Towards Building a Masquerade Detection Method Based on User File System Navigation

Benito Camiña, Raúl Monroy, Luis A. Trejo, and Erika Sánchez

Computer Science Department
Tecnológico de Monterrey — Campus Estado de México
Carretera al Lago de Guadalupe Km. 3-5, Atizapán, Estado de México, 52926, México
`{a00965049,raulm,ltrejo,snora}@itesm.mx`

Abstract. Given that information is an extremely valuable asset, it is vital to timely detect whether one's computer (session) is being illegally seized by a masquerader. Masquerade detection has been actively studied for more than a decade, especially after the seminal work of Schonlau's group, who suggested that, to profile a user, one should model the history of the commands she would enter into a UNIX session. Schonlau's group have yielded a masquerade dataset, which has been the standard for comparing masquerade detection methods. However, the performance of these methods is not conclusive, and, as a result, research on masquerade detection has resorted to other sources of information for profiling user behaviour. In this paper, we show how to build an accurate user profile by looking into how the user structures her own file system and how she navigates such structure. While preliminary, our results are encouraging and suggest a number of ways in which new methods can be constructed.

1 Introduction

For today's world, where information is reckoned to be an extremely valuable asset, it is vital to timely detect whether one's computer (session) is being illegally seized by some intruder, so-called a *masquerader*. A masquerader aims to put a computer holder into jeopardy, as quickly as possible, while impersonating the holder to avoid being caught.

Masquerade detection has been actively studied since the seminal work of Schonlau *et al.* [1], who suggested that, in order to profile a user, it should suffice to model the history of the (parameterless) commands she would type while logged in into a UNIX session. For that purpose, Schonlau *et al.* developed a masquerade dataset, commonly referred to as *SEA* [2], which has been a *de facto* standard for building, validating and comparing a number of masquerade detection methods (see, for example, [3–6]).

The performance of SEA-based methods, however, cannot be said to be overwhelming. Accordingly, in later experiments, SEA has been enriched to consider additional information; in particular, command arguments (see, for instance, [7]). Also as a result, research on masquerade detection has turned to using alternative sources of activity in an attempt to better characterise user behavior. Example alternative information sources include device usage (*e.g.* the mouse [8], or the well-studied keyboard [9]), and application usage (*e.g.* the use of a document management system [10]). In a similar vein, user activity has been clustered into activity types, such as information gathering, browsing, communications, etc. [19]. These activity types, 22 in total, aim at capturing user intention, and they have been characterized in terms of executions of Windows applications, dynamic link libraries, and Microsoft Disk Operating System commands.

In this paper, we claim that to better characterise user behaviour, it is necessary to consider how and what the user browses while working on her own file system. After all, a user file system is an abstract representation of some user, at least as far as the operating system is concerned. Crucial to our approach is a structure, we call a user *navigation structure*, representing the navigation of a user through her file system. A navigation structure contains information of a user file system. In particular, it contains the most recently visited file system objects; it also contains information about both how a user structures her own file system, and how she uses and browses such directory.

We will show that the aim of using a user navigation structure is twofold. First, a navigation structure allows to better understand the behaviour of a given user. Second, and more importantly, it allows to build a user profile suitable for masquerade detection. While we reckon that a combination of various kinds of user activities will be necessary, at the end of the day, for practical masquerade detection, our results show that file system navigation is central to achieve an accurate user profile.

While preliminary, our results are very promising. We shall see that even Naïve Bayes, when applied as an intrusion detection mechanism on the file system objects visited by a user, surpasses the performance of similar methods, that have been designed to consider alternate sources of information. We shall argue that the information gathered in navigation structures can make this classifier take more informed intrusion detection decisions.

Overview of Paper The remainder of this paper is organised as follows. First, §2, we shall show the limitations of UNIX-commands based masquerade detection methods. In particular, we shall argue that, on the one hand, they perform rather poorly, even when they are not subject to proper masquerader action; and that, on the other hand, they are difficult to implement, because the activity information chosen to profile a user is, *per se*, sparse. Second, §3, we will (operationally) introduce user navigation structures and the operations they take. Third, §4, we shall briefly outline the datasets we have collected throughout our experiments, both (honest) user and masquerader, and shall provide preliminary results on understanding and profiling user behaviour for masquerader detection, using the frequency of access to file system objects. Then, in §5, we shall discuss directions for further research, which aims to truly exploit the information contained in navigation structure. Finally, in §6, we will report on the conclusions we have drawn from our experiments.

2 Masquerade Detection

Masquerade detection is concerned with timely noticing that an intruder is impersonating a legitimate user in a computer session. [1]. It is usually undertaken as an anomaly detection task, where the masquerade detection mechanism aims at distinguishing any diversion in the current user activity from a given profile of ordinary user behaviour. Masquerade detection has been actively studied since the seminal work of Schonlau *et al.* [1], who have developed a database, called *SEA* [2], which is the *de facto* standard for building, validating and comparing masquerade detection mechanisms.

2.1 The SEA Dataset

SEA contains log information about the activity of 70 UNIX users. Each user log consists of 15,000 commands, that were gathered via the `aact` auditing mechanism, stripping off the arguments. 50 users were randomly selected to serve as honest holders, while the remaining ones were (artificially) set to act as masqueraders.

The SEA dataset is structured as follows. Every command sequence of a legitimate user is first divided into blocks of size 100. Each command block is called a *session*; thus, user activity is arbitrarily taken to be composed of, and represented by, 150 sessions. The first 50 sessions of every user are left untouched; they constitute ordinary user behavior, we call *the user history of commands*. The command history set is used as the *construction dataset* for any proposed masquerade detection system. The last 100 sessions, however, may or may not have been contaminated, and they are used as the detection mechanism *validation dataset*.

If contaminated, a validation session is taken to be a masquerade. A validation session is either totally contaminated or not. Session contamination in SEA amounts to replacing the selected, original validation session for one of a user who was arbitrarily set to be a masquerader. Contaminated sessions were inserted using this rule: if no masquerader is present, then a new masquerader appears in the next session with a probability of 1%. Otherwise, the same masquerader continues to be present in the next block with a probability of 80%. SEA comes with a matrix that, for each user, shows where masquerade sessions are located, if any.

As a test dataset, SEA is very restrictive, when more realistic conditions are considered. To begin with, a supposedly masquerader does not have any knowledge about the profile of an intended victim. A masquerade session is just a sequence of commands that a user, unwillingly marked as a ‘masquerader’, would type in a UNIX session. Moreover, in SEA, some of such masqueraders show very simple and repetitive behaviour, which is easily spotted as unusual, even by human inspection.

To overcome these limitations, one may embark oneself in gathering faithful UNIX sessions; however, such task has proven to be complex, given that real UNIX sessions are sparse and, hence, difficult to obtain. This has been shown by Chinchani *et al.* [11], who have attempted to synthesise sequences of user commands, with the aim of enabling the construction of a model for ordinary behaviour. RACOON, Chinchani *et al.*'s tool, synthesises user sessions in order to get around of the inherently long time it takes the collection of real ones. Its ultimate aim is to speed up the process of development and evaluation of masquerade detection systems.

Recently, research on masquerade detection has been looking at alternative sources of user activity to better profile user behavior. Example alternative sources of information include user device usage (*e.g.* the mouse [8]), and user interaction with specific applications (*e.g.* the use of a document management system [10].) There is large archive of research on keystroke dynamics [9]; however, though exhaustive, it makes very unrealistic assumptions, for user activity is not recorded under normal, working conditions, but under very artificial ones.

In a complementary vein, research on masquerade detection has attempted to characterize user activity considering a combination of several observable actions. For example, [19] provides 22 types of user activities, including information gathering, browsing, communications, etc. These activity types are defined in terms of executions of Windows applications, dynamic link libraries, and Microsoft Disk Operating System commands. Thus, the masquerade detection mechanism is ad-hoc, in that it is Operating System dependent, and needs to be adapted whenever a change on platform is in order.

2.2 Masquerade Detection Mechanisms: an Overview

Research on masquerade detection has been very prolific. Space constraints preclude us from giving a reasonable overview, or a fair comparison of these mechanisms (even if restrained to the most significant ones). We shall hence confine ourselves to overview a few of them, referring the reader to [12, 13] for a survey.

A masquerade detector is *global*, if, for determining masquerader presence, it uses both the profile of the user being protected and the profile of that user's colleagues. Otherwise, it is *local*. Similarly, a masquerade detector is *temporal-based*, if, for user profile formation, it considers both individual actions and the relations thereof; *e.g.* action sequences. Otherwise, it is *frequency-based*. Since global, temporal methods are more informed, they are usually more accurate than local, frequency-based ones; however, they demand more information and computing effort, and are inapplicable in some contexts.

Uniqueness [14] is a global, frequency-based detection method, which stems from the observation that the appearance of a command not seen in the construction of a user profile may indicate the presence of a masquerader. A command is said to be of *popularity* i if only i users use that command. It is said to be *unique* if it is of popularity one. The detection model consists of a score, computed on each test session, which increases (respectively decreases) upon the appearance of a unique (respectively unseen) command. Uniqueness has been an obliged reference for comparison purposes.

Customized grammars [5] is global, and temporal-based. It profiles ordinary user behavior using the unique and most repetitive sequences of user actions.

To identify such sequences of actions amongst a user history and across other users' audit logs, customized grammars applies *sequitur* [15], a method for inferring compositional hierarchies from strings. A test session is classified using an evaluation function, which yields a high score if the session includes very long action sequences with a high self, *i.e.* a high frequency in the user-history) and low unself (*i.e.* a low frequency in the all-users-history).

Naïve Bayes [16, 7, 17] assumes that the actions of a user are independent one another. Thus, the probability for an action c to have been originated from a user u, $\Pr_u(c)$, is given by $\frac{f_{uc}+\alpha}{n_u+\alpha\times K}$, where f_{uc} is the number of times user u issued c in the user log of actions, n_u the length of u's log, K is the total number of distinct actions, and where $0 < \alpha \ll 1$ prevents $\Pr_u(c)$ from becoming zero. Thus, naïve Bayes is frequency-based. It might be either local or global, with similar performance [17]. To evaluate a test session s, in which user u has allegedly participated, the cumulative probability of s is compared either against the probability that s has been output by somebody else [16] or against a user-defined threshold [17].

Sequitur-then-HMM [6] is a local, temporal masquerader detector, which characterises user behavior in terms of the temporal dependencies among action sequences of most frequent occurrence. This method first applies sequitur to a given user history of actions. Then, the main production rule of the grammar output by sequitur is used to train a Hidden Markov Model (HMM). Consequently, not only does this model consider temporal dependencies among actions, by virtue of sequitur, but it also considers temporal dependencies among sequences, by virtue of the HMM.

The performance of these (and other) methods, however, is far from conclusive. This can be seen in Figure 1, borrowed from [18] (there TRD stands for the method of [6]), which reports on that most of these methods, when operating at a 95% detection rate, produce over 40% of false alarms. Masquerader detectors have not been fully or faithfully tested. This is because they have not been put to work under stressing conditions, where it is required to distinguish an intruder who actually is aiming at effective user impersonation (see also [18]). From this, it follows that these methods have not been thoroughly compared either.

3 Navigation Structures

To capture how a user uses and browses her file system, we use a navigation structure. A navigation structure contains information about both how a user structures her own file system, and how she uses it and browses it. Objects in a navigation structure have been visited by the user, either directly, from a directory browser, or indirectly, from a user application. A navigation structure contains only user objects, ignoring system- or application-related files.

A navigation structure is composed of two graphs: an access graph and a directory graph. An *access graph* contains a record of the most recently visited file system objects, as well as the order (actually, a traversal path) which the user has followed to visit them. A *directory graph* is a proper subgraph of the

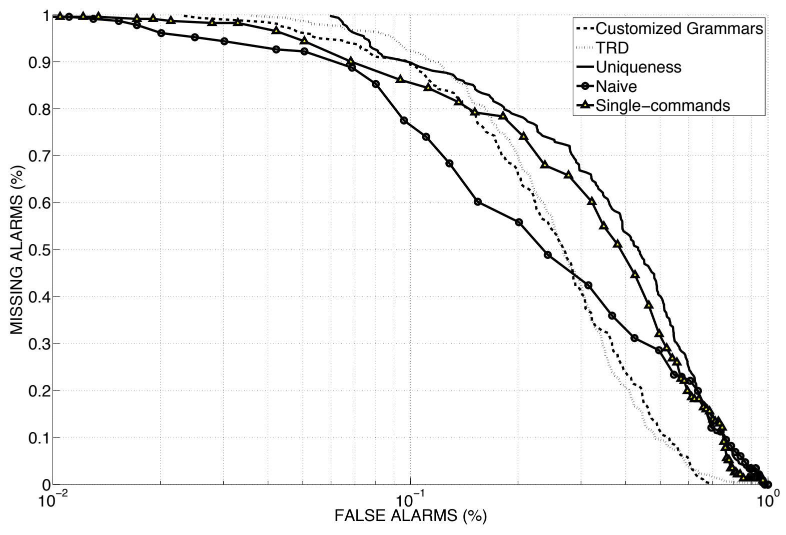

Fig. 1. Performance of detection methods with attack diffusion.

user file system; it is indeed an arborescence, with a distinguished vertex, called the *root*, and denoted by "/".

3.1 Access Graph

Let $\mathcal{FS}$ denote the file system of a given user u.[1] The access graph for u is a directed graph, given by the pair $G = (V, E)$, where $V \subset \mathcal{FS}$ collects all the nodes (files or folders) that u has recently visited, and E records the node visiting history. Arcs in E are annotated: $n \xrightarrow{k} n' \in E$ if and only if, having visited n, u has visited n' next, k times.

Initially, $G = (V, E)$ is such that $E = \emptyset$, and V is a singleton, containing the node representing the root folder. Then, $G = (V, E)$ is updated, upon access to node n', having visited node n, as follows:

$$(V, E) = \begin{cases} (V, (E - \{n \xrightarrow{k} n'\}) \cup \{n \xrightarrow{k+1} n'\}) & \text{if } n \xrightarrow{k} n' \in E \\ (V \cup \{n'\}, E \cup \{n \xrightarrow{1} n'\}) & \text{otherwise} \end{cases}$$

Every node $n \in V$ is a tuple with three elements: i) the path associated to the file system object identified with n, denoted path(n), ii) the position of node n in graph D (see below, §3.2), and iii) the weight of the node. As expected, for every $n \in V$, path(n) is a string of the form "$/(\alpha/)^{\star}\alpha$", where "α" stands for

[1] Given that u is understood from the context, we omit the subscript u in all these symbols, for the sake of clarity.

an alphanumeric, representing the name of the file system object, and $\star$ for the transitive, reflexive closure of string concatenation, denoted by juxtaposition.

The weight of a node n, denoted weight(n), represents the number of times n has been accessed and it is defined as follows:

$$\mathrm{weight}(n) = \mathrm{foldl}(\{\!|k \mid \exists n'.\, n' \xrightarrow{k} n|\!\}, +, 0)$$

where $\{\!|\ldots|\!\}$ denotes a multiset, and where foldl is a higher-order function, given by $\mathrm{foldl}(\{\!|X_0, X_1, \ldots, X_n|\!\}, F, E) = F(\ldots(F(F(E, X_0), X_1), \ldots)\, X_n)$, and where E, and F, respectively, denote the base element, and the step function.

3.2 Directory Graph

A directory graph is a directed acyclic graph representing a subtree graph of the user file system. Let $\mathcal{FS}$ denote the file system of a given user, u, with access graph $G = (V, E)$. Then, the directory graph for u is a pair $D = (V', E')$, where $V \subseteq V'$ and E' denotes the usual descendant relation of a tree. For every node $n \in V$, there are as many nodes in V', one for each file system object along path(n). Arcs in E are ordered pairs: $(n, n') \in E'$ if and only if n' is a descendant of E, in the user file system structure.

Like an access graph, a directory graph, $D = (V', E')$, is initially such that $E' = \emptyset$, and V' is a singleton containing the root node. Then, upon any access to an object o, we first use path(o) to compute the pairs that potentially are to be inserted to update D as follows. Let t be a singleton string. Then,

$$\begin{aligned}\pi_o = \{(n, n') \mid \exists \alpha, \alpha', t.\, &\mathrm{path}(o) = \alpha t \alpha', \\ &n, n' \text{ are created such that } \mathrm{path}(n) = \alpha,\ \mathrm{path}(n') = \alpha t\}\end{aligned}$$

where we assume pairs can be freshly created as required. Next, for every pair $(n, n') \in \pi_o$, we update $D = (V', E')$ as follows:

$$(V', E') = \begin{cases} (V' \cup \{n'\}, E' \cup (n, n')) & \text{if } (n, n') \notin E' \\ (V', E') & \text{otherwise} \end{cases}$$

Every node $n \in V'$ is annotated with accounting information. Formally, it is a tuple of three elements: i) the path associated to the file system object identified with n, ii) the time the object was last accessed, and iii) the position of node n, with respect to D, denoted pos(n, D, path(n)). Given that D is of the form: $/(\alpha_1, \ldots, \alpha_k)$; *i.e.* it has top node / and k object children, we define node position by:

$$\mathrm{pos}(n, /(\alpha_1, \ldots, \alpha_k), \mathrm{path}(n)) = \begin{cases} [\,] & \text{if } \mathrm{path}(n) = / \\ [i] & \text{if } \mathrm{path}(n) = /\alpha_i \\ [i]\,\mathrm{pos}(n', \alpha_i, \mathrm{path}(n')) & \text{if } \mathrm{path}(n) = /\alpha_i \mathrm{path}(n') \end{cases}$$

Navigation Structure Maintenance To keep the navigation structure of a given user to a manageable size, we have readily applied Least Recently Used, a page replacement algorithm for virtual memory management. Other algorithms, such as FIFO or Most Frequently Used are both applicable and easy to implement, while others, *e.g.* Second Chance, or Not Used Recently require us to include more accounting information inside a navigation node. We update a navigation structure whenever required; for example, after the user has moved or removed a file system object. We shall have more to say, later on in text, see §5, about how to exploit a navigation structure in the study of the masquerader detection problem.

4 Some Preliminary Results

4.1 Construction- and Validation-Masquerade Datasets

Datasets During the experimental phase, we invited six people, three to act as ordinary users (see Table 1), and the rest as masqueraders (see Table 2). On each user machine, we ran a process that, in the background, built the corresponding user navigation structure. Profile construction was, according to users, completely unnoticeable, and was carried out during normal working days. By way of comparison, the activities of masqueraders were tracked, while they were allowed a free five-minute navigation on the machine of each legitimate user; on each attempt, masqueraders aimed to compromise the victim user, as much as possible.

User	Profile	Logging		Log size	Directory
		Starting date	Ending date		Tidiness
1	Assistant: invoice and purchase orders procedures	12/04/2010	01/24/2011	3697	Low
2	Employee: Logistics and process documentation	12/17/2010	01/24/2011	2338	Medium
3	Manager: Finance and accounting	01/03/2011	01/24/2011	11494	High

Table 1. User profiles

Users and intruders, as can be noticed from Tables 1 and 2, are of different nature. We selected them this way driven towards validating our working hypotheses, namely: that there are users who are easy to impersonate, and that, conversely, there are intruders who are readily detectable.

Naïve Bayes As a classification model, we used Naïve Bayes, using access frequencies of objects (see Section 2.2). Hence, information about the behavior of the user under protection is needed. In our case, an action c, issued by user, u, is interpreted as user u has accessed file system object, c. Hence, the cumulative probability of a session s of length n is given by: $\Pr_u(s) = \Pr_u(c_1) \times \cdots \times \Pr_u(c_n)$. To evaluate a test session s, in which user u has allegedly participated, the cumulative probability of s has been compared against a user-defined threshold. If the probability is below this threshold, the session is considered normal, and, in that case, the user profile is updated. Otherwise, user u is considered a masquerader.

Intruder	Profile	Intruder log size generated against		
		user 1	user 2	user 3
1	PhD student, highly skilled in information security	969	444	1076
2	Lawyer with basic knowledge of computer usage	239	224	230
3	MSc student using specialised tools to obtain sensitive data	107	366	146

Table 2. Intruder profiles

4.2 Experimentation Results

Figure 2 summarizes, via a ROC curve, our results of experimentally testing Naïve Bayes against all three masqueraders. Our results are very promising, suggesting that file system navigation is a good means for masquerade detection. Two conclusions follow. First, it is more difficult to detect a masquerade attempt against a user who carries out a few, specific tasks, or who organizes badly his directory structure; this might be explained by that the access probability distribution associated to a user of this kind is near the uniform. For example, masquerade attempts against user 3 were readily detected (0% false negatives and and 1.75% false positives); by contrast, attempts against user 1 were difficult to spot (10.96% false negatives and 20.16% false positives). Second, it is more difficult to detect an occasional or novice intruder; for example, our worst results were obtained on masquerade attempts of intruder 2 against user 1. This actually is good news, because, using our approach, masquerading becomes a challenge for a highly skilled intruder; this is because he must behave as the legitimate user, while compromising sensitive data (at first sight, a contradictory goal.)

5 Discussion

Although the above results are promising, we reckon that taking full advantage of a navigation structure will improve the performance of masquerade detection mechanisms. In this section, we lay out a few ideas which might be helpful for further research.

Task and Distribution Probability of Task Transition People tend to collect similar or related objects into one or more common folders. Using this observation, we conjecture that whenever a user is carrying out a specific task, she will visit objects of a few folders, and that these folders will all be closed one another too. Then, we may define a single, unique folder to be considered as an abstract representation of a task, and call that folder simply a *task*.

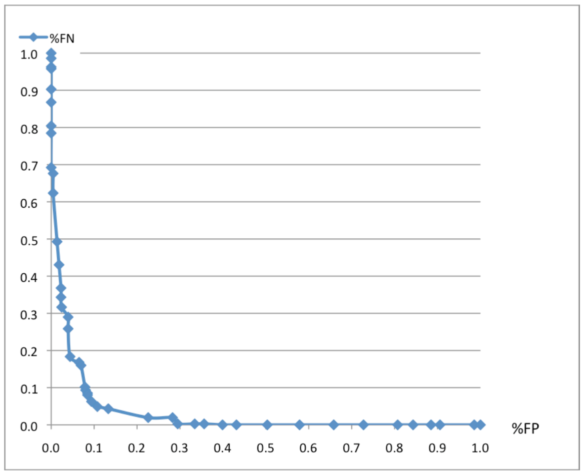

Fig. 2. ROC curve: masquerade detection results

In what follows, we aim to formalise this notion of task. Let $G = (V, E)$ be the access graph of a given user u. Let $n, n' \in V$. Then, $n \ll n'$ if and only if there exists a nonempty substring α, such that $\text{pos}(n') = \text{pos}(n)\alpha$. Let $\aleph(S)$ stand for the cardinality of set S. Then, the load of n, denoted load(n), is given by $\text{load}(n) = \aleph(\{t' | (t \xrightarrow{k} t') \in E \text{ and } n \ll t'\})$. Now, a node n is said to be a *task of* u if and only if $\text{load}(n) \neq 0$ and, except, possibly, from the root node, /, there does not exist a node t', such that $t' \ll n$, and that $(t \xrightarrow{k} t') \in E$.

Using this notion of task, t, together with a relation of task activity, which consists of visiting nodes underneath t, we have, experimentally, proven that the principle of *spatial* locality of reference holds for the behaviour of a user u. Figure 3 depicts spatial locality for user 3; there, we have conducted a simple normalisation of the position of the accessed object for the coordinate; this has been done so that two objects that, relative to their positions, are close one another, they they will also get close coordinate values. Similar experiments need to be conducted towards establishing temporal locality of reference.

Now, we could apply the notion of task to define a Markov chain in terms of a *task transition probability*. This Markov model could be used to compute the probability of reaching a particular task, having departed from some other, not necessarily different task, after a few accesses. This probability could be applied in a model for masquerade detection. Of course, a penalisation would have to be considered, if there is an access to a node that does not belong to any fixed task. Other Bayesian models, such as Hidden Markov Models, could be applied to obtain a masquerade detection method.

Task Fault Rate Actually, an access to a node that is not in the access graph could be regarded as a *task fault*. Then, the rate at which task faults occur could

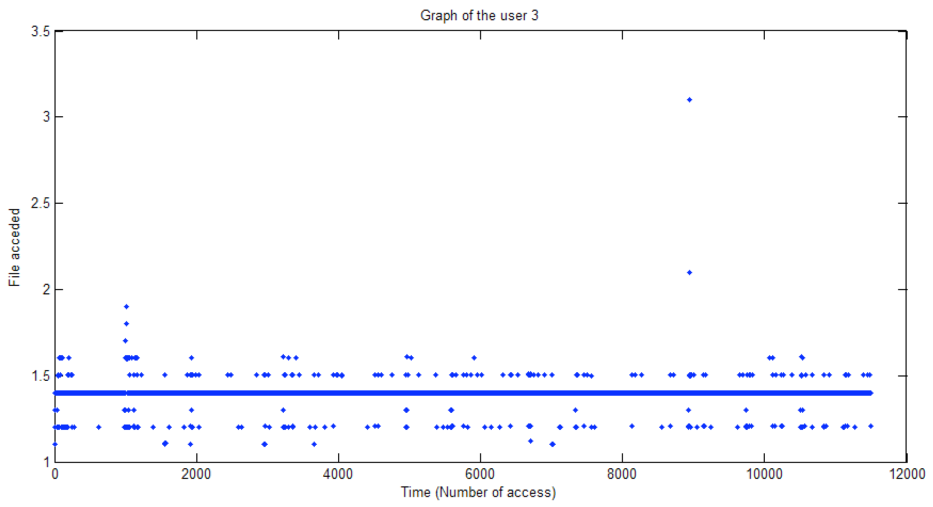

Fig. 3. Spatial Locality of reference for user 3

also be applied in a model for masquerade detection. Figure 4 depicts how task faults occur when intruder 1 is trying to masquerade user 3. We have found that, under attack, task faults occur much more frequently than under normal conditions.

Depth in File System Navigation Other measure that can be exploited to masquerade detection is the depth of node n, given by $\text{depth}(n) = \text{length}(\text{pos}(n)) + 1$. Then, we may try to model the behaviour of a user in terms of the depth at which she usually navigates through her file system. The lower and the upper quantiles associated to a box plot could, for example, be directly used to masquerade detection.

6 Conclusions

Towards the characterisation of file-system user navigation, we have proposed here a navigation structure. In a preliminary experiment, we have built a masquerader detection mechanism, based on naïve Bayes, applying one output of file system navigation, namely: the frequency of visited objects. Our results are very promising, showing a 1.75% false positive ratio and 0% false negative ratio, for a particular user, for the best case, and 79.84% of detection rate, at a cost of a 10.96% of false positive detection rate, for the worst case.

Given people diversity, a user profile can be more suitable than others to detect some form of intrusion. On the one hand, our results show that a very tidy and highly structured user will be much easier to separate from an actual masquerader. By way of comparison, a user who carries out a very restricted

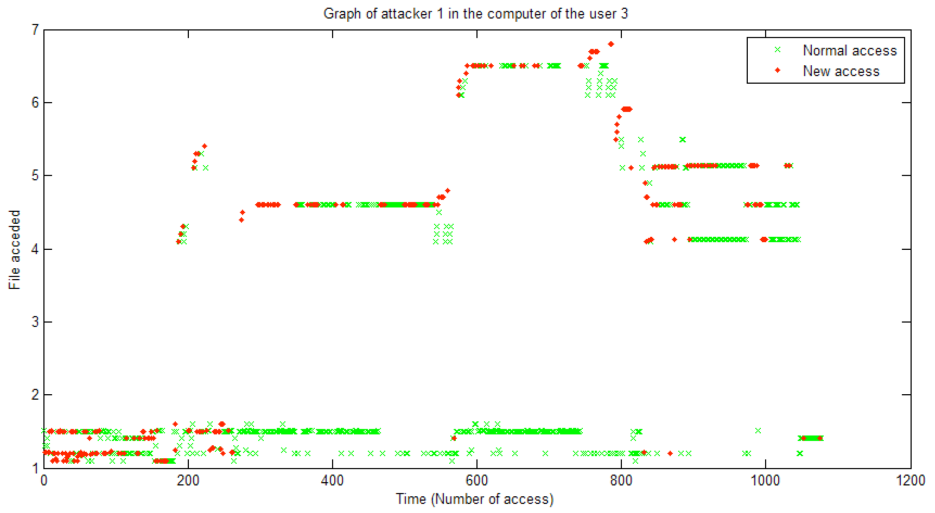

Fig. 4. Object creation rate when intruder 1 is masquerading user 3

collection of tasks may yield an object access probability distribution close to the uniform, and, hence, she might be very easy to masquerade. On the other hand, a highly skilled masquerader is more likely to obtain sensitive information in a short period of time. However, this behaviour might be much easier to spot, given the number of previously unseen visited objects he will raise to during an intrusion. Thus, a good profile, based on a navigation structure, makes a masquerader of this sort to become more easily detected. An occasional masquerader, however, will be harder to spot, due to his errant, and hesitant behaviour.

There exists a number of methods with which we can build a masquerade detection mechanism. In this paper, we have explored one method only. Further research is needed in order to explore the additional ideas laid out in this paper. Although a combination of various kinds of user activities will be necessary, at the end of the day, for practical masquerade detection, our results show that file system navigation is central to achieve an accurate user profile.

References

1. Schonlau, M., DuMouchel, W., Ju, W., Karr, A., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades. Statistical Science **16** (2001) 58–74
2. Schonlau, M.: Masquerading user data. http://www.schonlau.net (2008)
3. Maxion, R.A., Townsend, T.N.: Masquerade detection augmented with error analysis. IEEE Transactions on Reliability **53** (2004) 124–147
4. Oka, M., Oyama, Y., Abe, H., Kato, K.: Anomaly detection using layered networks based on eigen co-occurrence matrix. In Jonsson, E., Valdes, A., Almgren, M., eds.: Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004. Volume 3224 of Lecture Notes in Computer Science., Springer (2004) 223–237

5. Latendresse, M.: Masquerade detection via customized grammars. In Julish, K., Kruegel, C., eds.: Proceedings of the Second International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, DIMVA 2005. Volume 3548 of Lecture Notes in Computer Science., Springer (2005) 141–159
6. Posadas, R., Mex-Perera, C., Monroy, R., Nolazco-Flores, J.: Hybrid method for detecting masqueraders using session folding and hidden markov models. In: Proceedings of the 5th Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence. Volume 4293 of Lecture Notes in Computer Science., Springer (2006) 622–631
7. Maxion, R.A.: Masquerade detection using enriched command lines. In: Proceedings of the International Conference on Dependable Systems and Networks, DSN'03, San Francisco, CA, USA, IEEE Computer Society Press (2003) 5–14
8. Garg, A., Rahalkar, R., Upadhyaya, S., Kwiat, K.: Profiling users in GUI based systems masquerade detection. In: Proceedings of the 7th IEEE Information Assurance Workshop, IEEE Computer Society Press (2006) 48–54
9. Killourhy, K.S., Maxion, R.A.: Why did my detector do *that*?! - predicting keystroke-dynamics error rates. In Jha, S., Sommer, R., Kreibich, C., eds.: Recent Advances in Intrusion Detection, 13th International Symposium, RAID 2010. Volume 6307 of Lecture Notes in Computer Science., Springer (2010) 256–276
10. Sankaranarayanan, V., Pramanik, S., Upadhyaya, S.: Detecting masquerading users in a document management system. In: Proceedings of the IEEE International Conference on Communications, ICC'06. Volume 5., IEEE Computer Society Press (2006) 2296–2301
11. Chinchani, Ramkumar, Muthukrishnan, A., Chandrasekaran, M., Upadhyaya, S.: RACOON: Rapidly generating user command data for anomaly detection from customizable templates. In: Proceedings of the 20th Annual Computer Security Applications Conference, ACSAC'04, IEEE Computer Society Press (2004) 189–204
12. Salem, M.B., Hershkop, S., Stolfo, S.J.: A survey of insider attack detection research. In Stolfo, S.J., Bellovin, S.M., Hershkop, S., Keromytis, A., Sinclair, S., Smith, S.W., eds.: Insider Attack and Cyber Security: Beyond the Hacker. Advances in Information Security. Springer (2008) 69–90
13. Bertacchini, M., Fierens, P.: A survey on masquerader detection approaches. In: Proceedings of V Congreso Iberoamericano de Seguridad Informática, Universidad de la República de Uruguay (2008) 46–60
14. Schonlau, M., Theus, M.: Detecting masquerades in intrusion detection based on unpopular commands. Information Processing Letters **76** (2000) 33–38
15. Nevill-Manning, C.G., Witten, I.H.: Identifying hierarchical structure in sequences: a linear-time algorithm. Journal of Artificial Intelligence Research, JAIR **7** (1997) 67–82
16. Maxion, R.A., Townsend, T.N.: Masquerade detection using truncated command lines. In: Proceedings of the International Conference on Dependable Systems & Networks, Washington, DC, IEEE Computer Society Press (2002) 219–228
17. Wang, K., Stolfo, S.: One-class training for masquerade detection. In: Proceedings of the 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security, IEEE (2003)
18. Razo-Zapata, I., Mex-Perera, C., Monroy, R.: Masquerade attacks based on user's profile. Journal of Systems and Software **?** (2011) ?–? Submitted for evaluation.
19. Ben-Salem, M., S., S.: Modeling user search behavior for masquerade detection. Computer Science Technical Reports 033, Columbia University (2010)