This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Temporal and Spatial Locality: an Abstraction for Masquerade Detection

J. Benito Camiña, Raúl Monroy, Luis A. Trejo, and Miguel Angel Medina-Pérez

***Abstract*—Most research in masquerade detection focus mainly on the user action, ignoring the object upon which that action is performed. This may yield limited models since; for example, command execution (an action) usually ends up in the transformation of a file (the object). The overall goal of our research is to prove that the object is paramount to distinguishing a user from a masquerade. With this in mind, we have developed a new approach to masquerade detection, called *file system navigation*, and tested our ideas using the *Windows-Users and -Intruder simulations Logs Dataset* (WUIL), which unlike other datasets of its kind includes close-to-real simulated attacks. We have shown that our approach makes it possible to capture computer behavior in an abstract way difficult to realize in a purely action-based approach. In this paper we introduce an abstraction called *locality*, the tendency of programs to cluster references to memory. While temporal locality is applicable to both actions and objects, spatial locality is more suitable to objects, as it depends on a notion of position. We have successfully validated our working hypothesis: locality-based features better capture user behavior for masquerade detection. Particularly, results based on our approach report an Area Under the Curve of the ROC curve (AUC) value of 0.97 in average with 30% of users having an AUC equal to or above 0.99.**

***Index Terms*—Masquerade Detection, Information Security.**

I. Introduction

Computers nowadays handle critical information; unfortunately, this increase in computer dependence has not come with an increase in the use of appropriate mechanisms for securing information. It is still common to find users with poor computer security habits: for example, users set weak passwords or leave their computer sessions unattended. This situation has prompted the development of *Masquerade Detection Systems* (MDS). The objective of such systems is to raise an alert when computer behavior differs to a certain extent from common computer behavior, as profiled from a history of computer sessions. MDS are frequently seen as continuous authentication methods, which attempt to establish whether it is the genuine user whoever is in possession of a target device.

One of the first works that improved on the area sought to create an MDS considering the UNIX commands commonly employed by a user [1]. Subsequently, people explored very diverse types of user activity. among them, we have mouse usage [2], [3], [4], where user profile is built out of different patterns of mouse clicks and movements; keystroke dynamics [5], [6], [7], [8], where it is built from different keyup/keydown times; and file usage [9], [10], [11], where a user profile is built from file access history [11], [10].

Most research in MDS focus mainly on the user action, overlooking the object upon which the action is performed. For example, research in keystroke dynamics considers that keys are identical. This, however, is a very strong assumption, since, for example, the position of a key may impact keyup-keydown usage time, or the user language may affect the likelihood of a key being typed. The overall hypothesis of our research is that the object upon which an action is carried out is paramount to distinguishing a user from a masquerade. Fortunately, there exist contexts where this object information is readily available. For example, the password being typed, the URL being clicked upon, etc.; we should strive towards making use of such information, whenever available. With this in mind, we have developed a new approach to masquerade detection, called *file system navigation* [12], and tested our ideas using the *Windows-Users and -Intruder simulations Logs Dataset* (WUIL) [13], a dataset repository that contains file access logs from 20 users and includes close-to-real masquerade attacks.

In the file system navigation approach, a user profile is built from how a user navigates her file system structure. We have shown that our approach makes it possible to capture user behavior in an abstract way that cannot be realized in a purely command-based approach. An example abstraction is a notion of *tasks* [14], where each file system object, a file or a folder, is related with a specific user activity, such as job, entertainment, etc. There, we have shown that if we build a classifier considering only task usage, ignoring the associated command altogether, we then get roughly the same masquerade detection performance as though we used all the objects, but at a lower computational cost. Combining both commands and tasks/objects to investigate whether it increases overall performance detection is a path worth exploring, but out of the scope of this paper.

The main contribution of the present work is a new feature abstraction called *locality*, based on the memory-cache term of the same name. Locality can be either temporal, which captures repetition of events, or spatial, which captures occurrence of neighbor events. Temporal locality is to some extent readily used in the action-based approach. However, spatial locality is not, because it depends on a notion of position. Position is straightforwardly defined in the file system navigation approach using the directory tree structure. We have successfully validated our working hypothesis that locality-based features better capture user behavior for masquerade detection. Particularly, results based on our approach report

The authors are all with Tecnologico de Monterrey, Escuela de Ingeniería y Ciencias, Carretera al Lago de Guadalupe Km. 3.5, Atizapán, Estado de México, 52926, México. Email: {jb.camina, raulm, ltrejo, migue}@itesm.mx.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

an Area Under the Curve of the ROC curve (AUC) value of 0.97 in average with 30% of users having an AUC equal to or above 0.99. We have obtained these results using a TreeBagger classifier, and so in order to validate that they are not due to the classifier but to the features, we have built a second classifier of the same kind but using the features introduced in [13], we call *WUIL features*. Our results show that the classifier built from locality features attained a better performance than the one based on WUIL features. We shall have more to say on this in § V, where we also compare our detection method against those reported in [11].

The results above mentioned were obtained using the newly extended WUIL dataset repository, which involves a large update. In particular, extended WUIL contains logs for 73 users (the originally WUIL repository contained only 20); newly added users mostly run more recent versions of the MS Windows operating systems, including 8 and 8.1. This enhancement places WUIL as one of the largest and most comprehensive dataset repository in the masquerade detection domain.

A. Overview of Paper

The remainder of this paper is organized as follows. First in §II we overview different types of user behavior that have been studied for masquerade detection. Then, in §III, we give an outline of the WUIL dataset repository and introduce the update made to increase the dataset from 20 to 73 users. After that, in §IV, we introduce our locality abstraction and then all the locality features extracted from a WUIL user set of logs. Next, in §V, we describe how each dataset in WUIL is preprocessed, as well as how we have set our classification experiments. Then, we show the output of testing our locality-based classifier on the WUIL repository, and compare them against those previously obtained using WUIL features [13]. In §V, we also carry out a feature selection exercise in order to assess the relevance of our locality features for detection performance, and compare our method against others found in the literature. Finally, in §VI, we report on the conclusions drawn from this study and provide guidelines for further work.

II. User Behaviors for Masquerade Detection

Several different types of user behavior have been studied to design an effective MDS. In this section, we overview related work and organize our review in terms of the activity used to profile user behavior: command usage, hardware user interaction, and file usage.

A. Command-Based Masquerade Detection

A major breakthrough in masquerade detection has been the work of Schonlou *et al.* [1]. There, the objective is to detect a masquerader based on the commands that the target user commonly executes. To achieve this, Schonlou *et al.* developed a dataset, called *SEA* [15]. SEA is comprised of logs of UNIX commands, to each of which the corresponding arguments have been stripped off. SEA lacks from masqueraders attempts; to face this issue, Schonlou *et al.* simulate an attack by "infecting" a user's logs with portions of logs of other users. SEA is publicly available and has been actively used to compare different methods for masquerade detection, or to set up different masquerade experimentations. An example work in this latter vein is [16], where Maxion *et al.* suggested that to build a MDS for a user one should use information from both the user and every potential masquerader.

In [17], Maxion has sought to prove that using both commands and arguments one may surpass the performance results obtained with SEA. For that purpose, he used *Greenbergs* [18], a dataset that includes these two types of information. Greenbergs dataset contains logs of four different types of users: non-programmers, novice programmers, experienced programmers, and computer scientists. Similar to SEA, this dataset also lacks from attack attempts, and hence to simulate a masquerade Maxion makes use of other users' ordinary logs.

The use of UNIX is not unique in command-based masquerade detection. Masri *et al.* [19], for example, have developed a MDS based on the commands executed by a user when interacting with Microsoft (MS) Word. Example commands include "Copy", "Paste", "New", or "Bold". In their investigation, Masri *et al.* used the dataset reported in [20], which contains MS Word session logs of 24 users. In average, each user is given by 539 sessions, and each session is in turn given by 6.57 commands, taken from a set of 174 different commands. A supervised learning approach is used for building a classifier with information from both the normal user and the other users, artificially set as masqueraders.

B. Human-Computer Interaction Masquerade Detection

Human-Computer Interaction (HCI) has also been widely used for masquerade detection, being mouse- and keyboard-usage the most common. We give an outline of this research thereupon.

1) Mouse Usage: Mouse usage is perhaps the most studied HCI activity. The results reported in [2], [3], [4] are all similar, except for a slightly different motivation, namely: user re-authentication, biometric-based user identification, and masquerade detection. In all of these works, the authors independently developed a dataset, which contain pretty similar information about mouse usage. Example collected mouse activities are moves, drags, clicks, event times, pointer coordinates, etc. Interestingly, [2], [3], [4] do not use that information directly; instead, they all use it to extract dynamic features, involving speed, curve, curve length, among others. The datasets differ of course in the number of user specimens, and in the extent of each user record. The work of [2], [3], [4] also differ in the scenarios considered for gathering HCI mouse information. In particular, the results of Garg *et al.* [4] and Pusara & Brodley [2] are more general in that they respectively consider a user interacting with a GUI, or a user executing MS Internet Explorer to browse over the Internet. Weiss *et al.*s' work, [3], on the other hand, is more specific as it assumes a user interacting with a 25-button matrix application. In all these investigations, authors used a supervised learning approach, training a (multi-class) classifier with sample mouse trace activities from both the target user and the remaining users, artificially set as masqueraders.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

2) Keyboard Usage: Besides mouse usage, human keyboard interaction has been widely explored for masquerade detection. In this context, a key piece of work is that of Killourhy & Maxion [5], [6]. There, a dataset of 51 users was developed gathering information about how they typed the same password 400 times; the records were captured in different sessions of 50 repetitions each. They extracted 31 time features, including hold time, keydown-keydown time, and keyup-keydown time. Killourhy & Maxion trained several classification algorithms using different methodologies, varying the amount of training samples, applying classification update or not, and even considering various levels of impostor practice. In classifier construction, they used only the target user profile. However, they include no masquerade attempts; instead, Killourhy & Maxion use other users' password typing information. In [7], the authors used Killourhy & Maxion's dataset, but with the goal of predicting which users are easier to protect when a keystroke dynamics based MDS is used. They considered five statistical measures: variance, structural content, entropy, Kullback-Leibler divergence, and genuine scores; best results were obtained with Kullback-Leibler.

The methods of [5] are only applicable when both user and intruder type the same text, in this case a password. In a complementary vein, Messerman *et al.* [8] approached masquerade detection without this restriction; so, a user is allowed to type text freely. For that purpose, they built a dataset that gathers typing information of 55 users while they were making use of a web-mail application.

C. Search Patterns

Ben-Salem & Stolfo [21], [22] have analyzed how to use user search patterns for masquerade detection. They developed a dataset, called *RUU*, which contains logs of 18 users. For each, RUU collects user search patterns; however, it also collects operating system activity, such as registry modification, process information, GUI usage, file system access, and DLL libraries usage. For simulating masquerade attempts, Ben-Salem & Stolfo considered three scenarios, involving a malicious attacker, a rather benign attacker and a neutral user. Attack simulation was carried out in a computer other than the users, though.

In a similar vein, Denning [9] introduced a model for intrusion detection that monitors several system audit records. The model involves a lot of information, including user logins, user command execution, file system access, and device usage, among others. Remarkably, Denning considered to include, as part of a user model, information of a subject (the initiator of an action) and any associated objects (the receptors of an action).

D. Files

Interestingly, file usage for masquerade detection has recently attracted increasing attention. So, more related to ours are the works of [10], [11]. We describe each of these methods in a more elaborate level of detail below, for they will be useful for comparison purposes.

1) FPD: In [10], Wang *et al.* introduced a method for detecting malicious processes. The method makes use of an anomalous measure, called File Path Diversity (FPD), that takes the path diversity involved in a process's file system access. Roughly, Wang *et al.* start by defining an FPD function, fpd, that takes a file, f, and a most popular path, p; if f is a descendant of p, then fpd equals to 0; otherwise, it equals to $9/3^{\text{length}(\text{lca}(f,p))}$, 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 $f_0,\ldots,f_n$, and P, a set of most popular paths, Wang *et al.* mark w as abnormal if:

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

where u is a predefined threshold. Finally, a process is taken to be malicious if FPD finds in it at least k windows abnormal.

For training, Wang *et al.* gathered 7-day records of files being accessed by different processes in a computer system. For testing, they installed 10 different Trojans in the computer. Both sets, training and testing, were divided in time windows of 20 seconds; the FPD measures were calculated on a per process basis. Then, ten people were asked to use these Trojans for a 10-minute period seeking for sensitive information. They successfully detected 90% of malicious processes, True Positive Rate (TPR), with a 0% False Positive Rate (FPR), when $k = 13$. Notice how this evaluation methodology is rather soft, as the authors consider successful detection whenever 13 out of 30 attack windows are considered abnormal. As shall be discussed later on in the text, §V, we display our results on a per window basis.

The main problem with FPD is how to compute P, the set of most popular paths. Unfortunately, [10] gives no clue whatsoever to this aim. This could be attributable to that computing P for a process is rather straightforward. However, doing so for a user, as is the case in our setting, is not. Actually, FPD resembles our notion of tasks, [12], [13], in that access to a descendant of a task (a file path) amounts to accessing the task itself. From our experiments, we have learnt that automatically finding these user tasks is quite a challenge, and yet when tasks are adequately selected we get a paramount improvement in detection performance.

2) File Similarity: In a different vein, Gates *et al.* [11] aim to detect insider users' malicious behaviors, like stealing files users have the privilege to access to. 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 history, A_j^H. Gates *et al.* work under a hypothesis based on these two conditions: 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 if files are similar to files previously accessed, then this is less likely to be malicious theft. There is a close relationship of this with our concept of temporal and spatial locality that we shall introduce in §IV. To measure similarity, authors appeal

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

to two observations. First, files under the same directory may be considered as more similar than files that are far apart in the file system. And second, files accessed by essentially the same set of users may be viewed as more similar than files accessed by a disjoint sets of users.

Gates *et al.* use a function that assigns a score for each file f when given access history A^H, namely:

$$\text{score}[f \mid A^H] = \text{agg}_{g \in A^H} \text{score}(f, g)$$

The $\text{score}(f, g)$ function dictates how two files relate to each other while the agg expresses how a single file f relates to the entire history A^H. They propose five ways to evaluate $\text{score}(f, g)$. The first strategy, NewUnique, is binary: it scores 0 if the two files are the same and 1 otherwise. The next three strategies, called full distance, Least Common Ancestor (LCA), and log LCA, are very similar to our concept of distance between two files; however, we use distance as a building block for our four spatial locality features, and not as a final feature for masquerade detection (see §IV-A). The fifth function, Access Similarity, uses the Jaccard distance to measure how similar two files are, depending on the set of users that access them. Next, they used three different techniques to evaluate the aggregation function: selecting the minimum of all scores (f versus all files in A^H), the average of all scores, and by computing the average of the k files in A^H that have the lowest score. Finally, to generate a final score for a user, they use two functions: either the sum of all scores or the average of all scores, referred to as *SumScore* and *AveScore*, respectively.

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. According to the ROC curves reported by the authors, and depending on the number of files injected into each user's normal pattern to simulate an attack, their best results at a 5% FPR are the following: AveScore attained approximately a 75% TPR; NewUnique and SumScore achieved a TPR approximately in the range of 89% to 98%. We only discuss the case of an impetuous attacker, which is similar to our masquerade detection approach. SumScore and AveScore uses log LCA as a $\text{score}(f, g)$ technique and $\min$ as the aggregation method. NewUnique uses binary as a $\text{score}(f, g)$ technique, $\min$ as the aggregation method, and SumScore to obtain the final score.

3) WUIL: In our case, we have developed two works following the idea of analyzing the interaction of the user with the OS. In the first work [13], we introduced our dataset repository called WUIL, which is publicly available, upon request, at `homepage.cem.itesm.mx/raulm/wuil-ds/`. WUIL comprises information from 20 users and attacks. In §III we will explain in detail how the dataset repository is conformed and introduce the recent updates performed to the repository. We selected 25 different features referred to as the *WUIL features*. These features are extracted from time-based windows and can be classified in five different groups: number of accesses, time between accesses, path distance between accesses, graph (navigation structure), and depth; for details regarding these features the reader is referred to [13].

In the second work [14], using the WUIL dataset repository we developed an abstraction called *user task*. A user task holds different objects from the same activity of the user. WUIL repository lacks from information of the task of each object so we developed an algorithm to convert files into tasks based on the depth of the files. The classification was made with a classifier based on frequency, Naïve Bayes, and with a classifier based on temporal dependencies, Markov chains. We obtained comparable results using the task abstraction but with less resources. The results were appealing and have motivated us to continue the research using the file access approach to detect masqueraders.

III. The Updated WUIL Dataset Repository

In the present work we experiment using the WUIL repository [13]. WUIL contains logs of accesses made by 20 users, while working normally in their computer, and three simulated masquerader attacks performed in each user's computer. Different flavors of Windows were used, such as: MS Windows XP, MS Windows Vista, and MS Windows 7 OS. The logs gathered data from accesses made to the folders *My Documents* and *Desktop*.

To record accesses the MS Windows tool named *Audit* is used. Audit gathers information for the date, the time, the file access path, and other kind of information. The structure of our logs has the following fields: *ENTRY_ID*, that is a sequential number used as a unique identifier, starting from 0; *DATE*, the date when the access was made; *TIME*, the time when the access was made; *ELAPSED_TIME*, that is the time of the access in seconds from an epoch established on January 1st, 2011 at 00:00:00; *DEPTH*, the length of the object path; and finally, *OBJECT*, the path and the name of the object being accessed.

WUIL also contains logs of three kind of attacks performed on each user's computer: basic, intermediate, and advanced. The basic is when the masquerader is not prepared to conduct the attack so he only can access and see the content of the files manually. The intermediate is when the masquerader is prepared and brings a USB flash drive, searches files manually with interesting names using the search utility of MS Windows and copy them into the USB flash drive. The advanced is when the masquerader has computer skills and brings a USB flash drive with a script that searches and copies files of interest into the USB flash drive.

We enhanced the WUIL repository with 53 new user's datasets and their corresponding attacks. In this updated version we gathered information of several users with newer versions of MS Windows OS: 8 and 8.1. This update was produced with the purpose of having a bigger repository of datasets and more representative results. It is worth noting that each WUIL user dataset in the repository is comparable to a standard dataset in the literature, like IRIS, part of the datasets of the UCI Repository [23]. So more than a dataset, WUIL is actually a repository of datasets and under continuous extension.

Table I displays information of the 73 user datasets, including the number of days logged and the size of the user

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Table I
UPDATED WUIL DATASET REPOSITORY.

User	User Log			Total Time			Total Windows			Total Accesses		
	Number of Days	Total Windows	Total Accesses	Attack 1	Attack 2	Attack 3	Attack 1	Attack 2	Attack 3	Attack 1	Attack 2	Attack 3
1	49	3,269	272,642	04:54	04:53	04:56	8	8	10	2,354	4,352	14,825
2	48	1,559	63,536	04:50	04:37	04:50	10	9	4	2,032	3,968	1,273
3	35	606	15,159	04:29	04:35	03:14	8	9	4	2,155	1,069	1,836
4	29	1,715	129,847	04:13	04:32	04:46	8	9	10	3,646	7,422	11,235
5	54	1,734	338,788	04:50	04:39	04:47	10	10	10	3,001	14,926	10,378
6	57	4,189	356,194	04:51	04:42	03:56	9	9	8	3,499	8,530	7,466
7	56	3,309	335,081	04:51	04:53	04:50	10	9	10	3,091	18,625	11,464
8	31	1,213	85,482	04:43	04:45	04:54	10	10	10	5,267	7,898	10,404
9	34	3,086	178,460	04:54	04:43	04:50	10	10	10	3,546	10,711	9,519
10	13	6,597	545,715	04:57	04:51	02:20	10	9	5	6,033	25,162	842
11	44	3,288	197,639	04:58	04:41	04:57	9	8	9	1,865	3,273	8,537
12	34	1,011	19,813	04:58	04:54	04:59	10	9	10	1,073	553	1,716
13	54	9,413	453,204	04:58	04:59	04:53	9	10	10	847	5,779	4,148
14	37	8,382	1,290,295	04:59	04:57	04:58	10	8	10	2,845	911	1,862
15	35	4,467	213,147	04:52	04:56	04:57	10	10	10	1,547	2,282	3,330
16	30	720	10,315	04:59	04:39	04:59	10	7	10	1,610	498	1,312
17	36	2,708	54,302	04:42	03:37	03:59	8	7	3	588	392	139
18	23	1,057	57,608	04:56	04:54	04:53	10	10	10	3,443	52,804	10,361
19	56	4,610	144,797	04:53	04:56	04:53	9	9	10	828	305	27,039
20	38	2,684	167,909	04:41	04:19	04:29	7	8	9	715	1,812	13,902
21	1	32	2,716	04:54	04:55	04:58	9	9	10	1,621	3,242	16,571
22	43	2,613	116,191	04:43	04:55	04:29	9	9	9	1,625	1,101	23,486
23	41	1,018	54,860	04:56	04:49	04:49	10	10	10	3,355	1,906	16,637
24	56	9,606	571,422	04:22	04:49	04:53	8	9	10	733	825	31,486
25	63	7,880	778,116	04:54	04:38	02:55	8	9	6	3,915	6,201	26,780
26	60	5,919	629,253	04:55	04:58	04:56	10	10	10	2,320	3,751	9,608
27	58	4,266	460,432	03:37	03:23	04:30	7	7	10	1,602	2,988	86,561
28	37	6,927	464,522	04:59	04:59	04:59	10	10	10	12,662	3,609	33,675
29	7	176	3,116	04:27	04:25	04:20	9	6	9	2,118	3,823	26,586
30	63	3,226	108,441	04:44	04:44	04:48	8	7	10	744	99	15,128
31	50	1,626	45,539	04:55	04:45	03:59	10	9	8	1,497	354	41,626
32	47	3,766	1,060,389	04:31	04:33	04:52	9	9	10	551	1,541	18,014
33	61	5,349	553,489	04:43	04:51	04:00	9	10	9	3,955	952	7,382
34	62	2,559	371,217	04:48	04:34	04:52	9	9	10	1,579	703	12,183
35	62	8,122	515,309	04:33	04:43	04:58	8	8	10	1,172	870	13,137
36	1	304	13,431	04:46	04:31	04:20	9	7	7	844	219	592
37	53	4,333	664,662	04:58	04:50	03:39	8	3	8	2,262	124	5,343
38	51	4,713	138,512	04:41	04:54	01:55	8	7	4	482	2,108	18,993
39	29	195	3,142	04:57	04:41	04:00	9	8	8	577	151	675
40	54	2,201	119,218	04:49	04:38	04:59	10	9	10	4,587	6,542	62,222
41	63	7,541	316,529	04:54	04:41	04:54	10	9	9	3,987	1,120	70,623
42	50	2,361	188,235	04:51	00:33	04:53	9	0	10	807	2	8,074
43	56	3,328	218,947	04:54	04:50	04:55	9	9	10	1,984	1,520	18,608
44	3	1,018	221,612	04:52	04:34	04:57	10	9	10	1,438	9,034	37,311
45	12	2,964	433,426	04:10	04:53	04:45	9	10	8	3,542	1,383	52,026
46	62	5,523	244,202	04:40	03:57	03:18	7	6	2	328	645	5,727
47	1	21	474	04:35	04:54	04:45	8	9	10	392	3,025	9,209
48	58	3,078	376,373	04:55	04:45	04:56	9	9	10	554	7,527	18,869
49	56	6,115	170,785	04:57	04:43	04:51	7	9	10	1,168	1,684	45,630
50	45	3,602	251,810	04:57	04:53	01:07	9	9	2	1,446	1,101	80
51	18	2,757	417,005	04:59	04:49	04:49	10	10	10	9,046	2,264	28,155
52	17	529	482,944	04:52	04:58	04:46	9	9	10	15,946	819	12,284
53	59	5,034	480,525	04:54	04:58	04:56	9	9	10	4,220	797	32,578
54	58	2,660	148,998	04:59	04:17	03:54	8	7	8	1,012	63	2,557
55	61	3,376	219,761	04:52	04:56	04:58	10	10	9	2,567	1,780	32,736
56	59	3,499	101,282	04:58	04:53	04:29	8	9	9	3,183	4,305	90,799
57	47	5,365	503,581	04:59	04:19	04:58	10	9	10	3,344	3,264	8,207
58	9	847	119,243	04:00	04:52	04:58	8	10	10	2,700	4,532	101,901
59	44	3,671	505,784	02:58	04:49	04:55	6	10	9	1,285	3,490	10,585
60	49	1,781	156,852	04:52	04:58	04:57	10	9	10	4,539	753	5,314
61	48	3,389	119,002	04:47	04:47	04:53	8	8	9	2,152	13,549	10,292
62	51	4,223	246,622	04:52	04:42	04:45	9	7	10	3,103	3,372	12,743
63	49	2,502	132,002	04:55	04:58	04:48	9	9	10	2,047	2,365	22,922
64	58	2,880	120,037	04:48	04:56	04:50	8	9	10	3,716	2,616	65,969
65	61	1,927	39,918	04:51	04:21	04:02	3	9	8	424	1,725	11,064
66	54	3,225	225,680	04:54	04:42	04:42	10	10	10	2,141	509	39,886
67	64	6,170	1,211,044	04:52	04:51	04:39	7	9	10	3,439	4,282	15,172
68	1	70	867	04:44	04:46	04:59	10	7	10	9,120	4,746	2,785
69	49	7,316	225,208	04:56	04:59	04:50	9	9	10	701	726	70,484
70	47	3,476	512,354	04:53	04:44	04:49	10	9	10	1,739	9,093	5,046
71	50	17,863	453,125	04:53	04:43	01:02	9	5	3	905	151	922
72	47	5,606	457,845	04:48	04:50	04:59	9	10	10	3,300	1,262	41,081
73	33	998	44,673	04:56	04:55	04:58	10	10	10	1,008	401	8,442

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

log records, in terms of the number of windows and the total number of accesses. Each window comprises 30 seconds of user activity while the user interacted with her file system. The rest of the columns in the Table show information about the three types of attack: duration, number of 30-second windows, and total number of accesses.

IV. Locality Abstraction

In this paper, we introduce a new abstraction based on the cache-memory term locality. According to [24]: "Program locality refers to the tendency of programs to cluster references to memory, rather than dispersing references uniformly over the entire memory range". This term can be divided in two: temporal locality and spatial locality.

Kumar and Wilkerson [25] define spatial locality as "the property whereby a reference to a memory location indicates that a neighboring location will very likely be referenced in the near future" and temporal locality as "the property whereby a reference to a memory location indicates that the same location will very likely be referenced again in the near future." Following these ideas, we define spatial locality as the property of the user to access files that are close to each other and temporal locality as the property of the user to access the same file in the near future.

Following the locality principle, in this paper we aim to extract spatial- and temporal-locality features to characterize user behavior. Our working hypothesis is that locality-based features should improve on masquerade detection performance, pretty much as the locality principle has helped getting a functional computer system, *cf.* memory caching and memory paging would not work if it did not hold in practice. In what follows, we describe the 16 features that we have extracted from each user dataset in the WUIL repository. They are categorized in three different types: spatial locality features (four), temporal locality features (eight) and direction (four). Spatial locality features attempt to capture how a user commonly traverses her file system structure, while temporal locality features file access frequency and the time among two consecutive accesses to a given file object. By contrast, direction features combine aspects of both spatial locality and temporal locality; they denote the direction (north, south, east, and west) a user is heading at while browsing her file system.

A. Spatial Locality Features

In this section, we introduce our spatial locality features. They are all based on the idea that, while working, a user may access files that are close to each other. Before going anything further, we present some definitions.

We use $e_1, e_2, \ldots$ to denote a sequence of access events. An access event, e, and for short just event from now on, is a tuple including object o wherein the access is performed and a time stamp, t. An object o is itself a tuple including file name, position, and depth.

The position of an object is a list that includes the sequence of nodes to be traversed to reach the object in the file system, starting from the root directory, when structured as a tree. For example, $[2, 1, 2]$ stands for the position of the object named MyFile in the tree depicted in Fig. 1. We respectively use $\mathrm{pos}(e)$ and $\mathrm{file}(e)$ to denote the position and the file name of the object in e. The depth of the object in e, denoted $\mathrm{depth}(e)$, is simply given by the length of its position, *i.e.* length($\mathrm{pos}(e)$).

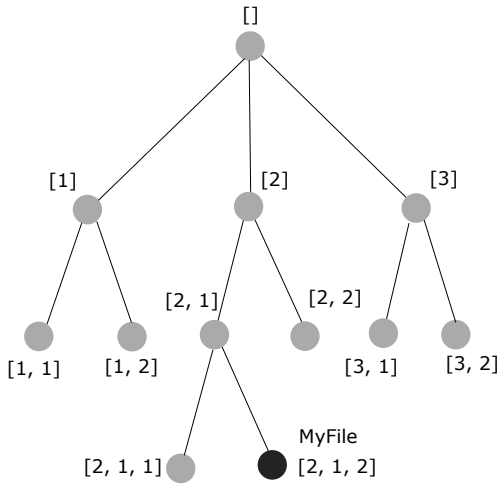

Fig. 1. A file system tree where object MyFile is at position $[2, 1, 2]$.

A window w of size l is a sequence of events, of the form $e_1, e_2, \ldots, e_l$. Windows are all of fixed time without overlapping. Empty or singleton windows are discarded for either MDS construction or evaluation. Notice that windows do not comprise the same number of events.

Two objects are considered siblings if they are under the same node; and this node is considered the parent of all objects, *i.e.* children, below it. We define the distance between two objects by means of the following values: 0 when the two objects are the same; the distance is 1 when the objects are siblings; and also 1 between a parent and one of its children. Next, we define the distance between any two objects as the aggregated distance when moving from one object to the other.

Given a window w, we compute the distance between a pair of events e_i, e_j, computing the distance between their positions: $\mathrm{dis}(\mathrm{pos}(e_i), \mathrm{pos}(e_j))$. Function $\mathrm{dis}(p_i, p_j)$ is defined by Algorithm 1. It simply removes the common prefix of both position lists since it denotes the common path before they start to diverge in the file system tree. Thus, the distance between the two objects is just the sum of the length of the two independent branches. Notice that in the case that the objects are in different branches, a shortcut is taken just before reaching their common parent. At that level, the route taken by the algorithm is directly between siblings, instead of visiting the parent to descend again. Hence, a value of 1 is held back in the computed distance.

We use $D(w)$ to denote the indexed list $d_1, d_2, \ldots, d_{l-1}$, such that d_i is the distance between two adjacent events e_i and e_{i+1} in w, and that $d_i = \mathrm{dis}(\mathrm{pos}(e_i), \mathrm{pos}(e_{i+1}))$.

We now define four spatial locality features: window-distance, average inter-event distance, diameter, and short inter-event distance rate, especially designed to put apart masquerade behavior.

Window distance amounts to the total distance travelled

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Algorithm 1 Function to obtain the distance between two objects

```
1: function DIS(listA, listB):
2:     prefix = Common_prefix(listA, listB)      ▷ Returns the common prefix list of listA and listB
3:     Remove (prefix, listA, listB)             ▷ Removes the common prefix list from listA and listB
4:     distance = length(listA) + length(listB)
5:     if (length(listA) ≠ 0 and length(listB) ≠ 0) then:   ▷ Objects are in different branches, in which case
6:         distance = distance - 1               ▷ the path is through the sibling instead of the parent node
7:     end if
8:     return distance
9: end function
```

in a window. It is expected to be short for a legitimate user, where spatial locality should hold, and long otherwise. Window-distance, $\mathrm{wd}(w)$, is given by the sum of all inter-event distances in a window; in symbols:

$$\mathrm{wd}(w) = \sum_{i=1}^{l-1} d_i$$

Being an overall measure, window distance does not give us any clue as to how far the files of two consecutive access are, or whether a window contains a number of accesses to files close one another. The three following features attempt to fill in this gap.

The *average inter-event distance* for a window w, $\mathrm{ied}_{\mathrm{avg}}(w)$, is simply given by:

$$\mathrm{ied}_{\mathrm{avg}}(w) = \frac{\mathrm{wd}(w)}{l-1}$$

Complementing this measure, we define the *diameter* of a window w, $\mathrm{diam}(w)$, as the maximum distance between any pair of events in w:

$$\mathrm{diam}(w) = \max(\{\mathrm{dis}(\mathrm{pos}(e_i), \mathrm{pos}(e_j)) \mid i, j \in [1, l]\})$$

Again, we expect these last two distance measures to be short for a legitimate user; intruders, however, for which presumably spatial locality does not hold, shall span over the file system, yielding rather large file leaps.

Finally, to capture that spatial locality implies using files next to each other, we compute the rate at which two consecutive accesses in w correspond to files with a zero- or one-inter event distance. We call this measure *short inter-event distance rate*, and define it as follows:

$$\mathrm{short\_ied}(w) = \frac{|\{d_i \in D(w) \mid d_i \in \{0,1\}, i \in [1, l-1]\}|}{l-1}$$

The value of this feature is in the range $[0, 1]$; it is expected to be close to 1 for users working on specific tasks, thus visiting objects close one another. An intruder, by contrast, may perform hops between far separated objects, while moving around looking for files of interest. In this case, this feature value will be close to 0.

B. Temporal Locality Features

In this section we introduce our eight temporal locality features. They are all based on the idea that, while working, a user will frequently access the same files within a short period of time. The first four features stem from file access frequency, while the latter four ones from the elapsed time between two consecutive accessed to a given file. The first set of features is: file diversity rate, highest file access rate, average file access frequency, and single-access files. The second set is: temporal aggregate, temporal aggregate average, maximum elapsed time, and first time access rate.

1) File Access Frequency Features: Given that some files appear several times in a window, we first define $\mathrm{Ff}(w)$, which returns a list whose elements are tuples, each of the form (f, c); f is the file name of the object in e, and c is the number of times that f appears in an event in window w, given by $\mathrm{count}(f, w)$. In symbols:

$$\mathrm{Ff}(w) = [(\mathrm{file}(e), \mathrm{count}(\mathrm{file}(e), w)) \mid e \in w]$$

where $[\ell \mid \phi]$ is the list formed by collecting every element ℓ such that ϕ holds.

The *file diversity rate* of a window w, $\mathrm{fdr}(w)$, amounts to the number of distinct files accessed in w divided by the total number of file accesses in w; in symbols:

$$\mathrm{fdr}(w) = \frac{\mathrm{length}(\mathrm{Ff}(w))}{l}$$

$\mathrm{fdr}(w)$ is in the $(0, 1]$ interval. We expect an intruder to have a large fdr value, close to 1, giving evidence of a widespread navigation in the file system with almost no repetitions; by contrast, a legitimate user will have a value close to 0, *i.e.* low diversity in the number of accessed files and, thus, compliant with temporal locality.

To complement fdr, we further include three features. The first feature corresponds to the access rate of the most frequently accessed file in a given window, w. We call it *highest file access rate*, $\mathrm{far}_{\max}(w)$, and define it as follows:

$$\mathrm{far}_{\max}(w) = \frac{\max(\{c \mid (f, c) \in \mathrm{Ff}(w)\})}{l}$$

The range of this feature is $(0, 1]$. However, contrary to fdr, a value of 1 exemplifies a user that is focused on a specific task, accessing a small set of files within a short period of time; whereas a value close to 0, denotes possibly a masquerader which avoids re-accessing the same file again.

The other feature is simply the normalized *average file access frequency*; it amounts to the average of the number of times each file in a window is accessed, and it is given by:

$$\mathrm{faf}_{\mathrm{avg}}(w) = \frac{\sum_{i=1}^{\mathrm{length}(\mathrm{Ff}(w))} [c_i \mid (f_i, c_i) \in \mathrm{Ff}(w)]}{\mathrm{length}(\mathrm{Ff}(w))}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

This feature takes values ranging over $[1, \mathrm{far}_{\max}(w) \times l]$. A value of 1 characterizes a user working on a dispersed set of files, where temporal locality does not hold. Complementary, a value close to the upper limit, represents a user working on a specific task, thus interacting with a reduced set of files.

Finally, our last feature regarding file access frequency is the number of files that, in a given window, are accessed only once. *Single-access files* is defined by:

$$\mathrm{single\_af}(w) = \mathrm{length}([c_i = 1 \mid (f_i, c_i) \in \mathrm{Ff}(w)])$$

The range of this value is $[0, l]$. In this case, we expect an intruder to score close to l in this feature, because it is working on a large set of files. By way of comparison, a user compliant to temporal locality will score close to 0, since she makes repeated use of the same files within a short period of time.

2) Elapsed Time Features: A key temporal locality property is the time between accesses to the same object. Take a window w, such that $e_i \in w$. Then, let $t_1(e_i)$ be the access time to object i in e_i, and $t_0(e_i)$ be the time of the prior access to the same object. The elapsed time between the last two consecutive accesses to the same object is defined as follows:

$$\Delta t(e_i) = \begin{cases} t_1(e_i) - t_0(e_i) & \text{if } t_0(e_i) \text{ is defined} \\ 0 & \text{otherwise} \end{cases}$$

Notice that $t_0(e_i)$ is undefined whenever $t_1(e_i)$ is the first access on o_i since system initialization. Notice also that $t_0(e_i)$ can refer to an event occurred outside the current window.

We define the *temporal aggregate* of a window as the sum of all $\Delta t(e_i)$ in a window, as follows:

$$\mathrm{ta}(w) = \sum_{i=1}^{l} \Delta t(e_i), \qquad e_i \in w$$

And the *temporal aggregate average* by:

$$\mathrm{ta}_{\mathrm{avg}}(w) = \frac{\mathrm{ta}(w)}{l}$$

These features both indicate whether or not objects are being referenced to repeatedly within a short period of time. A small value denotes a user exhibiting temporal locality, whereby a reference to an object anticipates that the same object will very likely be referenced again in the near future; the opposite identifies the presence of a masquerader.

Now, the *maximum elapsed time* in a window can be readily obtained as follows:

$$\mathrm{et}_{\max}(w) = \max(\{\Delta t(e_i) \mid e_i \in w\})$$

This feature needs to be interpreted with caution, since a legal user may score high if she moves to a file system portion totally apart from she was recently working on. A masquerader might also score high but as a consequence of malicious activity. Nevertheless, a low score indeed suggests legitimate user activity.

We are also interested in the number of occurrences where $\Delta t(e_i) = 0$ in a given window. To normalize this value, we divide it by the total number of accesses in that window. This measure, we call *first time access rate*, is computed as follows:

$$\mathrm{ftar}(w) = \frac{\mathrm{length}([\Delta t(e_i) = 0 \mid e_i \in w])}{l}$$

A value close to 1 suggests there is an intruder navigating through objects that had not been referenced to in a while, indeed beyond the computer system was started. A value close to zero is taken as normal user behavior.

C. Direction Features

We now introduce a collection of features, we call *direction*, which attempt at capturing where a user is heading at while interacting with her file system. There are four different directions, namely: north, south, east, and west. From one object to the other, north indicates that the user has moved up to one of her predecessors; south that she has moved down to one of her descendants; and complementary for the other cases. Each feature is represented by means of a variable, which is incremented whenever the user 'travels' toward the corresponding direction. Directionals such as northeast, northwest, and so on, straightforwardly correspond to a simultaneous update of two counter variables.

Let $\mathrm{direction}(e_i, e_j)$ be the function that returns $\langle 1, 0, 0, 0\rangle$ if moving from the object in e_i to that of e_j implies heading north, $\langle 0, 1, 0, 0\rangle$ if so implies moving south, etc., as expected from our discussion above. Further, let $w = e_1, \ldots, e_l$ be a window. Then the overall user direction for w, $\mathrm{heading\_at}(w)$, is defined as follows:

$$\mathrm{heading\_at}(w) = \sum_{i=1}^{l-1} \mathrm{direction}(e_i, e_{i+1}), \qquad e_i \in w$$

We expect an ordinary user to browse over her entire file system following a common direction and that the masquerader will have a strange direction pattern.

With this we have completed the features used in our masquerade detection system model. Attention is now given to model construction, model evaluation, and analysis of results.

V. Experiments and Results

To validate our working hypothesis, that locality features better separate a user from a masquerade, we shall first compare the performance of two classifiers, one built with locality features, and the other with the combination of features suggested in [13], here called WUIL features. In §V-C, we will see that the locality-based classifier outperforms that of [13]. Later, in §V-E, we shall compare our locality-based classifier against two of the methods proposed by Gates *et al.* Before going any further, though, let us discuss how data has been pre-processed.

A. Dataset Pre-processing

First, we have divided the WUIL log into time-based windows. A window is set to start when an access is performed and covers all accesses realized within the next 30 seconds. If a window contains only one access, the window is discarded. For windows with more than one access, we computed all locality features, temporal, spatial and direction, just as presented in §IV. Also, we computed the 25 features, called the WUIL features, introduced in [13]. Next for each user, let say u_i, we obtain the following logs:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

- *Locality User Log ($\mathcal{L}$:UL)*. Contains locality features from all access windows performed by user u_i during normal task activity.
- *Locality Log from Attack α ($\mathcal{L}$:ALα)*. Contains locality features computed from windows generated during attack α on user u_i, $\alpha \in [1, 2, 3]$.
- *Locality Log from All Attacks ($\mathcal{L}$:AAL)*. Contains locality features computed from windows produced during attack 1 through 3 on all users except u_i.
- *WUIL User Log ($\mathcal{W}$:UL)*. Contains WUIL features from all access windows performed by user u_i during normal task activity.
- *WUIL Log from Attack α ($\mathcal{W}$:ALα)*. Contains WUIL features computed from windows produced during attack α on user u_i, $\alpha \in [1, 2, 3]$.
- *WUIL Log from All Attacks ($\mathcal{W}$:AAL)*. Contains WUIL features computed from windows generated during attack 1 through 3 on all users except u_i.

During the next step of pre-processing, we removed and discarded for the rest of the experiments, the first 5% of windows in $\mathcal{L}$:UL and $\mathcal{W}$:UL, since there are some features that require a minimum of historical data to become steady (*e.g.* $\Delta t(e_i)$, and all features that depend upon it, see §IV-B); the remaining log was used for experimentation. From now on, and for the sake of simplicity, we will refer to $\mathcal{L}$:UL and $\mathcal{W}$:UL just as UL; to $\mathcal{L}$:AAL and $\mathcal{W}$:AAL as AAL; and to $\mathcal{L}$:ALα and $\mathcal{W}$:ALα as ALα. However, it is important to keep in mind that a process performed, for example on UL, translates into a dual and equivalent process, one over $\mathcal{L}$:UL and the other over $\mathcal{W}$:UL.

B. Experimental Settings

We applied in our experiment a Five-Fold Cross-Validation (FFCV) process: we randomly partitioned UL into five equal sized sub-samples. We used the first four sub-samples as part of the training set and the fifth sub-sample for validation purposes. We repeated the cross-validation process five times, so each of the five sub-samples is used exactly once as the validation set.

A natural problem when working with WUIL, which is common among different datasets, is the imbalance problem of the classes. Normally, there are many user events compared to a small amount of masquerade ones. In order to face this problem, we proceeded to re-balance the classes by selecting at random with replacement, windows from AAL until we reached the size of the four sub-samples used for training during the FFCV process. We call this set the balanced AAL set, *bAAL*. After this re-balance procedure, we obtained three different sets described below:

- *Training Set (TS)*: The four sub-samples from UL used for training during the FFCV process together with the bAAL set.
- *User Validation Set (UVS)*: The fifth sub-sample from UL used for validation during the FFCV process. From this validation set we obtain the False Positives (FP), whenever the user is erroneously classified as a masquerader, and the True Negatives (TN), whenever the user is correctly classified as the legitimate user.
- *Masquerader Validation Set (MVS)*: Made up from ALα, $\alpha \in [1, 2, 3]$, altogether. From this validation set we obtain the False Negatives (FN), when the masquerader is incorrectly classified as the user, and True Positives (TP), whenever the masquerader is properly classified as a masquerader.

Next, we performed a normalization and scaling process on the data within the three sets just described. To normalize data, we used the training set to find the media $\overline{X}$ and the standard deviation s of every feature. Then, the student's t-statistic, as recalled by equation (1), was applied to every feature X in the training, user validation and masquerader validation sets. Afterward, a scaling function was performed by finding the minimum and maximum values, X_{min} and X_{max}, for every feature in the training set; then, equation (2) was applied to every feature X from the three sets.

$$\frac{X - \overline{X}}{s} \tag{1}$$

$$\frac{X - X_{min}}{X_{max} - X_{min}} \tag{2}$$

Putting all pieces together, we summarize the FFCV process: at each step we used to train the classifier the training set (TS) produced during the fold, and for validation purposes the user validation set (UVS) obtained during the same fold together with the masquerade validation set (MVS). At the end of the fifth fold, all results were averaged.

C. Results using the TreeBagger Classifier

For our experiments, we used the MATLAB classifier TreeBagger [26], which is a bootstrap aggregating algorithm that generates n random trees using a subset of the training set. The subset is obtained by sampling with replacement. In our case, we selected $n = 100$. Each tree has just about 63.2% of information from the training set, while the rest is fulfilled with repeated samples. For window classification, TreeBagger returns the most voted class among the 100 trees. Notice how in our results detection performance is reported on a per window basis.

To evaluate our results we used five metrics: Area Under the Curve (AUC), Minimum Misclassification Point (MMP), Equal Error Rate (EER), Zero-FN, and Zero-FP. The first measure, AUC, stands for the Area Under the Curve of the Receiver Operating Characteristic (ROC) curve. The remaining ones, MMP, EER, Zero-FN and Zero-FP, all denote a point in the ROC curve that minimizes a particular criterion: for MMP the criterion is $\mathrm{FP} + \mathrm{FN}$, for EER it is $|\mathrm{FP} - \mathrm{FN}|$, for Zero-FP it is FN while FP = 0, and, finally, for Zero-FN the criterion is FP while FN = 0. Complete results are shown in Table II, there, we respectively use ZFN and ZFP as an acronym for Zero-FN and Zero-FP. A number in bold means that the corresponding classifier performs better than its counterpart does, for the same metric and the same user. An underlined number means equal performance.

Analyzing the results in Table II we can observe that, in average, all metrics obtained from locality features are better than their counterpart, derived from WUIL features. It is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

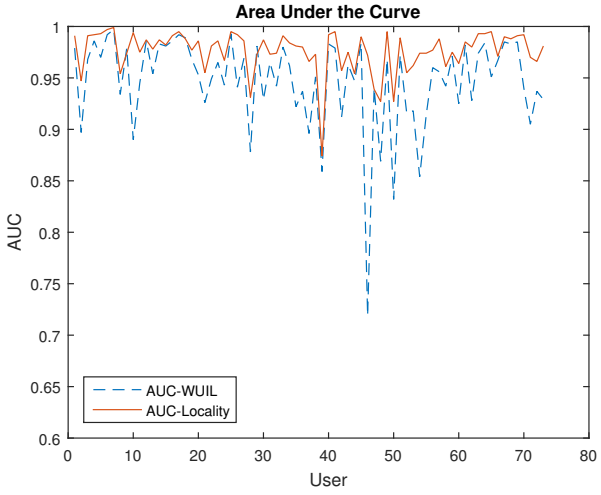

Fig. 2. Comparing WUIL versus locality features: AUC metric.

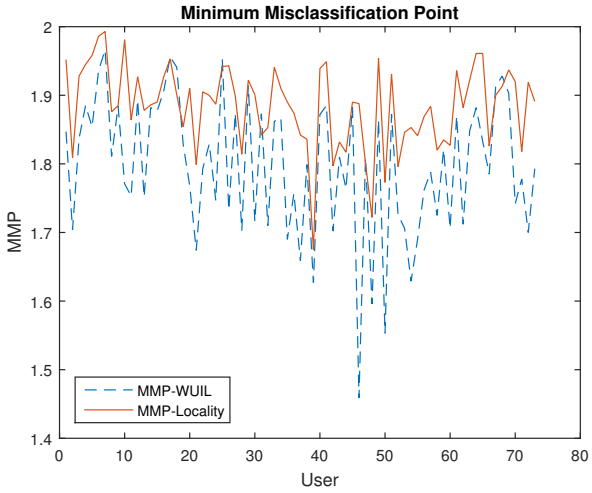

Fig. 3. Comparing WUIL versus locality features: MMP metric.

worth noticing that, for the WUIL features-based classifier, the results reported in this paper surpass the original ones reported elsewhere [13]. So the locality-based classifier outperforms the WUIL-based one even in optimistic comparison. It is important to highlight that in general terms both classifiers perform specially well; for instance, the average AUC is 0.97 and 0.94, in each case. Moreover, 30% of users (22 users) show an AUC value equal or above 0.99, when derived from locality features.

Now we move into a deeper inspection of metrics derived from locality features. In the case of MMP, in average we obtained a value of 188, which translates into a minimal misclassification rate of approximately 6% of false positives and about the same percentage of false negatives. 42% of users (31 users) scored above 190, which lowers FP and FN to about 5%. The results are similar in the case of EER; in average, 7% of false positives and false negatives. In addition, 46% of users attained a EER value below 6%. In the case of Zero-FN, we can detect all attacks but with 29% of false positives. Nevertheless, in the case of 28% of users, this value is lower than 10%. Unfortunately, Zero-FP is not an encouraging result: we can classify correctly a legitimate user in all cases but at the expense of misclassifying a masquerader as a legal user 88% of the instances; only 9 users scored less than 65%.

We can clearly appreciate in Figs. 2, 3, and 4 that the classifier based on locality features supersedes the classifier based on WUIL features. AUC, MMP, and EER are the metrics compared in each of the graphs. Even though this is also true in average for the case of Zero-FN and Zero-FP, we preferred to show their dispersion graph in Figs. 5 and 6 since the standard deviation is significant.

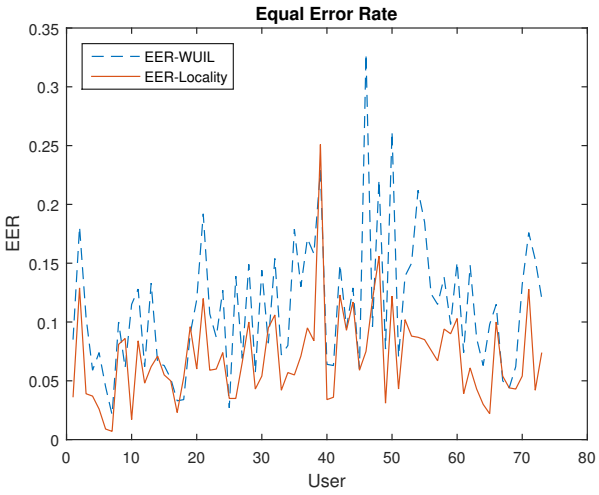

Fig. 4. Comparing WUIL versus locality features: EER metric.

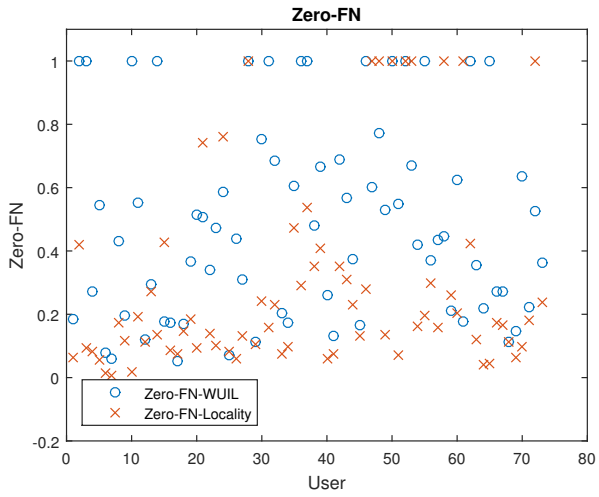

Fig. 5. Comparing WUIL versus locality features: Zero-FN metric.

D. Feature Selection

In order to determine if one or more of our proposed features do not contribute to classification, we tested three algorithms for feature selection: Sequential Forward Selection (SFS), Sequential Backward Selection (SBS), and Random Feature Selection (RFS) (the reader is referred to [27], which includes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Table II
RESULTS OF TREEBAGGER CLASSIFIERS APPLIED TO WUIL AND LOCALITY FEATURES.

User	Locality features					WUIL features				
	AUC	MMP	EER (%)	ZFN (%)	ZFP (%)	AUC	MMP	EER (%)	ZFN (%)	ZFP (%)
1	**0.991**	**195.22**	**3.64**	**6.36**	100.00	0.979	184.70	8.45	18.63	100.00
2	**0.947**	**180.91**	**12.90**	**42.14**	**87.83**	0.897	170.35	18.14	100.00	93.91
3	**0.991**	**192.82**	**3.88**	**9.40**	**53.33**	0.968	183.66	10.43	100.00	98.10
4	**0.992**	**194.50**	**3.72**	**8.10**	**59.26**	0.986	188.51	5.94	27.23	100.00
5	**0.993**	**195.84**	**2.57**	**5.53**	100.00	0.970	185.37	7.44	54.50	**92.67**
6	**0.997**	**198.62**	**0.90**	**1.38**	100.00	0.992	193.60	4.53	7.66	**77.69**
7	**0.999**	**199.27**	**0.68**	**0.73**	**46.21**	0.997	196.62	2.11	5.95	100.00
8	**0.954**	**187.62**	**8.12**	**17.54**	100.00	0.934	181.14	10.02	43.02	100.00
9	0.974	**188.51**	8.56	**11.49**	100.00	**0.979**	188.39	**6.06**	19.57	100.00
10	**0.994**	**198.06**	**1.74**	**1.94**	100.00	0.890	177.09	11.49	100.00	100.00
11	**0.975**	**186.41**	**8.42**	**19.34**	100.00	0.945	175.27	12.75	55.13	100.00
12	**0.987**	**192.67**	**4.84**	**11.19**	100.00	0.986	189.32	6.22	12.08	100.00
13	**0.978**	**187.78**	**6.20**	**27.15**	100.00	0.954	175.18	13.26	29.60	100.00
14	**0.987**	**188.65**	7.13	**13.37**	**77.14**	0.983	188.19	**6.47**	100.00	77.86
15	**0.982**	**188.98**	**5.51**	42.86	100.00	0.981	187.71	6.35	**17.77**	100.00
16	**0.991**	**192.73**	**4.95**	**8.47**	79.26	0.986	190.60	5.23	17.50	79.26
17	**0.995**	195.34	**2.33**	7.27	**85.56**	0.992	**195.57**	3.27	**5.06**	87.78
18	0.988	190.14	5.26	**14.65**	100.00	**0.989**	**194.13**	**3.41**	17.14	100.00
19	**0.977**	**185.41**	9.64	**18.42**	**71.43**	0.968	182.79	**8.98**	36.83	100.00
20	**0.986**	**191.05**	**6.01**	**9.35**	**74.17**	0.953	176.75	11.91	51.34	100.00
21	**0.955**	**179.88**	**11.96**	74.17	**27.86**	0.926	167.38	19.23	**50.83**	36.43
22	**0.981**	**190.47**	**5.91**	**14.00**	100.00	0.949	179.34	10.71	33.86	100.00
23	**0.986**	**189.97**	**6.05**	**10.03**	100.00	0.965	182.93	8.66	47.42	100.00
24	**0.967**	**188.68**	**7.41**	75.85	100.00	0.943	174.67	12.71	**58.63**	**99.26**
25	**0.995**	194.18	3.46	8.30	100.00	0.993	**195.29**	**2.73**	**7.23**	100.00
26	**0.992**	**194.26**	**3.51**	**5.81**	100.00	0.941	173.27	13.94	43.86	100.00
27	**0.986**	**189.87**	**6.63**	**13.08**	100.00	0.969	187.36	6.90	31.08	100.00
28	**0.931**	**181.43**	**9.98**	100.00	**78.00**	0.878	170.29	14.88	100.00	100.00
29	0.973	**192.15**	**4.31**	**10.63**	100.00	**0.981**	191.39	5.69	11.11	100.00
30	**0.987**	**190.08**	**5.36**	**24.15**	100.00	0.929	171.61	14.41	75.17	100.00
31	**0.973**	184.23	9.53	**15.95**	**74.81**	0.965	**187.30**	**8.18**	100.00	100.00
32	**0.974**	**185.31**	**10.59**	**23.19**	100.00	0.942	170.97	15.40	68.41	**64.29**
33	**0.991**	**194.12**	**4.16**	**7.63**	100.00	0.980	186.19	7.23	20.23	100.00
34	**0.984**	**191.03**	**5.67**	**9.57**	100.00	0.962	186.72	7.97	17.39	100.00
35	**0.981**	**189.00**	**5.50**	**47.32**	100.00	0.922	169.02	17.92	60.58	100.00
36	**0.980**	**187.42**	**7.12**	**29.06**	**51.30**	0.937	175.71	12.98	100.00	100.00
37	**0.966**	**184.21**	**9.55**	**53.80**	100.00	0.896	165.85	17.07	100.00	**96.84**
38	**0.973**	**183.55**	**8.42**	**35.33**	100.00	0.951	179.90	15.77	48.19	**70.53**
39	**0.874**	**167.51**	25.14	**41.03**	92.00	0.859	162.67	**22.88**	66.67	**91.20**
40	**0.992**	**193.92**	**3.38**	**6.09**	**73.79**	0.983	187.16	6.42	26.12	100.00
41	**0.995**	**194.93**	**3.58**	**7.47**	**82.86**	0.979	188.46	6.34	13.22	100.00
42	**0.957**	**179.75**	**12.35**	**35.07**	**68.42**	0.912	170.23	14.88	68.84	100.00
43	**0.975**	**183.21**	**9.27**	**31.07**	100.00	0.963	180.95	9.53	56.79	**67.86**
44	**0.954**	**181.71**	**11.66**	**23.00**	83.45	0.947	176.40	12.94	37.63	**66.21**
45	**0.990**	**189.01**	**5.93**	**13.02**	**62.96**	0.984	188.31	5.98	16.50	100.00
46	**0.972**	**188.83**	**7.51**	**28.03**	100.00	0.719	145.90	32.84	100.00	100.00
47	0.939	180.15	11.93	100.00	51.11	**0.940**	**180.89**	**9.56**	**60.00**	**44.44**
48	**0.927**	**172.17**	**15.62**	100.00	**69.29**	0.869	159.64	22.15	**77.32**	79.29
49	**0.995**	**195.37**	**3.09**	**13.65**	100.00	0.968	186.39	7.72	53.08	100.00
50	**0.927**	**177.31**	**12.24**	100.00	100.00	0.832	155.28	26.18	100.00	100.00
51	**0.989**	**193.11**	**4.28**	**7.10**	100.00	0.972	187.19	6.96	54.80	**84.00**
52	**0.955**	**179.57**	**10.22**	100.00	100.00	0.916	172.56	13.89	100.00	**82.86**
53	**0.962**	**184.59**	**8.78**	100.00	100.00	0.918	170.64	14.97	**67.01**	100.00
54	**0.974**	**185.26**	**8.73**	**16.13**	100.00	0.854	162.89	21.17	42.07	100.00
55	**0.974**	**184.11**	**8.48**	**19.64**	100.00	0.914	169.01	18.50	100.00	100.00
56	**0.977**	**186.90**	**7.63**	**29.78**	100.00	0.960	176.24	12.37	37.06	**89.23**
57	**0.988**	**188.41**	**6.66**	**15.82**	100.00	0.956	178.83	11.45	43.56	100.00
58	**0.961**	**181.98**	**9.42**	100.00	100.00	0.942	172.50	13.75	**44.71**	**75.71**
59	**0.975**	**183.53**	**8.98**	26.24	80.80	0.971	182.14	9.79	**21.00**	**70.40**
60	**0.964**	**182.73**	**10.34**	**20.21**	**93.10**	0.925	170.63	15.08	62.32	96.55
61	**0.985**	**193.56**	**3.86**	100.00	100.00	0.982	187.00	7.36	**17.59**	100.00
62	**0.980**	**188.16**	**6.14**	**42.22**	100.00	0.928	171.21	14.76	100.00	100.00
63	**0.993**	**192.20**	**4.34**	**12.04**	**47.86**	0.974	184.76	8.56	35.43	78.57
64	**0.993**	**196.08**	**3.01**	**3.92**	100.00	0.984	188.19	6.35	22.05	**72.59**
65	**0.995**	**196.09**	**2.25**	**4.36**	100.00	0.951	183.30	9.80	100.00	100.00
66	**0.971**	**182.60**	**9.96**	**17.40**	85.33	0.967	178.53	11.50	27.22	**78.67**
67	**0.990**	189.97	5.37	**16.43**	100.00	0.985	**191.26**	**5.01**	27.21	**79.23**
68	**0.988**	191.27	4.37	11.43	**32.59**	0.983	**192.75**	4.37	11.43	100.00
69	**0.991**	**193.73**	**4.29**	**6.27**	100.00	0.985	190.25	6.22	14.64	100.00
70	**0.992**	**192.01**	**5.42**	**9.83**	**75.17**	0.940	174.21	13.17	63.71	91.72
71	**0.970**	**181.81**	**12.83**	**18.19**	100.00	0.905	177.77	17.56	22.23	100.00
72	**0.966**	**191.90**	**4.17**	100.00	100.00	0.937	170.02	15.26	**52.70**	100.00
73	**0.981**	**189.09**	**7.37**	**23.63**	**72.67**	0.929	179.26	11.95	36.35	94.00
AVG	**0.976**	**188.40**	**7.00**	**29.23**	**88.19**	0.946	179.64	11.26	48.94	92.02

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

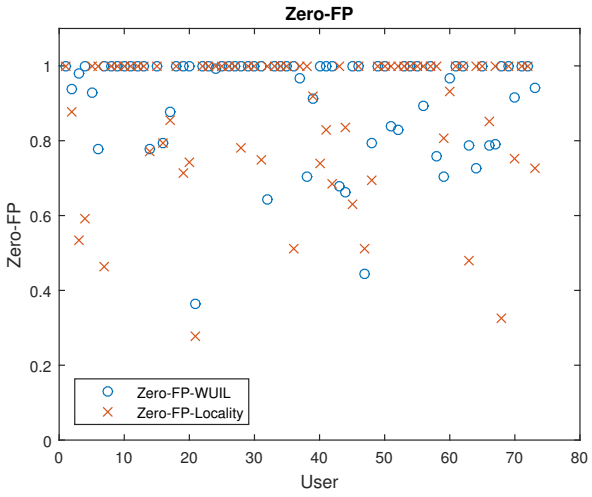

Fig. 6. Comparing WUIL versus locality features: Zero-FP metric.

a recent review about the subject). The rationale behind the selection of these three algorithms is that they all aim at improving classification accuracy. What is more, even though its simplicity, RFS has been shown to actually improve on classification accuracy [29], just as Random Forest does [28]. The three feature selection algorithms use TreeBagger as the base classifier, *i.e.* the feature selection takes place while constructing TreeBagger.

Table III summarizes the results we have obtained from conducting a comparative evaluation of all these methods. It contains in turn several tables, one for each performance indicator. For each inner table, cell (i, j) contains a tuple $x/y/z$ respectively meaning that method i has beaten x times method j, method i has lost y times against method j, and that method i and method j have tied z times.

From this table, we notice that SFS and SBS do not surpass TreeBagger. This implies that, while there might be features less influential for classification performance, getting rid of them severely affects the overall classification. By contrast, RFS surpasses TreeBagger: 3 out of 5 performance indicators show a significant statistical improvement. Moreover, for every performance indicator, RFS consistently outperforms SBS and SFS. This leads us to the conclusion that such less influential features do matter but in a distinct order for any other user. This will be argued below.

To explain these results, let us analyze frequency of feature use, called *feature usage* for short, and given by the ratio of the number of times a feature is used with respect to the number of times features are all used. A feature with high usage is more relevant for classification than one with not. Table IV displays feature usage, as output by each algorithm. In terms of usage, a top ranked feature appears at the top of the table, while a bottom one at the opposite side of it.

Table IV reveals the following:

1) We have identified that the most common top ranked features are to do with temporal locality, namely $\mathrm{et}_{\max}$ (maximum elapsed time), $\mathrm{ta}_{\mathrm{avg}}$ (temporal aggregate average), and ta (temporal aggregate). They involve three measures about the time between the last two consecutive accesses to the same file object, namely: maximum, sum and average. Notice how these measures span over several windows, since a file could have been referenced while ago or even not at all. We believe that this good ranking has to do with how well these features match the temporal locality property, whereby a reference to a file object anticipates that the same file object will very likely be referenced again shortly, in the case of a legitimate user. The power of these features resides in the fact that an intruder would very easily trigger these attributes by touching file objects that have not recently been accessed by a legitimate user.
   We have also identified that the most common bottom ranked features are to do with frequency of file use with respect to a window, namely $\mathrm{faf}_{\mathrm{avg}}$ (average file access frequency) and $\mathrm{single\_af}$ (single-access files). These measures are the average of the number of times each file in a window is accessed, and the number of files with access count equal to one, respectively. One possible reason for these attributes to score so low, might be that they do not span over several windows, and an intruder action can masquerade a legitimate user, since all happens inside a single window. However, such less influential features do matter but in a distinct order for any other user.
2) TreeBagger does not make uniform use of all features. This is because each individual decision tree tends to use more frequently those features that are good to discriminate objects of different classes, consequently overlooking those that do not. If there were any irrelevant feature, TreeBagger would have got rid of it.
3) Random Feature Selection, which achieved the best clas-

Table III
COMPARATIVE EVALUATION OF THE CLASSIFIER IN THE ROW VERSUS THE ONE IN THE COLUMN. CELL VALUES ARE OF THE FORM (WIN/LOSE/TIE). BOLD TYPEFACE IS USED TO STRESS SIGNIFICANTLY STATISTICALLY PERFORMANCE DIFFERENCE ACCORDING TO THE WILCOXON SIGNED-RANK TEST.

Performance indicators	Algorithms	TB	SFS+TB	SBS+TB	RFS+TB
AUC	TB		**67/6/0**	**58/14/1**	36/37/0
	SFS+TB	**6/67/0**		**24/49/0**	**8/65/0**
	SBS+TB	**14/58/1**	**49/24/0**		**12/61/0**
	RFS+TB	37/36/0	**65/8/0**	**61/12/0**	
MMP	TB		**73/0/0**	**73/0/0**	**70/3/0**
	SFS+TB	**0/73/0**		**28/45/0**	**12/61/0**
	SBS+TB	**0/73/0**	**45/28/0**		**12/61/0**
	RFS+TB	**3/70/0**	**61/12/0**	**61/12/0**	
EER	TB		**49/22/2**	39/34/0	**20/53/0**
	SFS+TB	**22/49/2**		**20/52/1**	**14/59/0**
	SBS+TB	34/39/0	**52/20/1**		**12/61/0**
	RFS+TB	**53/20/0**	**59/14/0**	**61/12/0**	
Zero-FN	TB		**47/25/1**	39/33/1	**24/49/0**
	SFS+TB	**25/47/1**		31/41/1	**18/55/0**
	SBS+TB	33/39/1	41/31/1		**24/49/0**
	RFS+TB	**49/24/0**	**55/18/0**	**49/24/0**	
Zero-FP	TB		**14/59/0**	**8/64/1**	**0/73/0**
	SFS+TB	**59/14/0**		36/37/0	**9/64/0**
	SBS+TB	**64/8/1**	37/36/0		**8/65/0**
	RFS+TB	**73/0/0**	**64/9/0**	**65/8/0**	

TB stands for TreeBagger, SFS+TB stands for Sequential Forward Selection+TB, SBS+TB stands for Sequential Backward Selection+TB, and RFS+TB stands for Random Feature Selection+TB.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Table IV
FEATURE USAGE PER ALGORITHM.

TreeBagger		Random Feature Selection + TreeBagger		Sequential Forward Selection+TreeBagger		Sequential Backward Selection+TreeBagger	
Feature	**Usage (%)**	**Feature**	**Usage (%)**	**Feature**	**Usage (%)**	**Feature**	**Usage (%)**
et_{max}	16.46	et_{max}	11.95	ta_{avg}	30.73	ta_{avg}	19.32
wd	10.24	ta	9.23	et_{max}	20.25	et_{max}	17.05
ta_{avg}	9.52	ta_{avg}	9.07	ied_{avg}	12.66	east	16.11
far_{max}	8.35	far_{max}	7.04	wd	11.15	west	11.97
ied_{avg}	7.93	wd	6.97	far_{max}	6.36	ftar	10.78
ta	7.89	east	6.51	fdr	3.95	south	9.34
diam	6.36	ftar	6.35	ta	3.78	ta	5.25
east	5.96	diam	6.15	short_ied	2.25	north	5.05
ftar	5.34	ied_{avg}	6.12	faf_{avg}	2.15	single_af	1.71
fdr	4.94	west	5.02	ftar	1.82	faf_{avg}	1.28
short_ied	4.64	short_ied	4.54	diam	1.58	far_{max}	0.70
single_af	3.69	south	4.40	east	1.05	ied_{avg}	0.43
west	3.30	fdr	4.34	south	0.97	diam	0.34
south	2.45	faf_{avg}	4.34	single_af	0.93	fdr	0.34
north	2.16	north	4.16	west	0.31	short_ied	0.23
faf_{avg}	0.79	single_af	3.80	north	0.07	wd	0.11
Standard deviation	3.73	**Standard deviation**	2.18	**Standard deviation**	8.30	**Standard deviation**	6.63

sification performance (see Table III), makes a more uniform use of the features, *i.e.* it yields the lowest usage standard deviation (see last row of Table IV).

4) Sequential Forward Selection and Sequential Backward Selection highly discriminate features but at the cost of a significant statistical reduction in classification performance with respect to the rest of the algorithms.

For the classification problem at hand, our conclusions are threefold. First, our proposed features are all relevant. Second, bottom ranked features do not uniformly contribute to masquerade detection performance: they are influential for some users, but not for all. And third, the classification consistently improves when the classifier makes uniform use of our features.

Attention is now turned into comparing our locality-based MDS against rival methods from others.

E. Comparison

For comparison purposes, we have implemented and tested SumScore and AveScore [11]. We did not consider NewUnique, as it is essentially a temporal feature, or Access Similarity, as it is not applicable in our single-user context. Given that [10] does not provide an algorithm for popular file path identification, we could not replicate FPD (see §II-D1). As in all our previous experiments, we used a five fold cross validation approach: we built SumScore and AveScore with samples in the training set of a given fold; then, for validation, the user validation set given rise to by the same fold was mixed together with the masquerade validation set. At the end of the fifth fold, results are all averaged.

Tables V and VI summarize our experimental comparison. As before, we use bold typeface to stress best results, and follow the same convention in the confection of each table. Looking closely at these tables, we conclude that our locality-based MDS outperforms both SumScore and AveScore. For example, the left part of Table V, which corresponds to our locality-based MDS, contains a majority amount of bold typeface numbers. Further, Table VI shows that in four out of five performance indicators, our locality-based MDS surpasses the others with significant statistical difference. This can be explained by that the SumScore and AveScore are both based on a single measure, and hence get easily overwhelmed by more robust classifiers.

VI. CONCLUSIONS AND FURTHER WORK

The need to protect information has motivated research on *Masquerade Detection Systems* (MDS), which work by detecting anomalies given a profile of user behavior. Existing approaches for user profiling have focused mainly on one type of activity, usually the user actions (command usage, keyboard usage, and so on). By contrast, we have pushed forward considering the objects that are subjects of such actions, although we openly acknowledge that only one type of behavior will not be enough to build an effective MDS.

In this work we introduced a new abstraction called *locality* which is based on the memory-cache term of the same name. We described in detail a set of 16 new temporal, spatial, and directional locality features. In order to assess the effectiveness of these features, we used the new extended WUIL dataset repository. From it, we built a TreeBagger classifier based on locality features. We compared our classifier against other classifier of the same kind but based on WUIL features.[1] We also compared our classifier against others found in the literature [11].

The classifier based on locality features attained a better performance than the others, thus improving our own results presented in [13]. We used AUC, MMP, EER, Zero-FN, and Zero-FP as the metrics for comparison. Particularly, our results based on locality features are very encouraging: an AUC value of 0.97 in average and 30% of users equal to or above 0.99. We conclude that locality features, temporal, spatial, and

[1]WUIL features were introduced in [13] and consist of 25 statistical values.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2571679, IEEE Transactions on Information Forensics and Security

Table V
COMPARATIVE RESULTS OF LOCALITY FEATURES AND GATES *et al.* [11] AVESCORE AND SUMSCORE METHODS.

User	Locality features					AveScore					SumScore				
	AUC	MMP	EER (%)	ZFN (%)	ZFP (%)	AUC	MMP	EER (%)	ZFN (%)	ZFP (%)	AUC	MMP	EER (%)	ZFN (%)	ZFP (%)
1	**0.991**	**195.22**	**3.64**	**6.36**	100.0	0.444	119.1	46.22	100.0	100.0	0.48	124.64	37.68	100.0	100.0
2	**0.947**	**180.91**	**12.9**	**42.14**	**87.83**	0.747	149.52	25.3	100.0	100.0	0.761	149.4	25.3	100.0	100.0
3	**0.991**	**192.82**	**3.88**	**9.4**	**53.33**	0.5	100.64	49.93	100.0	100.0	0.5	100.14	49.93	100.0	100.0
4	**0.992**	**194.5**	**3.72**	**8.1**	**59.26**	0.5	103.97	48.01	100.0	100.0	0.508	104.03	48.01	100.0	100.0
5	**0.993**	**195.84**	**2.57**	**5.53**	100.0	0.781	164.89	18.33	100.0	100.0	0.838	164.49	18.33	100.0	100.0
6	**0.997**	**198.62**	**0.9**	**1.38**	100.0	0.836	170.79	14.63	100.0	100.0	0.854	171.96	14.63	100.0	100.0
7	**0.999**	**199.27**	**0.68**	**0.73**	**46.21**	0.573	115.83	42.1	100.0	100.0	0.588	122.09	42.1	100.0	100.0
8	**0.954**	**187.62**	**8.12**	**17.54**	100.0	0.335	100.0	61.38	100.0	100.0	0.346	100.0	61.38	100.0	100.0
9	**0.974**	**188.51**	**8.56**	**11.49**	100.0	0.759	152.85	23.59	100.0	100.0	0.81	158.54	23.59	100.0	**86.67**
10	**0.994**	**198.06**	**1.74**	**1.94**	100.0	0.922	184.59	7.89	100.0	100.0	0.93	185.2	7.89	100.0	100.0
11	**0.975**	**186.41**	**8.42**	**19.34**	100.0	0.534	116.18	41.95	100.0	100.0	0.595	118.64	41.95	100.0	100.0
12	**0.987**	**192.67**	**4.84**	**11.19**	100.0	0.846	174.1	19.85	100.0	100.0	0.904	174.1	16.4	100.0	100.0
13	**0.978**	**187.78**	**6.2**	**27.15**	100.0	0.767	157.73	23.96	100.0	100.0	0.845	161.56	22.61	100.0	100.0
14	**0.987**	**188.65**	**7.13**	**13.37**	**77.14**	0.728	142.34	28.83	100.0	100.0	0.773	148.73	28.83	100.0	100.0
15	**0.982**	**188.98**	**5.51**	**42.86**	100.0	0.931	181.13	14.04	100.0	100.0	0.965	183.19	9.0	100.0	100.0
16	**0.991**	**192.73**	4.95	**8.47**	**79.26**	0.97	190.32	6.69	100.0	92.59	0.972	191.16	**4.42**	100.0	100.0
17	**0.995**	**195.34**	**2.33**	**7.27**	**85.56**	0.592	125.47	37.28	100.0	100.0	0.616	125.78	37.28	100.0	100.0
18	0.988	190.14	5.26	14.65	100.0	0.958	181.46	10.34	18.54	100.0	**0.992**	**196.22**	**3.18**	**3.78**	**90.0**
19	**0.977**	**185.41**	**9.64**	**18.42**	**71.43**	0.812	167.96	16.09	100.0	100.0	0.858	167.83	16.09	100.0	100.0
20	**0.986**	**191.05**	**6.01**	**9.35**	**74.17**	0.513	124.11	46.2	100.0	100.0	0.632	130.04	47.93	100.0	100.0
21	0.955	179.88	11.96	**74.17**	27.86	0.981	196.43	**3.45**	100.0	3.57	0.981	196.43	5.12	100.0	3.57
22	**0.981**	**190.47**	**5.91**	**14.0**	100.0	0.876	173.48	14.84	100.0	100.0	0.933	178.76	11.04	100.0	100.0
23	**0.986**	**189.97**	**6.05**	**10.03**	100.0	0.93	186.15	6.92	35.84	100.0	0.969	189.3	6.68	34.67	100.0
24	**0.967**	**188.68**	**7.41**	**75.85**	100.0	0.392	100.0	60.37	100.0	100.0	0.395	100.0	60.37	100.0	100.0
25	**0.995**	**194.18**	**3.46**	**8.3**	100.0	0.723	157.03	33.89	100.0	100.0	0.878	167.76	21.29	100.0	100.0
26	**0.992**	**194.26**	**3.51**	**5.81**	100.0	0.727	151.43	25.95	100.0	100.0	0.807	153.68	24.33	100.0	100.0
27	**0.986**	**189.87**	**6.63**	**13.08**	100.0	0.549	117.07	41.46	100.0	100.0	0.56	117.5	41.46	100.0	100.0
28	**0.931**	**181.43**	**9.98**	100.0	**78.0**	0.846	168.79	16.81	100.0	100.0	0.87	170.53	16.81	100.0	100.0
29	0.973	**192.15**	**4.31**	**10.63**	100.0	0.974	184.44	8.33	21.18	62.5	**0.987**	190.0	8.61	20.63	**33.33**
30	**0.987**	**190.08**	**5.36**	**24.15**	100.0	0.567	119.33	40.34	100.0	100.0	0.591	119.51	40.34	100.0	100.0
31	**0.973**	**184.23**	9.53	**15.95**	**74.81**	0.816	173.95	24.14	100.0	100.0	0.921	181.84	**9.08**	100.0	100.0
32	**0.974**	**185.31**	**10.59**	**23.19**	100.0	0.705	152.23	25.63	100.0	100.0	0.812	159.49	24.44	100.0	100.0
33	**0.991**	**194.12**	**4.16**	**7.63**	100.0	0.913	177.68	13.52	100.0	**89.29**	0.923	180.07	11.21	100.0	100.0
34	**0.984**	**191.03**	**5.67**	**9.57**	100.0	0.689	147.14	26.57	100.0	100.0	0.746	148.28	26.57	100.0	100.0
35	**0.981**	**189.0**	**5.5**	**47.32**	100.0	0.453	100.0	50.88	100.0	100.0	0.493	106.16	50.88	100.0	100.0
36	**0.98**	**187.42**	**7.12**	**29.06**	**51.3**	0.679	132.14	39.2	100.0	95.65	0.713	141.07	29.47	100.0	100.0
37	**0.966**	**184.21**	**9.55**	**53.8**	100.0	0.439	100.0	53.47	100.0	100.0	0.454	103.63	53.47	100.0	100.0
38	**0.973**	**183.55**	**8.42**	**35.33**	100.0	0.77	160.93	19.53	100.0	100.0	0.828	161.0	19.53	100.0	100.0
39	0.874	167.51	25.14	41.03	92.0	0.862	**167.69**	27.08	**32.31**	**64.0**	**0.902**	166.67	**20.26**	33.33	72.0
40	**0.992**	**193.92**	**3.38**	**6.09**	73.79	0.955	180.61	10.35	23.9	100.0	0.981	188.07	10.06	20.4	**51.72**
41	**0.995**	**194.93**	**3.58**	**7.47**	**82.86**	0.844	177.98	14.58	100.0	100.0	0.91	178.59	15.44	100.0	100.0
42	**0.957**	**179.75**	**12.35**	**35.07**	**68.42**	0.64	148.96	30.78	100.0	100.0	0.765	148.96	27.3	100.0	100.0
43	**0.975**	183.21	**9.27**	31.07	100.0	0.91	173.17	17.87	26.83	100.0	0.968	**188.52**	10.32	**11.48**	100.0
44	0.954	181.71	11.66	**23.0**	83.45	0.96	185.46	7.52	100.0	100.0	0.96	**186.33**	**6.88**	100.0	**75.86**
45	**0.99**	**189.01**	**5.93**	**13.02**	**62.96**	0.863	168.79	19.27	100.0	100.0	0.934	175.85	14.66	100.0	100.0
46	**0.972**	**188.83**	**7.51**	**28.03**	100.0	0.449	100.0	55.07	100.0	100.0	0.449	100.0	55.07	100.0	100.0
47	**0.939**	180.15	11.93	100.0	51.11	0.935	180.89	9.56	100.0	**25.93**	0.926	180.89	9.56	100.0	37.04
48	**0.927**	**172.17**	**15.62**	100.0	**69.29**	0.713	150.83	31.96	100.0	100.0	0.842	158.33	27.78	100.0	100.0
49	**0.995**	**195.37**	**3.09**	**13.65**	100.0	0.847	167.45	16.75	100.0	100.0	0.867	171.07	16.75	100.0	100.0
50	**0.927**	**177.31**	**12.24**	100.0	100.0	0.48	100.0	51.54	100.0	100.0	0.483	101.05	51.54	100.0	100.0
51	**0.989**	**193.11**	**4.28**	**7.1**	100.0	0.863	176.29	11.85	100.0	100.0	0.903	177.46	11.85	100.0	100.0
52	**0.955**	**179.57**	**10.22**	100.0	100.0	0.872	168.08	17.74	100.0	78.57	0.898	177.75	14.07	100.0	**67.86**
53	**0.962**	**184.59**	**8.78**	100.0	100.0	0.654	136.73	31.94	100.0	100.0	0.707	140.63	31.94	100.0	100.0
54	**0.974**	**185.26**	**8.73**	**16.13**	100.0	0.464	100.0	52.38	100.0	100.0	0.466	100.0	52.38	100.0	100.0
55	**0.974**	**184.11**	**8.48**	**19.64**	100.0	0.877	165.64	24.88	100.0	100.0	0.915	170.49	18.09	100.0	**82.76**
56	**0.977**	**186.9**	**7.63**	**29.78**	100.0	0.759	153.83	23.24	100.0	100.0	0.823	162.67	23.24	100.0	**73.08**
57	**0.988**	**188.41**	**6.66**	**15.82**	100.0	0.81	163.6	21.42	100.0	100.0	0.884	170.44	17.19	100.0	100.0
58	**0.961**	**181.98**	**9.42**	100.0	100.0	0.92	169.85	17.97	100.0	100.0	0.955	181.05	10.55	100.0	**64.29**
59	**0.975**	**183.53**	**8.98**	**26.24**	**80.8**	0.69	144.61	35.38	100.0	100.0	0.806	156.77	23.44	100.0	100.0
60	**0.964**	**182.73**	**10.34**	**20.21**	**93.1**	0.598	121.06	39.47	100.0	100.0	0.612	121.06	39.47	100.0	100.0
61	**0.985**	**193.56**	**3.86**	100.0	100.0	0.749	154.76	22.62	100.0	100.0	0.778	154.91	22.62	100.0	100.0
62	**0.98**	**188.16**	**6.14**	**42.22**	100.0	0.592	132.36	33.82	100.0	100.0	0.646	132.36	33.82	100.0	100.0
63	**0.993**	**192.2**	**4.34**	**12.04**	**47.86**	0.793	161.96	19.14	100.0	100.0	0.836	164.59	19.14	100.0	100.0
64	**0.993**	**196.08**	**3.01**	**3.92**	100.0	0.822	170.08	14.96	100.0	100.0	0.903	176.9	14.92	100.0	**74.07**
65	**0.995**	**196.09**	**2.25**	**4.36**	100.0	0.72	145.33	27.34	100.0	100.0	0.727	145.33	27.34	100.0	100.0
66	**0.971**	**182.6**	**9.96**	**17.4**	**85.33**	0.874	176.45	13.43	100.0	100.0	0.914	175.61	13.77	100.0	100.0
67	**0.99**	**189.97**	**5.37**	**16.43**	100.0	0.695	153.72	26.99	100.0	100.0	0.762	153.72	23.14	100.0	100.0
68	**0.988**	**191.27**	**4.37**	**11.43**	32.59	0.957	189.74	5.85	100.0	**22.22**	0.957	188.31	5.85	100.0	25.93
69	**0.991**	**193.73**	**4.29**	**6.27**	100.0	0.84	173.35	15.01	100.0	100.0	0.94	184.74	7.63	100.0	100.0
70	**0.992**	**192.01**	**5.42**	**9.83**	**75.17**	0.621	125.6	40.55	100.0	100.0	0.644	134.91	40.55	100.0	100.0
71	**0.97**	**181.81**	**12.83**	**18.19**	100.0	0.696	141.07	29.58	100.0	100.0	0.705	140.9	29.58	100.0	100.0
72	**0.966**	**191.9**	**4.17**	100.0	100.0	0.776	164.52	21.05	100.0	100.0	0.858	164.16	17.92	100.0	100.0
73	**0.981**	**189.09**	**7.37**	**23.63**	**72.67**	0.911	177.01	13.73	100.0	100.0	0.934	179.81	12.87	100.0	83.33
AVG	**0.976**	**188.4**	**7.0**	**29.23**	**88.19**	0.741	151.9	26.58	93.95	94.99	0.783	155.36	24.58	93.48	92.08

Table VI
COMPARATIVE EVALUATION OF THE CLASSIFIER IN THE ROW VERSUS THE ONE IN THE COLUMN (INCLUDING GATES *et al.* [11] AVESCORE AND SUMSCORE METHODS). CELL VALUES ARE OF THE FORM (WIN/LOSE/TIE). BOLD TYPEFACE IS USED TO STRESS SIGNIFICANTLY STATISTICALLY PERFORMANCE DIFFERENCE ACCORDING TO THE WILCOXON SIGNED-RANK TEST.

Performance indicators	**Algorithms**	TreeBagger+ WUIL features	TreeBagger+ Locality features	AveScore	SumScore
AUC	TreeBagger + WUIL features		**4/69/0**	**69/4/0**	**62/11/0**
	TreeBagger + Locality features	**69/4/0**		**70/3/0**	**68/5/0**
	AveScore	**4/69/0**	**3/70/0**		**3/68/2**
	SumScore	**11/62/0**	**5/68/0**	**68/3/2**	
MMP	TreeBagger + WUIL features		**7/66/0**	**68/5/0**	**59/14/0**
	TreeBagger + Locality features	**66/7/0**		**69/4/0**	**68/5/0**
	AveScore	**5/68/0**	**4/69/0**		**9/52/12**
	SumScore	**14/59/0**	**5/68/0**	**52/9/12**	
EER	TreeBagger + WUIL features		**9/63/1**	**68/5/0**	**63/10/0**
	TreeBagger + Locality features	**63/9/1**		**70/3/0**	**66/7/0**
	AveScore	**5/68/0**	**3/70/0**		**6/31/36**
	SumScore	**10/63/0**	**7/66/0**	**31/6/36**	
Zero-FN	TreeBagger + WUIL features		**12/57/4**	**55/4/14**	**54/5/14**
	TreeBagger + Locality features	**57/12/4**		**62/2/9**	**61/3/9**
	AveScore	**4/55/14**	**2/62/9**		1/5/67
	SumScore	**5/54/14**	**3/61/9**	5/1/67	
Zero-FP	TreeBagger + WUIL features		18/22/33	**25/8/40**	22/13/38
	TreeBagger + Locality features	22/18/33		**24/7/42**	22/14/37
	AveScore	**8/25/40**	**7/24/42**		**6/11/56**
	SumScore	13/22/38	14/22/37	**11/6/56**	

directional together with the analysis of the interaction of a user with her file system, are very promising when applied to characterize users and detect masqueraders.

Also in this work, we undertook an important update of the WUIL repository, increasing the number of user datasets from 20 to 73. We paid special attention to users 21, 29, 36, 39, 47, and 68 since after we applied our window generation procedure, they ended up with very few windows (below 350). We decided to include these users in all our experiments and we observed that their individual scores were very close to average. This shows that our methodology based on locality features is very robust even when facing very few input data from normal user activity. Therefore, this enhancement places WUIL as one of the largest and most comprehensive dataset repository in masquerade detection literature.

Future work will focus on the merge of locality and WUIL features, instead of treating them separately, by choosing the more representative ones.

ACKNOWLEDGMENTS

This paper was largely improved from the invaluable comments and support from Ph.D. Félix Castro and Ari Yahir Barrera, and students and personnel from the Universidad Autónoma del Estado de Hidalgo. We thank the anonymous reviewers and the members of the GIEE-ML group at Tecnológico de Monterrey for providing useful suggestions and advice on earlier versions of this paper. The first author was supported by CONACYT student scholarship 329962.

REFERENCES

[1] M. Schonlau, W. DuMouchel, W. Ju, A. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical Sci.*, vol. 16, no. 1, pp. 58-74, Feb. 2001.

[2] M. Pusara and C. Brodley, "User re-authentication via mouse movements," *Proc. ACM Workshop on Visualization and Data Mining for Comput. Security (VizSEC/DMSEC)*, pp. 1-8, 2004.

[3] A. Weiss, A. Ramapanicker, P. Shah, S. Noble, L. Immohr, "Mouse movements biometric identification: A feasibility study," *Proc. Student/Faculty Research Day, (CSIS)*, pp. 1-8, May 2007

[4] A. Garg, R. Rahalkar, S. Upadhyaya, K. Kwiat, "Profiling users in GUI based systems for masquerade detection," *Proc. Inform. Assurance Workshop*, pp. 48-54, June 2006.

[5] K. Killourhy and R. Maxion, "Why did my detector do that?!: Predicting keystroke-dynamics error rates" *Proc. Recent Advances in Intrusion Detection (RAID)*, pp. 256-276, Sept. 2010.

[6] K. Killourhy and R. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," *Proc. Int. Conf. Dependable Syst. Networks (DSN)*, pp. 125-134, July 2009.

[7] A. Morales, J. Fierrez, J. Ortega-Garcia, "Towards predicting good users for biometric recognition based on keystroke dynamics," *Comput. Vision Workshop ECCV*, vol. 8926, pp. 711-724 , 2015.

[8] A. Messerman, T. Mustafic, S. A. Camtepe, and S. Albayrak, "Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics," *Proc. Int Joint Conf. on Biometrics (ICJB)*, pp. 1-8, Oct. 2011.

[9] D. Denning, "An intrusion-detection model," *Trans. Softw. Eng.*, vol. 13, no. 2, pp. 222-232, Feb. 1987.

[10] X. Wang, Y. Sun, and Y. Wang, "An abnormal file access behavior detection approach based on file path diversity," *Int. Conference on Inform. and Commun. Technologies (ICT)*, pp. 455-459, May 2014.

[11] C. Gates, N. Li, Z. Xu, S. Chari, I. Molloy, and Y. Park, "Detecting insider information theft using features from file access logs," *European Symp. on Research in Comput. Security (ESORICS)*, vol. 8713, pp. 383-400, Sept. 2014.

[12] B. Camiña, R. Monroy, L. Trejo, and E. Sánchez, "Towards building a masquerade detection method based on user file system navigation," *Proc. Mexican Int. Conference on Artificial Intell. (MICAI)*, vol. 7094, pp. 174-186, Nov. 2011.

[13] B. Camiña, C. Hernández-Gracidas, R. Monroy, and L. Trejo, "The windows-users and -intruder simulations logs dataset (WUIL): An experimental framework for masquerade detection mechanisms," *Expert Syst. with Applicat.*, vol. 41, pp. 919-930, Feb. 2014.

[14] B. Camiña, J. Rodrıguez, and R. Monroy, "Towards a masquerade detection system based on user's tasks," *Symp. on Recent Advances in Intrusion Detection (RAID)*, vol. 8688, pp. 447-465, Sept. 2014.

[15] M. Schonlau (2015, May 20), "Masquerading user data" [Online] Available: http://www.schonlau.net/intrusion.html

[16] R. Maxion and T. Townsend, "Masquerade detection using truncated

command lines," *Proc. Int. Conf. Dependable Syst. and Networks (DSN)*, pp. 219-228, June 2002.
[17] R. Maxion, "Masquerade detection using enriched command lines," *Proc. Int. Conf. Dependable Syst. and Networks (DSN)*, pp. 5-14, June 2003.
[18] S. Greenberg, "Using Unix: Collected traces of 168 users," Dept. Comput. Sci., Univ. of Calgary, Alberta, Rep. 88/333/45, Nov. 1988
[19] A. El Masri, H. Wechsler, P. Likarish, and B. Kang, "Identifying users with application-specific command streams," *Proc. Int. Conference on Privacy, Security and Trust (PST)*, pp. 232-238, July 2014.
[20] F. Linton, D. Joy, H. Schaefer, and A. Charron, "Owl: A recommender system for organization-wide learning," *Educ. Technol. Soc.*, vol. 3, no. 1, pp. 62-76, 2000.
[21] M. Salem and S. Stolfo, "Modeling user search behavior for masquerade detection," *Proc. Symp. Recent Advances in Intrusion Detection (RAID)*, pp. 181-200, Sept. 2011.
[22] S. Yingbo, M. Salem, S. Hershkop, and S. Stolfo, "System level user behavior biometrics using Fisher features and Gaussian mixture models," *Security and Privacy Workshops (SPW)*, pp. 52-59, May 2013.
[23] K. Bache, M. Lichman. (2013). {*UCI*} *Machine Learning Repository* [Online]. Available: http://archive.ics.uci.edu/ml
[24] A. Moulton and S. Madnick, "A temporal and spatial locality theory for characterizing very large data bases," *Proc. Int. Conference on Intelligent Transportation Syst. (HICSS)*, vol. 2, pp. 612-620, Jan. 1989.
[25] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," *Proc. Int. Symp. Comput. Architecture*, pp. 357-368, June 1998.
[26] MathWorks, Inc. (2015, February 15). *TreeBagger* [Online]. Available: www.mathworks.com/help/toolbox/stats/treebagger.html
[27] N. Gu, M. Fan, L. Du, and D. Ren, "Efficient sequential feature selection based on adaptive eigenspace model," *Neurocomputing*, vol. 161, pp. 199-209, August 2015.
[28] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, October 2001.
[29] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*, 2nd ed. New Jersey, John Wiley & Sons, Inc., 2014.

J. 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.

Raúl Monroy obtained a Ph.D. in Artificial 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 México. Since 1998 he is a member of CONACyT's National Research System, currently rank 2. He is the leader of the GIEE-ML (Machine Learning) research group at Tecnológico de Monterrey. Dr. Monroy's research is concerned with the discovery and application of general search control strategies for uncovering and correcting errors in either a system or its specification; for detecting anomalies endangering information security; and for planning robot motion.

Luis A. Trejo obtained a Ph.D. in Computer Science (Parallel Processing) in 1993 from Université Claude-Bernard de Lyon, France. He is a full-time Professor at the School of Science and Engineering, from Tecnológico de Monterrey. Since 2015, he is a member of CONACyT's National Research System, Level 1, and a member of the GIEE-ML (Machine Learning) research group at Tecnológico de Monterrey. His topics of interest are internetworking, Internet of Things, information security, intrusion detection and prevention systems, machine learning, data science, and parallel processing.

Miguel Angel Medina-Pérez graduated in Informatics Engineering from University of Ciego de Ávila, Cuba, in 2007 and received his M.Sc. degree in Applied Informatics from the same university in 2007. He received his Ph.D. in Computer Science in 2014 from National Institute of Astrophysics, Optics and Electronics, Mexico. Currently, he is a Research Professor at Tecnológico de Monterrey, Campus Estado de México, where he is also a member of the GIEE-ML (Machine Learning) research group. His research interests include supervised classification, clustering, data mining and knowledge discovery, feature selection, one-class classification, masquerader detection, fingerprint recognition, and palmprint recognition.