Identification of Irregularities and Allocation Suggestion of Relative File System Permissions

S. Parkinson[a,*], A. Crampton[a]

[a] *Department of Informatics, School of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, HD1 3DH, UK*

Abstract

It is well established that file system permissions in large, multi-user environments can be audited to identify vulnerabilities with respect to what is regarded as standard practice. For example, identifying that a user has an elevated level of access to a system directory which is unnecessary and introduces a vulnerability. Similarly, the allocation of new file system permissions can be assigned following the same standard practices. On the contrary, and less well established, is the identification of potential vulnerabilities as well as the implementation of new permissions with respect to a system's current access control implementation. Such tasks are heavily reliant on expert interpretation. For example, the assigned relationship between users and groups, directories and their parents, and the allocation of permissions on file system resources all need to be carefully considered.

This paper presents the novel use of statistical analysis to establish independence and homogeneity in allocated file system permissions. This independence can be interpreted as potential anomalies in a system's implementation of access control. The paper then presents the use of instance-based learning to suggest the allocation of new permissions conforming to a system's current implementation structure. Following this, both of the presented techniques are then included in a tool for interacting with Microsoft's New Technology File System permissions (NTFS). This involves experimental analysis on six different NTFS directories structures within different organisations. From using this tool we can establish the effectiveness of the developed techniques by evaluating the true positive and true negative instances. The presented results demonstrate the potential of the proposed techniques for overcoming complexities with real-world file system administration.

Keywords: access control, auditing, decision support.

1. Introduction

File systems are common amongst the majority of computer operating systems and from a user perspective their primary use is to store files in an organised and accessible manner. Modern, multi-user computer systems contain high quantities of data that require strong access control mechanisms to restrict data access to intended users. Different operating systems provide different implementations of access control. However, common to the most prevalent is that they provide a customisable architecture for access control. This is implemented through the use of both *coarse*- and *fine*-grained permissions (De Capitani di Vimercati et al., 2003). Coarse-grained permissions are predefined levels (e.g read, write, full control, etc.) and fine-grained permissions are customised permissions created from a set of predefined attributes to represent highly customised access control rules.

Administration of file system permissions on large file systems is both a challenging and cumbersome task as there is a large volume of information to consider. Due to the complexities of managing permissions,

*Corresponding author
Email address: s.parkinson@hud.ac.uk (S. Parkinson)

unforeseen weak and incorrect allocations are often made. These complexities are usually the result of there being a large number of directories to secure, a large number of users that need to be correctly assigned, and a large number of access control rules. There is a wide range of literature discussing these complexities (Cao and Iverson, 2006; Beznosov et al., 2009; De Capitani di Vimercati et al., 2003), and many factual guides have been produced for different operating systems (Thomas, 2010; Solomon, 2005).

When analysing for vulnerabilities within file system permissions they can be divided into two groups of; (1) *known* system vulnerabilities, and (2) those *relative* to the access control structure implemented within a system. An example of a known system fundamental is that users should not have access to an important system directory (e.g `C:\windows\system32`). These are programmatically easy to find by using a predefined knowledge base of potential vulnerabilities. Identifying such vulnerabilities is at most $O(n \times v)$ where n is the number of access control entries to examine and v is the number of known vulnerabilities. An example of a relative vulnerability is an incorrect assignment of permission with respect to an organisation's implementation of access control. For example, the *anomaly* of one user having write privileges on a directory where all other users have read access. Such anomalies are very difficult to identify within access control as there is no quick method of determining potential vulnerabilities. Throughout this paper the words anomaly and irregular are used interchangeably to state a permission which deviates from the normal distribution of permissions within a given directory structure.

In addition to the examination of permissions, allocating new permissions whilst complying to known good practice is typically part of a system administrator's job and will often require collaboration (Haber et al., 2011). Complying to good practice when implementing new permissions is achievable through the use of well defined guides (Russinovich, 2012; Russinovich et al., 2012; Govindavajhala and Appel, 2006). However, less well defined, and often more complicated to perform, is the implementation of new permissions in-keeping with the current implementation of access control.

The aim of this paper is to introduce a technique for identifying irregular relative file system permissions, as well as suggesting suitable allocation methods that are relative to a system's access control structure. This is accomplished by developing suitable models and techniques and then testing them on Microsoft's New Technology File System. The primary contributions presented in this paper are:

- **A novel technique for identifying irregular file system permissions**. This technique takes an *object-centred* modelling perspective to file system access control. Following this, a statistical analysis (χ^2) technique is used to identify permissions which fail a test of dependence. Empirical analysis is then provided to evaluate the effect of directory complexity, and the frequency distribution of permissions.

- **Novel use of an instance-based learning algorithm for allocations suggestion of relative permissions**. A technique is developed which implements a k-NN algorithm and can aid with the implementation of new file system permissions by suggesting suitable access control rules based on the system's current implementation. Once again, empirical analysis is then provided to evaluate the effect of directory complexity and the frequency distribution of permissions.

- **A novel tool for interacting with Microsoft's NT file system**. Although the techniques provided in this paper are file system independent, a file system dependent tool (`ntfs-r.exe`) has been implemented for interacting with NT file systems. This tool implements a depth-first recursive directory search and implements an algorithm for efficiently calculating the *effective* permission of an interacting object.

- **Empirical Analysis and Case Study**. The significance of the proposed techniques and tool is then evaluated on through the use of many real file systems to determine the detection results (i.e. false positive, and false negative rate). These results show an accuracy fraction of 0.911 for permission analysis, and 0.805 for permission allocation (see Section 6.2 for explanation).

The remainder of the paper is organised as follows. In Section 2 information regarding the structure of access control on different platforms is discussed. This then leads to a discussion regarding the administrative complexity that is a result of their implementation. Section 3 contains a survey of related work and discusses state-of-the-art in permission administration and auditing. Section 4 presents the development of a model

which is then used alongside a statistical test of independence to determine irregular permissions. Examples and empirical analysis regarding performance and accuracy are also presented. Following this, a model is then developed in Section 5 to be used alongside an instance-based learning algorithm to aid with the suggestion of new allocations. In Section 6 the development of a tool which implements the proposed techniques for the NT file systems is presented. Following this, the tool and the proposed techniques are evaluated in Section 6.2 on multiple real-world file systems to determine their efficiency and correctness.

2. Background and overview

Access control is typically defined as a relational model over the following domains: O the set of objects (i.e users), the set of resources R and the set of permissions P. Access control is a characteristic function on the set $A \subseteq S \times O \times R$. A subject s is granted permission r over resource o iff $\langle s, o, r \rangle \in A$. Access control models are typically called the access matrix. In many operating systems the access matrix is stored as an access list, which is associated with a resource object and is used to list all subjects and their permissions. In NTFS, access lists are implemented as Discretionary Access Control List (DACL) models, where only the owner of a resource is authorised to change its access permissions. However, in multi-user environments it is possible for any user with sufficient permission to take ownership of a resource or change its permission.

Other operating systems also implement access control lists to manage file system permissions. Unix (including Max OSX and Linux) and Portable Operating System Interface (POSIX) compliant systems have a simple system for managing individual file permissions. This is the ability to assign a predetermined set of coarse-grained permissions (often called "traditional Unix permissions"). Most of these systems also support the use of Access Control Lists (ACL), such as POSIX.1e ACLs (coarse-grained) (Nemeth, 2010) or NFSv4 ACLs (fine-grained) (Pawlowski et al., 2000). Interestingly, the implementation between NFSv4 and NTFS ACLs is very similar and also caters for fine-grained permissions. The ability to create fine-grained permissions significantly increases the complexity of the access control implementation. For this reason, and due to strong similarities between NFSv4 and NTFS, the rest of this section will describe in more detail the access control implementation of NTFS.

2.1. Access Mask

NTFS implements DACLs by applying a DACL to each object within the file system. Each DACL will contain a Security Identifier (SID) which is a unique key that identifies the owner of the object and the primary associated group. The structure of the DACL is a sequential storage mechanism which contains access control entries (ACEs). An ACE is an element within a DACL which dictates the level of access given to the interacting subject. The ACE contains a SID that identifies the particular subject, an access mask which contains information regarding the level of permissions and the inheritance flags. An ACEn (permission) within the NTFS is a set of attributes, p, from the predefined set of attributes $p \subseteq A$, $A = \{a_1, \ldots, a_n\}$. Table 1 shows the fourteen predefined attributes in the NTFS. The NTFS provides six levels of standard coarse-grained permission that consist of a combination of predefined attributes. It is also the case that NTFS allows for the creation of special fine-grained permissions which consist of any combination of the fourteen individual attributes (Russel et al., 2003).

Creating a special permission for most is a very useful feature; however, it can often be a source of confusion as it requires the complete understanding of the authority that each attribute holds (Thomas, 2010). An example of having to use special permissions is when you wish to assign to a group of users the standard privilege elevation of `Modify` for all the contents of a shared folder. However, creating an ACE with the modify permission on the folder explicitly will result in the user being able to delete the folder itself rather than the child objects. The solution to this problem is to assign to the group or user the default permission level of Modify, and then modify the permissions' attributes turning it into a special permission so that only subfolders and files can be deleted.

Attribute Number (a)	**Description**
1	Full control
2	Traverse folder \ execute files
3	List folder \ read data
4	Read attributes
5	Read extended attributes
6	Create files \ write data
7	Create folders \ append data
8	Write attributes
9	Write extended attributes
10	Delete subfolders and files
11	Delete
12	Read permissions
13	Change permissions
14	Take ownership

Table 1: Individual attributes.

2.2. Propagation and Inheritance

It is necessary to discuss the different mechanisms through which NTFS permissions can propagate throughout the directory structure. Within the DACL there are two types of ACE; (1) Explicit and (2) Inherited. Explicit entries are those that are applied directly to the objects' DACL, whereas inherited are those that are propagated from their parent object. The type of ACE allows to determine whether the permission was assigned directly to the directory in question (explicit) or if it was inherited from the directory that it resides within (inherited). Furthermore, the creation of fine-grained special file system permissions also allows for the creation of custom fine-grained inheritance rules. Special inherited permissions can be different depending on whether the ACE has the container inherit ACE bit flag set which controls whether the ACE is applied to all the children objects or not. The creation of fine-grained propagation rules can easily be overlooked and can ultimately result in the unintended propagation of access.

2.3. Accumulation

Accumulation is the possibility for the subject to receive the effective permission of multiple different policies. This feature is prominent within the NTFS resulting in the possibility for a subject to receive permissions from multiple different ACEs within the same DACL. Furthermore, any subject that interacts with the NTFS can be assigned to any number of groups, which can be entered into the ACE. This means that the user does not have to be directly entered into the ACE, they could simply be a member of the group that is entered. The policy combination is handled within the operating system by the Local Security Authority Subsystem Service (LSASS). This service combines the permissions together to effectively create an ordered union of all the policies. There are few complexities within permission accumulation due to the structured way in which ACEs are processed. These are: (1) Explicit permissions take precedence over inherited permissions, (2) Explicit deny permissions always take precedence over apply permissions, and (3) Permissions inherited from closer relatives take precedence over relatives further away.

It might be expected that deny permissions always take precedence over apply permissions to ensure that during the policy combination stage the user always operates at the least possible privilege elevation. However, the first point regarding explicit permissions taking precedence over inherited permissions can result in a situation where an inherited deny permission is never reached. This goes against a fundamental aspect of policy combination to ensure that a deny permission is never ignored. If the case where a user is able to ignore a deny permission to receive access was to either intentionally or unintentionally arise, the system administrator needs to be made aware of this situation. In addition to the explicit permissions

taking precedence over inherited permissions, inherited permissions that are of closer distance to the invoked object will take precedence over more distant relatives. For example, a folder's inherited permissions will take precedence over those from their grandparent.

2.4. Group Membership

A fundamental aspect of access control within the NTFS is that of group membership. A subject (group, user or process) that interacts with the file system can be a member of any group. This means that permissions can be inherited from any of the associated groups if they are entered within any DACL. Subjects, in this case users, will often be grouped together (by separation of duty) to make management easier, and as Hanner, et. al. (Hanner and Hörmanseder, 1999) identifies, understanding effective file permissions can become significantly more complex by group association. To correctly evaluate a user's effective permissions we would have to know which groups they are a member of. We should note that this is not directly related to the mechanism of how NTFS implements access control, it is an unavoidable component of how Microsoft allows for users, groups and processes to be managed by group association.

3. Related Work

There are many tools available to assist with the examination of NTFS permissions allocation (Microsoft, 2006b,a). However, there is one common weakness in that they all still require expert knowledge to analyse the output of the tools to determine if there are any weaknesses. Previous work in the area of file system permission analysis resulted in the development of a novel tool for permissions administration that allows users to easily view file system permissions for large directories (Parkinson and Crampton, 2013; Parkinson and Hardcastle, 2014; Parkinson et al., 2016). The tool provides features to help the user identify vulnerabilities and view necessary information to make better informed permission allocations. For example, the reduction in reported permissions by not displaying inherited permissions, and allowing the user to filter and show effective permission for a specified user, are two prominent features.

Identifying anomalies or irregularities in system security is by no means a new topic (Lazarevic et al., 2003; Bhuyan et al., 2014). For decades, researchers have been developing techniques and tools to identify security anomalies. For example, recent works have covered topics from the identification of anomalous user behaviour in social networks (Viswanath et al., 2014), anomalies in network traffic (Catania et al., 2012; Mahoney, 2003), anomaly detection in wireless networks (Islam and Rahman, 2011; Xie et al., 2011), and the anomaly detection in power station security (Ten et al., 2011). All these studies generate good results and demonstrate the potential of using machine learning and statistics to identify anomalies. Recent research in the area of file systems includes the construction of a Bayesian network and neural network from the predetermined knowledge of the manipulation of file system artifacts for file system forensic analysis (Khan, 2012). Research has been carried out into developing tools for identifying vulnerabilities in file system access control based on some predetermined knowledge-base of vulnerability definitions. Nalgurd et. al. (Naldurg et al., 2006) use an inference engine alongside a snapshot of the access control implementation with static knowledge, and in further work they produce a technique where a model checking algorithm (Naldurg and KR, 2011) is used to identify vulnerabilities.

These methods show significant ability at identifying vulnerabilities, but they require statically defining a knowledge-base of vulnerabilities in respect to relative file system permissions which is heavily reliant on expert knowledge. The focus of the work presented in this paper is to provide a mechanism to identify likely vulnerabilities without the reliance on expert knowledge. To the best of the authors' knowledge there has been no work in the detection of irregularities in file system security without the need to statically define what constitutes an irregular permission. This is surprising considering the vast user base and the potential benefits that can be achieved.

4. Measuring Independence

In this section we show how anomalies in file system permissions can be measured as the independence of a permission, p, from a subject's, s, (e.g user or group) distribution of permissions. This object-centred

approach is taken as we are interested in permissions that are irregular for an interacting subject, rather than what is irregular for a directory. Furthermore, the allocation of irregular permission attributes provides a greater granularity of analysis. The notation of an attribute, a, from the set of permissions, $p\{a_1, a_2, \ldots, a_n\}$ is used. Using statistical analysis to determine irregular permissions will most likely categorise irregular, but correctly assigned, file system permissions. This is because in large multi-user systems there are many file system permissions which are implemented for only a few people. However, identifying these permissions as irregular is still useful as it is important that they are monitored and removed when necessary. It is also important to be aware of them when assigning new permissions as if more users are requiring this custom permission, then it would be sensible to form an access control group.

4.1. χ^2 Analysis

χ^2 statistics are used to measure the lack of independence between a and s_j - which can then be compared to the χ^2 distribution with one degree of freedom to judge extremeness (Greenwood, 1996). The χ^2 statistic is chosen as it is a well established technique for measuring independence. For example, it has been successfully used in text categorisation (Yang and Pedersen, 1997; Aas and Eikvil, 1999). Other techniques are available for measuring independence. However, χ^2 is not only computationally easy to compute, it is also a non-parametric test which makes no assumption regarding the distribution of the population (Balakrishnan et al., 2013). This makes it a suitable candidate for the novel work presented in this paper. Using a two-way contingency table for attribute a and subject s where: A is the number of times a and s_j co-occur, B is the number of times a occurs without s_j, C is the number of times s_j occurs without a, D is the number of times neither s_j or a occur, N is the total number of attributes to examine.

From this we can measure the lack of independence between attribute a and object s_j by:

$$\chi^2(a, s_j) = \frac{N(AD - CB)^2}{(A + B)(A + C)(B + D)(C + D)} \tag{1}$$

The χ^2 statistic has a natural value of zero if a and s_j are independent. Therefore, it can be assumed that any permission attribute a that has been assigned to subject s with a χ^2 value close to zero is either an anomaly or an irregular permission attribute. Following the calculation of χ^2 scores, it is then useful to compute the mean χ^2 for each permission using the following equation where l is the number of attributes specified for a permissions, $|p|$:

$$\chi^2_{avg}(p) = \frac{\sum_{j=1}^{l} \chi^2(a, s_j)}{l} \tag{2}$$

This allows us to identify permissions which appear irregularly. However, difficulty arises when deciding the cut-off threshold for χ^2 that should be treated as potentially problematic. Expert analysis would help separate the anomalies from regular permissions, but in some cases such expert knowledge is not available. Therefore, in this paper a technique is presented which attempts to firstly classify χ^2 scores which are most likely to be anomalies or irregular. To perform this classification, Jenks natural breaks classification method (Jenks, 1967) to determine the best arrangement of values into different classes. This is performed by minimising each class's standard deviation, whilst maximising the standard deviation between classes. The class with the minimum standard deviation (i.e lower χ^2 scores) is the class of permissions which have failed the test of dependence and are to be treated as potential anomalies. To perform this the following classification function is used:

$$I(x) \colon \{1..n\} \mapsto \{1..k\} \tag{3}$$

where n is the number of data samples, and k is the number of classes where $k \leq n$. S_j are the set of indices that map to class j. The minimal sum of standard deviations ($SDD_{n,k}$) is then calculated by:

$$SDD_{n,k} = \frac{min}{I} \sum_{j=1}^{k} ssd(S_j) \tag{4}$$

Directory	Group/User	Permission	$\chi^2_{avg}(P)$	Class No.	Irregular
C:\test					no
	Administrators	Full control	59	2	no
	System	Full control	59	2	no
C:\test\dir1					
	Administrators	Full control	59	2	no
	System	Full control	59	2	no
C:\test\dir2					
	Administrators	Full control	59	2	no
	System	Full control	59	2	no
	Test user	Full control	3	1	yes

Table 2: Permission independence scores for the three test directories(`\test`, `\test\dir1`, and `\test\dir2`) residing within `C:\work`.

$ssd(S_j)$ is the sum of the squared deviations of the values of any index set S calculated using the following equation where A is an ordered set of χ^2 scores.

$$ssd(S) = \sum_{i \in S} A[i] - \left(\frac{\sum_{i \in S} A[i]}{|S|} \right) \tag{5}$$

In the first instance of computing $SDD_{n,k}$ values, $k = 2$ are then incremented by 1 until there is no further improvement between the current and previous $SDD_{n,k}$ value. At this point it is assumed that the optimum set of classes has been found. Following the classification of χ^2 scores, it can be assumed that the first class containing those χ^2 scores close to zero (S_1) are likely to be irregular or anomalies. However, the case may arise where multiple classes (e.g. S_1, S_2) both contain permissions that are irregular or anomalies. It is also possible that in the case that no anomalies are detected, the lowest class (S_1) will be contain χ^2 values for correct permissions. To avoid both these situations, the two following rules has been derived based on empirical analysis: (1) The mean for the lowest S_j should be under 5, and (2) The different between S_j and S_{j+1} should be no more than S_j.

4.2. Experimental Analysis

To demonstrate the suitability of χ^2 analysis for identifying irregularities on an albeit trivial example, an example is provided consisting of the creation of a directory (`\test`) with two subdirectories (`\dir1`, `\dir2`). Each of the directories has a permission entry for the `Administrators` and `System`, both of which have the permission of `Full control` assigned. A further permission entry has been made for user `Test user` of `Full control` to the `\dir2` subdirectory. The test directory structure is a subdirectory of a folder (`C:\work`) containing a further 687 directories with 5496 permission entries spread across a total of 5 different users and groups. In this example there are no entries for `Test user` anywhere other than `\dir1`, and therefore is regarded as an irregular permission. It is also important to note for reproducibility that the allocation (permission level, group membership, etc) of the 5496 is not important providing they do not involve `Test user` and are not `Full control`.

The results shown in Table 2 show $\chi^2_{avg}(p)$ scores for each permission entry in the test directory. The $\chi^2_{avg}(p)$ are calculated from the $\chi^2(a, s_j)$. The lower the $\chi^2_{avg}(p)$ value, the greater is the independence between a subject and a permission entry. In Table 2 the score for the `Test user` is significantly lower than all other subjects. Table 2 also shows that the entries have separated into classes using the Jenks natural break method previous described. It is then possible to determine that the lowest class contains permissions that are to be treated as irregular. In this example the `Test user` is the only entry in the first class and has therefore been correctly identified as an anomaly. This shows that the technique has correctly identified an irregular permission. However, from Table 2 it is not possible to establish the ranking of `Test user's` independence compared to all other permission allocations, and it is not feasible to include all of

the information from analysing C:\work within this paper. However it is worth noting that `Test user` is the only entry in the lowest class and that all the other permissions not displayed in Table 2 are in classes 3 and beyond.

4.3. Scalability

In this section the scalability of the proposed χ^2 analysis and the Jenks classification is tested to establish the effect of increasing the directory size, as well as to determine the relationship between the frequency of a permission occurring and correct classification. Although an increasing directory size will certainly increase computation time, it should not affect the proposed technique's ability to identify an anomaly. That is because: (1) if the directory size and allocation of permissions (regular and irregular) increases proportionally, the most infrequent permissions would still be classified as irregular, and (2) if the regular permissions increase but the irregular permission frequency stays the same, or increase by a lower proportion, then its χ^2 score will decrease meaning that it is still classified as an anomaly. Computation time is considered to establish how long it would take to analyse directory structures of different sizes. As this technique is executed offline, increasing computation time is only detrimental to the user.

The main concern regarding scalability of the technique is around at what point a file system anomaly, if frequently reoccurring, is no longer identifiable (i.e a state of convergence has been reached). The following experiment has been performed to gain an understanding of this relationship. The same directory structure will be used as in Section 4.2 to identify the anomaly of user `Test user` having `Full control` on a directory that they should not have access. The number of directories that `Test user` has `Full control` of will be increased first in multiples of 1 until 10, multiples of 5 until 50, and then in 25 until convergence is reached. This is sufficient to gain a comprehensive understanding of scalability. Figure 1 illustrates the results from this experiment. The graph shows the difference between $\chi^2_{avg}(p)$ and the next most significant permission. The x-axis in the graph details represents the total number of anomalous permissions the `Test user` holds within the directory structure. However, as `Test user` has no other permissions, it can also be stated that these are the total number of permissions for `Test user`. This allows us to establish by what margin p_j is independent from p_{j+1}. From the graph it is evident that the measure of independence is decreasing as the number of permissions `Test user` has increases. The experiment was stopped at 100 permissions as the total number of permissions reached 516. Even though the analysis is still suggesting that `Test user` has the highest measure of independence, it is no longer feasible to consider `Test user` as an anomaly. The reason that `Test user` still has a high measure of dependence is because the remaining 416 permission entries are distributed across four other groups (`System`, `Administrators`, `Authenticated Users`, and `Users`. This results presented in Figure 1 also demonstrate the potential implication of incorrect classification of regular permissions should a user have fewer permissions than other users within the system. For example, from Figure 1 it can be seen that a permission is more likely to be classified as an anomaly if it appears infrequently when compared to other permissions within the system. It can therefore be deduced that an infrequent yet correct permission is likely to be incorrectly classified if there are not enough permission entries for a user within the directory structure to make it appear regular in terms of frequency.

Calculation of χ^2 scores is quadratic and so has a complexity of $O(n^2)$. Therefore, the time taken to calculate χ^2 for large file systems can be large. For example, analysing a file system with 350 directories containing a total of 1,400 permission entries made up of 19,600 attributes (from a finite set of 14) took twenty two seconds on a 3.6 GHz i7 CPU with 16GB of available RAM. The first system would double in size in each iteration to gain an understand regarding the performance. Figure 2 illustrates the increasing computation time as the directory size increases. This shows concerns regarding the scalability of the technique. However, execution time is not a primary concern as the primary use of this technique is for auditing purposes and is performed offline. It is also important to note that permissions analysis does not have to occur during extraction. The tool provided in this paper (`ntfs-r.exe`) allows the permissions to be extracted and then analysed at a later a more convenient time. The time required to perform the extraction of file system permissions only occupies a low portion of the total execution time. For example, the example provided with 350 directories required 8 seconds to extract the file system information and store it in a persistent form for later processing.

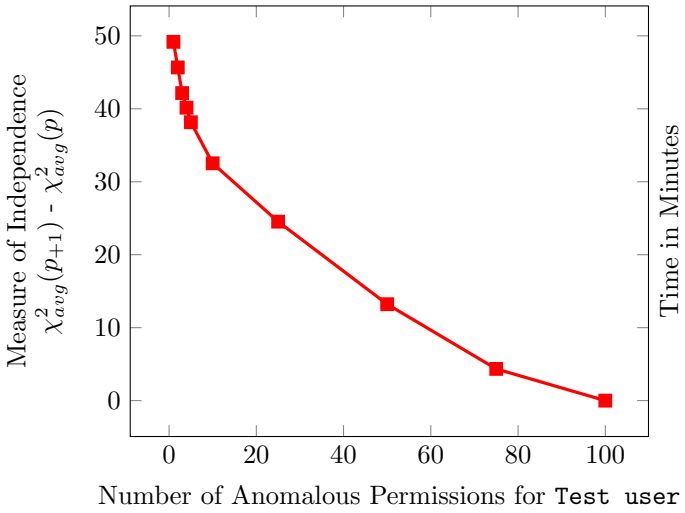

Figure 1: Results from calculating the independence of `Test user` with an increasing number of associated permissions. The graph shows the difference between $\chi^2_{avg}(p)$ and the next most significant permission. Therefore, the higher the number the better.

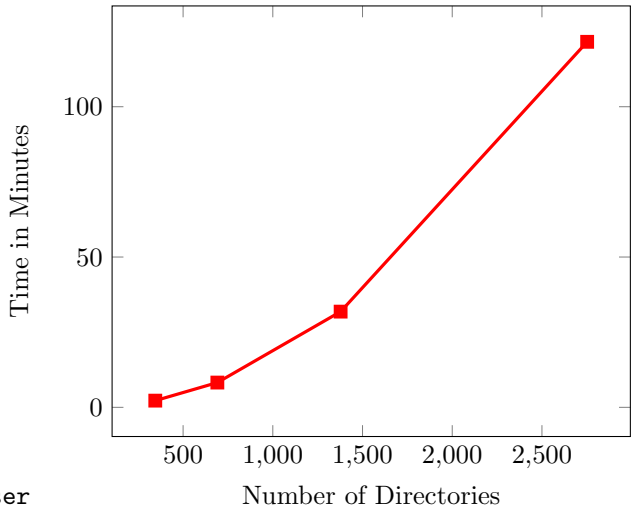

Figure 2: Required computation time for computing χ^2 scores.

	Directory (e.g C:\test)													
Subject (s)	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
Administrators - Full control	1	1	1	1	1	1	1	1	1	1	1	1	1	1
System - Full control	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Test user - Modify	1	1	1	1	1	1	0	1	1	1	1	0	0	1

Table 3: Example data structure for a test directory with three permissions: 1) `Administrator - Full control`, 2) `System - Full control`, and 3) `Test user - Modify`.

5. Allocation Suggestion

As described in the Introduction, allocating new file system permissions which are relative to the current implementation requires careful examination. It is for this reason that in this section a technique is proposed for suggesting new permission allocations relative to the rest of the allocations within the system. This technique aims to find similar allocations that already exist within the file system. This will result in a suggestion and a measure of which other permissions have been implemented to the required level, and therefore help to maintain a consistent file system permissions implementation structure. This technique makes the assumption that a directory structure is homogeneous in that existing and desired permissions are comparable. This assumption is based upon the fact that the granularity of access control rights in NTFS is finite and that a frequently used subset will emerge in large systems.

5.1. k-Nearest Neighbours

k-Nearest Neighbours (k-NN) is a type of instance-based learning which trains the classifier function locally by a majority vote of its neighbouring data points (Altman, 1992; Givens and Hoeting, 2012; Manning et al., 2008). k-NN collects all available data points and classifies new data based on a similarity measure. The idea is that the k-NN method will assign new unclassified examples to the class that the majority of its k nearest neighbours belongs. This make the k-NN particularly attractive when trying to suggest new file system permission which are in keeping with current allocations.

Applying a k-NN algorithm to allocation suggestion for file system permissions requires representing a file systems' access control implementation as a matrix. As the k-NN is scoring a new permission against

another subject's permission, it is necessary to adopt a subject-centred data structure. The data structure for each subject, s, consists of a set of attributes, $\{a_1, a_2, \ldots, a_n\}$. Each attribute has a value of 1 if it is assigned, and 0 otherwise.

These values are stored within a matrix, X, where the set entries, $\{s_1, s_2, \ldots, s_n\}$, represent the matrix rows. An example for one directory is provided in Table 3 where the data structure for three subjects on one directory is shown. To add another directory, the number of columns in the structure would increase by the number of attributes that the subject holds. The $|X|$ is equal to the number of directories residing within the directory structure in question multiplied by the number of attributes that the subject holds on each directory. The example is considering the NTFS access control structure where there are fourteen individual attributes.

Using X it is possible to find the k-nearest neighbours of a constructed s among all the training data, and score the subject candidates based on the score of k-neighbours. Here s is our constructed matrix row which represents a new (user specified) permission entry. The k-NN score is a measure of similarity between s and each s_n in X. A similarity measure of zero would mean that a perfect match has been found. By ranking the similarity scores, it is possible to gain an understanding of which permissions in the file system are most similar to what is required. To calculate this distance, $d(p)$, the following Euclidean distance (Cooke and Clarke, 1989) function is used:

$$d(p) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{r}(x_j - y_{ij})^2}, \tag{6}$$

where, n is the number of attributes in the new permission, r is the number of attributes for each permission entry (14), x_j is an attribute within the new permission at position j, and y_{ij} is the attribute among the training data at position ij. Each distance score, d, is then ranked in order of similarity, d_1 ¡ d_2.

5.2. Experimental Analysis

This section demonstrates the suitability of using k-NN to identify matching permission entries, which can aid when implementing new permissions, using the same example as used for when evaluating the χ^2 analysis. This involved the use of three directories, two groups, and one user. However, modifications have been made to the permissions allocation to add in more variety. This includes modifying the entries seen in Table 2 with the removal of `Test User` and the introduction of `Test Group` with the permission of `Modify` on all three directories.

A new user `Test user` has been introduced to the system and requires `Modify` access on `C:\test\dir2`. The first stage is to compute the training matrix, X, and the required permissions, s. The size of the matrix for this experiment is 42 as there are three directories to process. The need to consider all the directories is to consider permission inheritance and the subject's permission on other directories. The next stage is to compute the Euclidean distance between permission entries in X and s, and the final stage is to order the distances to determine the best match.

The ordered results from performing this experiment are: (1) `Test Group 4.69`, (2) `Administrator` 5.56, (3) `System` 5.56. This clearly identifies the nearest neighbour as `Test group`. However, this is an example containing few directories and objects. A larger and more realistic example is now provided where it is required to allocate the same `Test user` a permission entry of `Read and Execute` on a networked directory of `\\server\share\admin`. This directory structure that `\admin` resides within contains 158 directories, and there are 7 access control groups used to restrict access.

Table 4 describes the results from performing the k-NN analysis. The table shows the distance score for each permission implementation as well as the level of permission that it maintains. The results in this table demonstrate the ability of this technique as well as the significance of the ordered list. It can be seen that the lowest distance not only matches the permission that the new entry requires, but, as the distance increases, the level of permission also increases. The reason that the value is not zero is because the new entry only required `Modify` on `\admin` and `AD_share_users` has permission elsewhere within the `\share` directory. These results are promising, however, it is still necessary to establish the accuracy of the proposed technique on large multi-user file systems.

Object	Euclidean Distance	Permission
AD_share_users	30.1	Read and Execute
AD_stud_admins	41.6	Modify
AD_cmsx_c	41.6	Modify
AD_scom_staff	41.6	Modify
AD_seng_staff	41.6	Modify
AD_backup_admin	46.9	Full control
AD_domain_admin	46.9	Full control

Table 4: Results demonstrating the suitability of k-NN analysis when finding a suitable group to assign `Test user` a `Read and Execute` level of permission.

Subject	Permission	Experiment 1 ($d(p)$)	Experiment 2 ($d(p)$)	Experiment 3 ($d(p)$)	Experiment 4 ($d(p)$)
Admin	Full control	98	139	196	278
System	Full control	98	139	196	278
Authenticated Users	Read	87	123	174	246
Users	Read and Execute	64	91	128	182
Test group	Modify	5	5	5	3
Time (minutes)		2.1	3	12	50
Number of Directories		9618	19306	38542	77014
Matrix Size		134652	270284	539588	1088196

Table 5: Results demonstrating the effect of an increasing directory size on finding a suitable match for applying `Test user` `Modify` rights. Scores in each experiment column are Euclidean distance measures.

5.3. Scalability

It is important to establish the accuracy of using k-NN in order to suggest which groups a subject can become a member of or other subjects with similar permissions to use as a reference when assigning new file system permissions. This section is aimed at gaining an understanding of scalability in terms of: (1) The effect of file system complexity, and (2) How to identify the point at which the k-NN technique does not provide useful results.

Similar to evaluating the effect of increasing directory size on calculating χ^2 values, the same directory structure (Section 5.2) is going to be adopted to include the request to assign `Test user` a permission level of `Modify` on `C:\test\dir2`. The results shown in Table 5 clearly illustrate that an increasing directory size actually has a positive effect on identifying the most suitable group. However, it should be noted that this is because `Test group` is only applied to the three directories (`C:\test`, `\dir2`, `\dir2`) whereas the other four groups are allocated on each of the other directories residing in `C:\test`. Therefore, as the number of directories in our experiment increases, the difference between the number of permissions assigned between `Test group` and the other four groups also increases.

It is important to establish a relationship between the number of permissions that an subject has on the ability to identify k-nearest neighbours for a new permission. To investigate this relationship, Experiment 2 is used. The experiment will be iteratively executed with an increasing number of `Modify` permissions for `Test user`. The number of permissions is increased first in multiples of 1 until 10, then in multiples of 10 until 100, and then by 50 until `Test group` is no longer identified as the k-nearest neighbour.

The results shown in Figure 3 illustrate that the correct group (`Test group`) is identifiable until after 200 occurrences when it is no longer the entry with the lowest distance. At this point, there are a total of 1592 permission entries and 200 (12.6 %) of those are directories that have an entry for `Test group` or `Modify`. This raises some doubt regarding the applicability of this technique to large file systems. This will be further evaluated and discussed in Section 6.

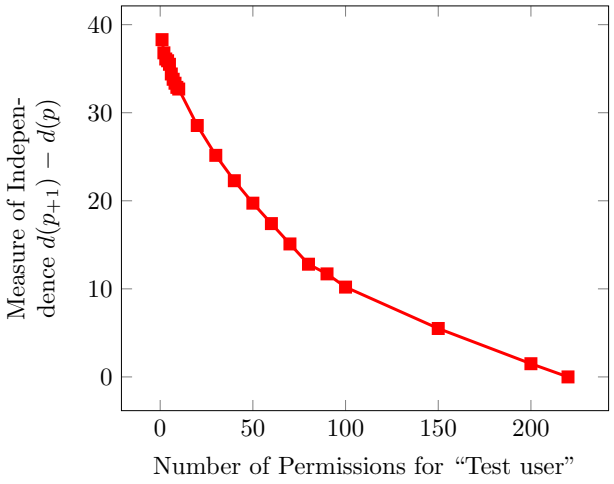

Figure 3: Results from calculating the difference between the two lowest k-NN values.

Calculating k-NN values has a linear complexity of $O(m \times n)$, where m is the number of rows and n is the number of columns within the matrix. It is for this reason that the computation time of calculating k-NN is insignificant when compared to calculating χ^2 values. For example, calculating χ^2 values for a directory size of 1378 required 32 minutes, whereas calculating k-NN values takes around 30 seconds.

6. NTFS Analysis Tool

The previous sections have described how χ^2 and k-NN techniques can be used for identifying anomalies in file system permissions, as well as identifying similar permissions within file system permissions which can help when applying new permissions. Both techniques are used to perform administrative actions which are *relative* to the current file system permission structure. In the following section, the development of a tool is presented which allows the user to perform relative administration of Microsoft's New Technology File System (NTFS). The tool is named `ntfs-r`.

6.1. Implementation and Use

The produced tool (`ntfs-r.exe`) was developed in C# using the .NET `System.Management namespace`. The developed application is a command line application which can be executed on any Windows NT operating system. The application will run under the user's credentials, and therefore can only analyse directories on which the user has authority to read security permissions. Instructions on how to use the tool, as well as the tool itself can be found at the web address provided in Section 8.

6.1.1. Permission Accumulation Algorithm

`ntfs-r.exe` has two main algorithms for computing χ^2 and k-NN scores based on the Section 4.1 and Section 5.1, respectively. However, before performing any analysis, the application recursively processes a specified file system to acquire all the permissions for processing. To account for permission inheritance, Algorithm 1 calculates the effective permission for each entry within the DACL.

As described in Section 4.1, performing χ^2 analysis is $O(n^2)$ and processing directory structures of large sizes (such as those seen in commercial environments) is computationally expensive. However, ignoring subdirectories inheriting permissions from their parent can significantly reduce the quantity of directories that the application needs to analyse. This decision was taken as in many real-world file systems, the number of nested directories inheriting permissions can become large. For example, in one directory structure used in the empirical analysis section of this paper, the total number of directories is 2856 directories of which only 55 have permissions different from their parent. Ignoring directories with permissions different from

Algorithm 1: Depth-first recursive directory search, returning the effective permission of a specified user or group.

```
   Input: Initial directory d
   Input: Initial group or user u
   Output: Set of ordered directories and ACEs P = (d_1, d_2, ..., d_n) where d_n is the directory and a set
           of permissions {p_1, p_2, ..., p_n} for each ACE in d_n's ACL
1  Algorithm algo()
3  | P ← proc(d)
5  | return
6
1  Procedure proc(directory d)
2  | pACL ← d(ACL)
3  | foreach subdirectory c of d do
4  | cACL ← c(ACL)
5  | if cACL != pACL then
6  | | ex = ∅, in = ∅
7  | | foreach ACE a in cACL do
8  | | if isExplicitDeny(a) then
9  | | | P ← (c, a)
10 | | | break
11 | | else
12 | | | else if isExplicitAllow(a) then
13 | | | | ex ← a
14 | | | else if isInherited(a) then
15 | | | | in ← a
16 | | | P ← (c, calculatedEffective(ex, in))
17 | | end
18 | |
19 | end
20 |
```

their parent is not detrimental as long as the number of directories does not drop to a point where an anomalous permission is no longer identified. It is worth noting that this has not not been an issue when processing all the directory structures within this paper.

The depth-first algorithm shown in Algorithm 1 has complexity $O(n)$ where n is the number of directories to process. Algorithm 1 is used to store the explicit ex and inherited in permissions based on the inheritance and propagation. This algorithm considers both the inheritance and deny hierarchies. For speed purposes the algorithm can identify deny permissions and stop the algorithm from continuing to examine the ACL. Line 16 shows that once the explicit and inherited permissions have been identified a function is then called to calculate the effective permission. In this algorithm $calculatedEffective(explicit, inherited)$ represents a native Microsoft .NET command that is able to return the effective access mask (i.e the accumulated permissions for a subject). Using this native method ensures that the correct effective permission is reported.

6.1.2. χ^2 Algorithm

Once all of the permissions for a directory structure have been acquired, it is then possible to process them and look for anomalies. The algorithm for calculating χ^2 values processes the output of Algorithm 1 (P) and computes χ^2 scores. The algorithm has a complexity of $O(n^2)$, where n is the number of permissions to process. This algorithm grows exponentially as the size of P increases, and therefore any reduction in the size of P can have significant performance benefits. The algorithm iterates over P and calculates the two-way contingency table for each attribute followed by calculating the χ^2. Following this, the $\chi^2(P)$ scores

Number	Directories	Permissions levels	Permission Entries
1	55	2	16
2	36	8	248
3	108	3	1142
4	407	6	3719
5	42	2	1708
6	553	4	3184

Table 6: Test NTFS directory structures specifics. The numbers are only considering directories with permissions different from their parent. I.e. not inherited.

are calculated and then separated into respective classes using the Jenks natural break method. The classes are then processed to identify those containing irregular permission.

6.1.3. k-NN Algorithm

k-NN analysis is the other analysis technique implemented in `ntfs-r.exe` that can be performed once the directory structure has been processed. The computational complexity of k-NN is $O(m \times n)$ where m is the number of objects specified in P and n is the total number of attributes identified within the directory structure. Typically, the number of identified objects will be much smaller than the number of directories. For example, in all the examples presented in this paper so far, the number of objects has not been greater than six. As n will grow with the number of directories, the reduction of permissions presented in Algorithm 1 can have significant benefits. This reduction technique also has the potential to improve accuracy as in Section 5.3 it was established that the presented k-NN algorithm is sensitive to the number of permission entries for a given object. Reducing the number of directories by looking at permission inheritance will in turn reduce the number of duplicate permissions.

6.2. Empirical Analysis

In this section, a case study is presented where `ntfs-r.exe` is used on large, multi-user networked NTFS directory structures. In the first section, the ability to identify irregular permissions is tested, and in the second, the ability to suggest suitable access control allocations which are relative to the systems implementation. In both tests, to asses the performance, the following measures are considered: (1) True Positive Rate (*tpr*): the fraction of irregular permissions correctly identified as irregular; (2) False Positive Rate (*fpr = 1 - tnr*): the fraction of regular permissions incorrectly identified as irregular; (2) True Negative Rate (*tnr*): the fraction of regular permissions correctly identified as regular; (3) False Negative Rate (*fnr = 1- tpr*): the fraction of regular permissions incorrectly classified as irregular; Finally, the *accuracy* is reported as the fraction of all samples correctly identified.

The software is tested on a total of six previously unseen multi-user networked NTFS directory structures, each of which is implemented in a different organisation. This allows us to evaluate the tool's characteristics whilst minimising bias. Some organisations might have good access control implementation which is well maintained, while others may have a vast number of users making management difficult and often resulting in quick 'ad-hoc' fixes. Table 6 details the specifics of each file system under test. It is worth mentioning that the number of directories presented in Table 6 is significantly lower than the total number of directories in the file system. This is because our algorithm only looks at directors which have permissions different from their parent. For example, file system Number 1 actually contains 2856 directories of which only 55 have permissions which are different from their parent. In all the case studies, the ground truth (I.e. the list of irregular permissions derived from expert knowledge) is acquired through the reports produced by an independent security auditing company.

6.2.1. Identifying Irregular Permissions Test

From analysing the Receiver Operator Characteristics (ROC) (Hanley and McNeil, 1982) curve in Figure 4 it can be established that the technique works well to classify both regular and irregular file permissions.

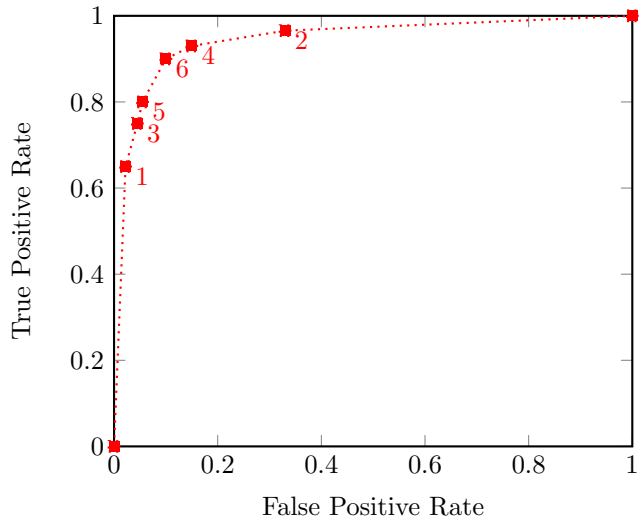

Figure 4: ROC curve for the analysis of the test file systems

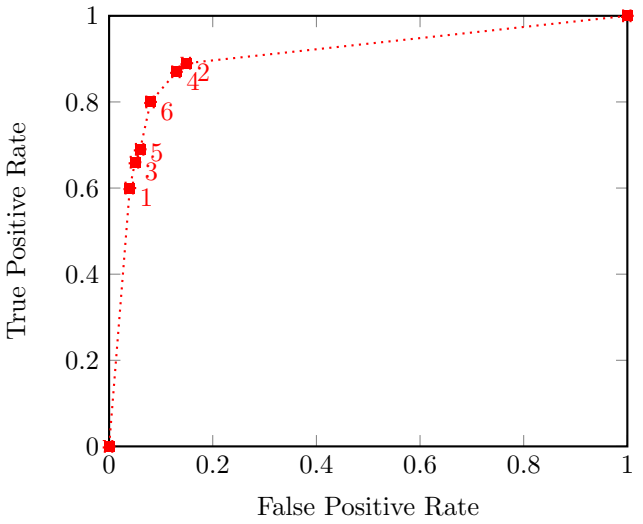

Figure 5: ROC curve for allocation suggestions on the test file systems

The average accuracy ratio for all six samples is 0.911 which indicates a good level of performance. The reason for this ratio not being higher is due to the fact that in some of the tested file systems (especially those managed in an ad-hoc way) there were large volumes of irregular permissions. In some cases some irregular file permissions were classified as normal due to the frequency of irregular permissions. Conversely, on file systems with well structured access control, the accuracy improves and the techniques becomes more useful, and less expert knowledge is required to interpret the results. This can be seen in Figure 4 where the *tpr* is best for directories 1, 3, and 5. The *fpr* increases slightly for directory number 6; however, this is interesting as the directory size and number of permissions for directory 6 are significantly higher than 1 and 3, and 5 but the *tpr* only decreased by 0.045. This is significant as it demonstrates that the accuracy is worse on file systems with diverse and poorly structures permissions and that directory size has little effect.

6.2.2. Permission Allocation Test

The second technique to be demonstrated in this case study is the suggestion of new allocations. To establish the error rates and accuracy of the technique. The allocation suggestion technique is going to be performed on the same six file systems, where the software will be challenged with identifying a new allocation rule for each permission level on each file system. This results in 20 different tests which should help to establish the suitability of the proposed technique. 10 of the samples are positive tests where a new allocation is required for a permission which already exists in the file system. The remaining 10 are negative tests where allocations are required for a permission level which does not exist in the file system.

The results are presented in the ROC curve in Figure 5. The average accuracy ratio for all six samples is 0.805. Although ratio indicates reasonable at best, it does demonstrate that in most cases the tool can suggest valid allocations, thus minimising the reliance on expert knowledge. The file systems that the tool scored badly on are 4 and 2 in Table 6. In particular, the negative tests resulted in the suggestion of a non-valid allocation. Both directories are of different size and contain a different number of permissions. However, what is common is the diversity of permissions allocation. Therefore, it can be stated that the tool's accuracy diminishes as the file system complexity increases, and thus the reliance on expert knowledge also increases.

7. Conclusion

This paper describes the complexities associated with NTFS permission administration that results due to their implementation. The paper discusses how these complexities can result in the misallocation of file system permissions which could potentially result in a vulnerability. The paper has identified the difference between the administration of known file system vulnerabilities and those relative to a system's implementation of access control. The difficulty of identifying these relative permission vulnerabilities motivated the exploration of applying novel techniques to file system administration.

The first technique explored is the use of statistical analysis (χ^2) to measure the independence of file system permissions. The implementation resulted in the production of a method where a higher measure of independence indicated that a permission was irregular and warrants further investigation by the user. Experimental analysis demonstrated how the accuracy of the tool did not diminish with an increasing directory size. However, the analysis did demonstrate how the computational complexity increases quadratically with an increasing directory size. This raised concern over the technique's performance on large directory structures. The produced tool (`ntfs-r.exe`) implements a solution to reduce the computational complexity by ignoring subdirectories with the same permissions as their parents. This resulted in the execution of the tool on a large directory structure within a user-friendly timescale (directory size of 34,695 in 116 seconds).

The second technique is the use of instance based learning (k-NN) to identify file system permissions that are similar to a new permission for an object that requires implementation. This technique produces a measure of similarity where a lower value indicates a better match. This score of similarity can then be used by the user to identify a user or group that already has the required permission level. This is useful as it allows the user to make new permission allocations consistent with those already in place, and subsequently helps maintain the structure of permissions allocation. Experimental analysis demonstrated that the accuracy of the technique reduces as the number of permissions that the object has within the directory structure increases. However, as `ntfs-r.exe` ignores subdirectories with the same permissions, the number of permission entries for a given object will also reduce (i.e. unnecessary duplication is ignored). In the case study, `ntfs-r.exe` calculates k-NN scores for the same directory structure discussed in the previous paragraph in 35 seconds.

The accuracy of both techniques decreases as the diversity of the file system permissions increases. This implies that the tool's suitability for systems which have been administered in an ad-hoc way is decreases. A suitable solution would be to iteratively use the technique, rectifying the irregularities at each stage, until no further irregularities are detected. However, this approach does rely on the assumption that the directory structure has irregular permissions.

A case study was performed which validated the use of the produced application on six multi-user file systems. The results from the case study clearly demonstrate the potential of the application for aiding with the analysis and administration of relative file system permissions. The average accuracy of identifying irregular permissions and the suggestion of new allocations is 0.911 and 0.805, respectively. An accuracy less than 100% during the identification of irregular permissions means that either irregular permissions are ignored or regular permissions are incorrectly identified as irregular. Less than 100% accuracy when specifying new permissions means that in some instances incorrect groups or other permission entries are identified as being suitable to achieve the desired permission level. The tool's contribution is still significant even with less than 100% accuracy. This is because a reduced level of expert knowledge is required to identify false positives, as well as vast amounts of time will have been saved by not requiring a manual exhaustive audit of the file system. Future work includes implementations for other file systems, as well as the implementation of other statistical analysis and instance-based learning techniques to further increase accuracy. Other methods of extracting, representing, and interpreting file system data are to be explored to try and find suitable ways of improving accuracy.

8. Availability

`ntfs-r.exe`, usage instructions, and source code are available from:

http://selene.hud.ac.uk/scomsp2/

In addition, all datasets used in this paper are available from the authors upon request.

9. References

Aas, K., Eikvil, L., 1999. Text categorisation: A survey. Raport NR 941.

Altman, N. S., 1992. An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician 46 (3), 175–185.

Balakrishnan, N., Voinov, V., Nikulin, M. S., 2013. Chi-squared goodness of fit tests with applications. Academic Press.

Beznosov, K., Inglesant, P., Lobo, J., Reeder, R., Zurko, M. E., 2009. Usability meets access control: challenges and research opportunities. In: Proceedings of the 14th ACM symposium on Access control models and technologies. SACMAT '09. ACM, New York, NY, USA, pp. 73–74.
URL http://doi.acm.org/10.1145/1542207.1542220

Bhuyan, M., Bhattacharyya, D., Kalita, J., First 2014. Network anomaly detection: Methods, systems and tools. Communications Surveys Tutorials, IEEE 16 (1), 303–336.

Cao, X., Iverson, L., 2006. Intentional access management: making access control usable for end-users. In: Proceedings of the second symposium on Usable privacy and security. SOUPS '06. ACM, New York, NY, USA, pp. 20–31.
URL http://doi.acm.org/10.1145/1143120.1143124

Catania, C. A., Bromberg, F., Garino, C. G., 2012. An autonomous labeling approach to support vector machines algorithms for network traffic anomaly detection. Expert Systems with Applications 39 (2), 1822 – 1829.
URL http://www.sciencedirect.com/science/article/pii/S0957417411011808

Cooke, D., Clarke, G., 1989. A basic course in statistics. Arnold.

De Capitani di Vimercati, S., Paraboschi, S., Samarati, P., 2003. Access control: principles and solutions. Software: Practice and Experience 33 (5), 397–421.
URL http://dx.doi.org/10.1002/spe.513

Givens, G. H., Hoeting, J. A., 2012. Computational statistics. Vol. 710. John Wiley & Sons.

Govindavajhala, S., Appel, A. W., 2006. Windows access control demystified. Jan 31, 1–11.

Greenwood, P. E., 1996. A guide to chi-squared testing. Vol. 280. John Wiley & Sons.

Haber, E. M., Kandogan, E., Maglio, P. P., 2011. Collaboration in system administration. Communications of the ACM 54 (1), 46–53.

Hanley, J. A., McNeil, B. J., 1982. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology 143 (1), 29–36.

Hanner, K., Hörmanseder, R., 1999. Managing windows nt file system permissions— a security tool to master the complexity of microsoft windows nt file system permissions. Journal of Network and Computer Applications 22 (2), 119 – 131.
URL http://www.sciencedirect.com/science/article/pii/S1084804599900863

Islam, M. S., Rahman, S. A., 2011. Anomaly intrusion detection system in wireless sensor networks: security threats and existing approaches. International Journal of Advanced Science and Technology 36 (1).

Jenks, G. F., 1967. The data model concept in statistical mapping. International yearbook of cartography 7 (1), 186–190.

Khan, M. N. A., 2012. Performance analysis of bayesian networks and neural networks in classification of file system activities. Computers & Security 31 (4), 391 – 401.
URL http://www.sciencedirect.com/science/article/pii/S0167404812000533

Lazarevic, A., Ertöz, L., Kumar, V., Ozgur, A., Srivastava, J., 2003. A comparative study of anomaly detection schemes in network intrusion detection. In: SDM. SIAM, pp. 25–36.

Mahoney, M. V., 2003. Network traffic anomaly detection based on packet bytes. In: Proceedings of the 2003 ACM Symposium on Applied Computing. SAC '03. ACM, New York, NY, USA, pp. 346–350.
URL http://doi.acm.org/10.1145/952532.952601

Manning, C. D., Raghavan, P., Schütze, H., 2008. Introduction to information retrieval. Vol. 1. Cambridge university press Cambridge.

Microsoft, 2006a. AccessEnum V1.32.
URL http://technet.microsoft.com/en-us/sysinternals/bb897332

Microsoft, 2006b. How to use Xcalcs.vbs to modify NTFS permissions.
URL http://support.microsoft.com/kb/825751

Naldurg, P., KR, R., 2011. Seal: a logic programming framework for specifying and verifying access control models. In: Proceedings of the 16th ACM symposium on Access control models and technologies. ACM, pp. 83–92.

Naldurg, P., Schwoon, S., Rajamani, S., Lambert, J., 2006. Netra:: seeing through access control. In: Proceedings of the fourth ACM workshop on Formal methods in security. ACM, pp. 55–66.

Nemeth, E., 2010. UNIX and Linux system administration handbook. Pearson Education.

Parkinson, S., Crampton, A., 2013. A Novel Software Tool for Analysing NT File System Permissions. International Journal of Advanced Computer Science and Applications 4 (6), 266–272.

Parkinson, S., Hardcastle, D., 2014. Automated planning for file system interaction. Proceedings of The 32nd Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG2014).

Parkinson, S., Somaraki, V., Ward, R., 2016. Auditing file system permissions using association rule mining. Expert Systems with Applications 55, 274 – 283.
URL http://www.sciencedirect.com/science/article/pii/S0957417416300586

Pawlowski, B., Shepler, S., Beame, C., Callaghan, B., Eisler, M., Noveck, D., Robinson, D., Thurlow, R., 2000. The nfs version 4 protocol. In: Proceedings of the 2nd International System Administration and Networking Conference (SANE 2000). Vol. 2. p. 50.

Russel, C., Crawford, S., Gerend, J., 2003. Microsoft windows server 2003 administrator's companion. Microsoft Press.

Russinovich, M., Solomon, D., Ionescu, A., 2012. Windows internals. Pearson Education.

Russinovich, M. E., 2012. Windows internals, part 1: Covering windows server 2008 r2 and windows 7 author: Mark e. russinovich, david a. solomon, ale.

Solomon, D. A., 2005. Microsoft windows internals: Microsoft windows server 2003, windows xp, and windows 2000.

Ten, C.-W., Hong, J., Liu, C.-C., Dec 2011. Anomaly detection for cybersecurity of the substations. Smart Grid, IEEE Transactions on 2 (4), 865–873.

Thomas, O., 2010. Are NTFS and share permissions a bit too complicated. Windows IT Pro.

Viswanath, B., Bashir, M. A., Crovella, M., Guha, S., Gummadi, K. P., Krishnamurthy, B., Mislove, A., 2014. Towards detecting anomalous user behavior in online social networks. In: Proceedings of the 23rd USENIX Security Symposium (USENIX Security).

Xie, M., Han, S., Tian, B., Parvin, S., 2011. Anomaly detection in wireless sensor networks: A survey. Journal of Network and Computer Applications 34 (4), 1302 – 1325, advanced Topics in Cloud Computing.
URL http://www.sciencedirect.com/science/article/pii/S1084804511000580

Yang, Y., Pedersen, J. O., 1997. A comparative study on feature selection in text categorization. In: ICML. Vol. 97. pp. 412–420.