA (1999)
36. Pusara, M., Brodley, C.E.: User re-authentication via mouse movements. In: *Proceedings of the Workshop on Visualization and Data Mining for Computer Security, VizSEC/DMSEC '04*, pp. 1–8. ACM, New York (2004). <https://doi.org/10.1145/1029208.1029210>
37. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., Los Altos (1993)
38. Rodríguez, J., Cañete, L., Monroy, R., Medina-Pérez, M.A.: Experimenting with masquerade detection via user task usage. *Int. J. Interact. Des. Manuf. (IJIDeM)* **11**(4), 771–784 (2016). <https://doi.org/10.1007/s12008-016-0360-1>
39. Salem, M.B., Stolfo, S.J.: Modeling user search behavior for masquerade detection. In: *Sommer, R., Balzarotti, D., Maier, G. (eds.) Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection*, pp. 181–200. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-23644-0_10
40. Saljooghinejad, H., Bhukya, W.N.: Layered security architecture for masquerade attack detection. In: *Cuppens-Boulahia, N., Cuppens, F., Garcia-Alfaro, J. (eds.) Proceedings of the 26th Conference on Data and Applications Security and Privacy*, pp. 255–262. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-31540-4_19
41. Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y.: Computer intrusion: detecting masquerades. *Stat. Sci.* **16**(1), 58–74 (2001)
42. Shen, C., Cai, Z., Guan, X., Maxion, R.: Performance evaluation of anomaly-detection algorithms for mouse dynamics. *Comput. Secur.* **45**, 156–171 (2014). <https://doi.org/10.1016/j.cose.2014.05.002>
43. Song, Y., Salem, M.B., Hershkop, S., Stolfo, S.J.: System level user behavior biometrics using fisher features and gaussian mix-

- ture models. In: IEEE Security and Privacy Workshops, pp. 52–59 (2013). <https://doi.org/10.1109/SPW.2013.33>
44. Vidal, J.M., Orozco, A.L.S., Villalba, L.J.G.: Online masquerade detection resistant to mimicry. *Expert Syst. Appl.* **61**, 162–180 (2016). <https://doi.org/10.1016/j.eswa.2016.05.036>
 45. Wang, K., Stolfo, S.J.: One-class training for masquerade detection. In: *Workshop on Data Mining for Computer Security*, p. 10. Citeseer (2003)
 46. Wang, X., Sun, Y., Wang, Y.: An abnormal file access behavior detection approach based on file path diversity. *IET Conference Proceedings*, pp. 455–459 (2014). <http://digital-library.theiet.org/content/conferences/10.1049/cp.2014.0632>
 47. Wang, X., Wang, Y., Liu, Q., Sun, Y., Xie, P.: Insider detection by analyzing process behaviors of file access. In: Park, J.J.J.H., Yi, G., Jeong, Y.S., Shen, H. (eds.) *Advances in Parallel and Distributed Computing and Ubiquitous Services (UCAWSN & PDCAT)*, pp. 209–219. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-0068-3_28
 48. Weiss, A., Ramapanicker, A., Shah, P., Noble, S., Immohr, L.: Mouse movements biometric identification: a feasibility study. *Proc. Student/Faculty Research Day CSIS. Pace University, White Plains* (2007)



José Benito Camiña obtained a Ph.D. degree in Computer Science in 2015 from Tecnológico de Monterrey, Campus Estado de México, under the supervision of Prof. Raúl Monroy. Currently, he is a member of the GIEE-ML (Machine Learning) Research Group. Camiña's research is concerned with the use of machine learning techniques for masquerade detection.



Miguel Angel Medina-Pérez received Ph.D. degree in Computer Science from the National Institute of Astrophysics, Optics, and Electronics, Mexico, in 2014. He is currently a Research Professor with the Tecnológico de Monterrey, Campus Estado de Mexico, where he is also a member of the GIEE-ML (Machine Learning) Research Group. His research interests include supervised classification, clustering, data mining, big data, feature selection, one-class classification,

masquerader detection, fingerprint recognition, and palmprint recognition. Prof. Medina-Pérez has been involved in several research projects about pattern recognition, and he has published several papers in referenced journals such as "Information Fusion," "Pattern Recognition," "Knowledge-Based Systems," "Information Sciences," and "IEEE Transactions on Information Forensics and Security". The Ministry of Higher Education of Cuba recognized Prof. Medina-Pérez with the National Award to the result of highest importance and scientific originality in Cuba, in 2016. Prof. Medina-Pérez also received the National Award of the Cuban Academy of Sciences to research students of Technical Sciences in 2006.



Raúl Monroy obtained a Ph.D. in Intelligence in 1998 from Edinburgh University, under the supervision of Prof. Alan Bundy. He is a (full) Professor in Computing at Tecnológico de Monterrey, Campus Estado de Mexico. Since 1998, he is a member of CONA-CyT's National Research System, currently ranked 2. He is the leader of the GIEE-ML (Machine Learning) Research Group at Tecnológico de Monterrey. Dr. Monroy's research focuses on the discovery of novel ML methods for anomaly detection (applied especially in cybersecurity); the automated use of theorem proving to formal methods of system development; the discovery of an application of general search control strategies for uncovering and correcting errors in either a system or its specification; and motion planning.



Octavio Loyola-González graduated from University of Ciego de Ávila, Cuba, in 2010. He received M.Sc. degree in Applied Informatics from University of Ciego de Ávila in 2012. He received his Ph.D. in Computer Science in 2017 from National Institute of Astrophysics, Optics and Electronics, Mexico. Currently, he is a Postdoctoral Researcher at Tecnológico de Monterrey's State of Mexico Campus, where he is also a member of the GIEE-ML (Machine Learning) Research

Group. His research interests are pattern recognition, data mining, class imbalance problems, and biometrics. Loyola-González has been awarded by the Cuban Ministry of Higher Education (2016) according to the "Result of Major Transcendence and Originality Scientific" in Cuba. He has published several research articles in international journals and conferences. He is a professor in undergraduate and graduate programs of Computer Sciences at Tecnológico de Monterrey's State of Mexico Campus.



Luis Angel Pereyra Villanueva obtained his degree in Mathematical Engineering at the Universidad Autónoma de Chihuahua (MEX) in 2016. In 2017, he joined Master's Degree in Computer Engineering at the same university. His current research interests are pattern recognition, one-class classification problems, and big data analysis.



Luis Carlos González Gurrola obtained his Ph.D. from The University of North Carolina at Charlotte (USA) in 2011. The same year, he joined the Universidad Autónoma de Chihuahua, in the north of Mexico, where he was a founder (and now a professor) of the graduate program of Computer Engineering. Currently, he is a member of the National System of Researchers of the National Council of Science and Technology of México (CONACyT). His current research interests are pat-

tern recognition and combinatorial optimization problems.