Auditing file system permissions using Association Rule Mining

S. Parkinson[a,*], V. Somaraki[a], R. Ward[a]

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

Abstract

Identifying irregular file system permissions in large, multi-user systems is challenging due to the complexity of gaining structural understanding from large volumes of permission information. This challenge is exacerbated when file systems permissions are allocated in an *ad-hoc* manner when new access rights are required, and when access rights become redundant as users change job roles or terminate employment. These factors make it challenging to identify what can be classed as an irregular file system permission, as well as identifying if they are irregular and exposing a vulnerability. The current way of finding such irregularities is by performing an exhaustive audit of the permission distribution; however, this requires expert knowledge and a significant amount of time. In this paper a novel method of modelling file system permissions which can be used by association rule mining techniques to identify irregular permissions is presented. This results in the creation of object-centric model as a by-product. This technique is then implemented and tested on Microsoft's New Technology File System permissions (NTFS). Empirical observations are derived by making comparisons with expert knowledge to determine the effectiveness of the proposed technique on five diverse real-world directory structures extracted from different organisations. The results demonstrate that the technique is able to correctly identify irregularities with an average accuracy rate of 91%, minimising the reliance on expert knowledge. Experiments are also performed on synthetic directory structures which demonstrate an accuracy rate of 95% when the number of irregular permissions constitutes 1% of the total number. This is a significant contribution as it creates the possibility of identifying vulnerabilities without prior knowledge of how to file systems permissions are implemented within a directory structure.

Keywords: Access control, Auditing, Association Rule Mining.

1. Introduction

File systems are common amongst the majority of computer operating systems and from a user perspective their primary use is to securely store files. 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, but 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. Using a mixture of both types of permissions provides a flexible architecture which can accommodate a large verity of different access control levels. However, this flexible nature also provides the possibility for complex permission relationships and anomalous permissions which often go undetected.

*Corresponding author

Email addresses: s.parkinson@hud.ac.uk (S. Parkinson), v.somaraki@hud.ac.uk (V. Somaraki), r.ward@hud.ac.uk (R. Ward)

Administration of file system permissions on large file systems is both a challenging and cumbersome task as it is often difficult to conceptualise the large volumes of information available. Due to the complexities of managing permissions, 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). However, even with such guides, users are left to analyse the large amount of access control information independently. The level of their knowledge regarding access control and their system's configuration often directly determines the quality of their audit, and thus produces a heavy reliance on expert knowledge.

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. A trivial 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. The consequences of both types of vulnerabilities can be severe and can be generated from both user and software actions. For example, if a user has an elevated level of permission on a network directory structure, they could unintentionally modify or remove important data. A more severe consequence would be if the user could access data which is sensitive, as this could result in the organisation breaching the Data Protection Act. It is also possible that software (such as viruses, etc.) executing under the user's credentials could exploit their file system permissions to perform malicious activity.

Trend mining, in the form of Association Rule Mining (ARM) (Ma, 1998), has been extensively used to identifying anomalies and irregularities in many different application areas (Cheng et al., 2015). ARM is a method of automatically identifying interesting relationships amongst variables in large datasets. Interesting relationships are often those that frequently occur; however, in some applications, such as the one presented in this paper, an interesting relationship is one which occurs infrequently. There have been many successful applications of ARM for identifying interesting (both frequent and infrequent) relationships in large datasets. For example, in finance (Yu et al., 2009; Barak and Modarres, 2015), medical data (Somaraki et al., 2011, 2015), and cellular networks (Khatib et al., 2015). The use of ARM is therefore a natural selection for applying to the challenge of identifying irregularities in file system permissions.

The work presented in this paper is tailored towards Microsoft's New Technology File System (NTFS); however, it can be easily adapted to other file systems. The majority of multi-user environments in organisations will use Microsoft's NTFS for providing a distributed mechanism of file storage. There are many complexities associated with administrating and auditing file system permissions on Microsoft's NTFS which can result in the creation of vulnerabilities. The complexity of identifying these vulnerabilities has resulted in a unhealthy reliance on expert knowledge. The work presented in this paper helps to remove this unhealthy balance on expert knowledge and provide a method of auditing file system permissions for all NTFS users. This paper first provides a review into related work in the area of aiding auditing of file system permissions. The next section the provides a description of NTFS access control structure, highlighting auditing complexities. At the end of this section, detail is provides on how file system permissions are modelled in the work presented in this paper, including the use of an algorithm to combine permissions to determine the effective permission. This then leads to a description of association rule mining techniques which are useful for identifying irregularities in file system permissions. In this section the chosen technique is also discussed and justified. Empirical observations are then provided where the developed technique is tested on five diverse real-world file systems. In this empirical observation, the results are compared to those of a domain expert.

2. Related work

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.

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). 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 artefacts 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. Another area that has recently received significant interest in the access control community is that of access control and anomaly detection for cloud computing. For example, recent work by Hu et al. (2013) both motivates the need for identifying anomalies in web access control policies and provides a potential solution through the use of a binary decision diagram for anomaly discovery and resolution.

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. Literature suggests that there has been no work in the detection of irregularities in file system security without the need to statically define what constitutes an

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.

irregular permission. This includes the potential application of Association Rule Mining, This is surprising considering the vast user-base and the potential benefits that can be achieved.

3. NTFS Implementation and Modelling

In this section, a description of how NTFS permissions are implemented is provided. Alongside this description is a discussion of how its implementation can results in complexities which ultimately can result in vulnerabilities. Following this, a description is provided of how NTFS permissions are translated into a relational object-model.

3.1. 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 ACE within the NTFS is a set of attributes, p, from the predefined set of attributes $p \subseteq P$, $P = \{a_1, \ldots, a_n\}$. 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. The NTFS also allow for the creation of special fine-grained permissions that are constructed from a combination of fourteen permission attributes (Russel et al., 2003).

The potential to create a special permission is a useful feature; however, it can introduce complexity as it requires detailed knowledge regarding the authority that each attribute holds (Thomas, 2010). A example of where a special permission could be used is if the user wanted to assign a user or group the standard access level of `Modify` for all the contents of a shared directory. However, this has adverse consequences as the `Modify` permissions will allow the user(s) to be able to delete the folder itself. A potential solution here is to assign the user or group the default permissions level of `Modify`, and then modify the permissions' attributes so that only subfolders and files can be deleted, and not the folder itself. This modification would result in the creation of a special permission.

3.2. Propagation and Inheritance

In this section, a discussion is provided on the different mechanisms through which NTFS permissions can propagate through a directory structure. There are two types of ACE entires within the DACL; (1) Explicit and (2) Inherited. Explicit entires are those that are applied directly to a DACL, whereas inherited are propagated from the directory's parent. The type property of the ACE allows programmatic determination of whether the permission has been explicitly assigned to a directory or if it was inherited from a parent directory. Furthermore, it is also possible to create custom fine-grained inheritance rules. Such special inheritance rules can easily be overlooked during administration and auditing, resulting in theunintended propagation of access.

3.3. Accumulation

Permissions accumulation creates the potential for an interacting subject to receive permissions from multiple different policies. Any interacting subject within the NTFS can be assigned to access control groups. This means that they do not have to be explicitly added in the ACE, they could be an associated group member. Such policy combination is controlled by the Local Security Authority Subsystem Service (LSASS). The high-level functionality of this service is to create an ordered union all relevant policies. However, there are a few complexities that can occur in this combination process. These are: (1) explicit permissions take priority over those inherited, (2) explicit deny permissions always take priority over any other assigned permission, and (3) permissions inherited from closer relatives take priority over those further away.

It would be logical to assume that deny permissions would always take precedence over apply permissions to ensure that the user operates at the least possible level of access. However, the first point makes it possible for an inherited deny permission to never be reached. This goes against a fundamental aspect of policy combination that a deny permission should never be ignored. If a situation where a user is able to ignore a deny permission was to arise, a system administrator would need to be aware of this so that they could take rectifying action. In addition to explicit permissions taking priority over inherited permissions, inherited permissions closer to the derrived directory will take priority over those more distance. For example, a folder's inherited permissions will take priority over those inherited from their grandparent.

To summarise, the ordered hierarchy for policy combination is:

1. Explicit deny.
2. Explicit allow.
3. Inherited deny.
4. Inherited allow.

3.4. Group Membership

A powerful feature of NTFS access control is that of group membership. An interacting subject (group, user, or process) that interacts with the file system can hold membership to another group. This creates the possibility for permissions to be inherited from all of the associated groups. Users will often be grouped together to mange management easier. This separation of duty can often be seen in many organisations. For example, users within a finance department will have different permissions that of management. As Hanner, et. al. (Hanner and Hörmanseder, 1999) identifies, understanding effective file permissions can become increasingly more complex by group association. This is because to evaluate a user's effective permission would require the knowledge of which groups that they are inheriting from. It should be noted here that it is not a direct mechanism of how NTFS access control is implemented, rather it results from how Microsoft allows for users, groups and processes to be managed and controlled through group memberships.

Algorithm 1: Depth-first recursive permission extraction algorithm, returning an ordered list of effective permissions for each object within the directory structure.

```
Input: Initial directory d
Output: Set of ordered effective permissions E = e_1, e_2, ..., e_n where e_n = {d, o, p}. Here d is the
        directory resource, o is the object, and p is the permission level, p = {a_1, a_2, ..., a_n}
Output: Membership relation set, G = {g_1, g_2, ..., g_n} where g_n = {o_1, o_2, ..., o_n}
Algorithm algo()
    P ← proc(d)
    return

Procedure proc(directory d)
    pACL ← d(ACL)
    foreach subdirectory c of d do
    cACL ← c(ACL)
    if cACL != pACL then
        ex = ∅, in = ∅, r = memberships(G)
        foreach ACE a in cACL do
        foreach Membership g in r do
        if isExplicitDeny(a) then
            E ← (c, getObj(a), getPerm(a))
            break
        else
            else if isExplicitAllow(a) then
                E ← (c, getObj(a), getPerm(a))
                break
            else if isInherited(a) then
                if isInheritedDeny(a) then
                    E ← (c, getObj(a), getPerm(a))
                    break
                else if isExplicitAllow(a) then
                    E ← (c, getObj(a), getPerm(a))
                    break
        end

    end
```

3.5. Object-centric Modelling

In order to represent the accumulated permission that each object holds on a file system resource, is it necessary to adopt an object-centred approach to permissions modelling. In this section, the description of NTFS's access control structure is translated into an object-centred model suitable representation which can be used for association rule mining.

Here each interacting object (user, group, etc.), o, holds a combination of permission attributes which are accumulated based on (1) the assigned permissions, (2) propagated and inherited permissions, and (3) group membership. Algorithm 1 has been developed to iterate over all the directories and create a set of effective object permissions for each directory. Each effective permission for an object is the set of permission attributes, p, that o holds on a directory, d. As seen in Algorithm 1, the hierarchy for permission accumulation (Section 3.3) is taken into consideration. The algorithm also ignores permissions which do not

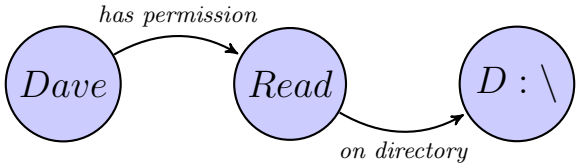

Figure 1: Object-based diagram for the permission entry of $D:\backslash, Dave, FileReadData$

differ from those of their parent directory. This helps to remove repetitive information and provides a useful heuristic for increasing performance and reducing the volume of captured data. The data structure returned from executing the algorithm is a list ordered by the directory structure. This list, $E = \{e_1, e_2, \ldots, e_n\}$, contains an ordered list of tuples, $e_n = \{o, p, d\}$, where d is the directory resource object, o is the interacting object, and p is the set of permission level objects, $p = \{a_1, a_2, \ldots, a_n\}$.

The returned list (E) contains the object model for each files system entry. For example, consider a directory where a user Dave, $o = Dave$, has permission of $FileReadData$, $p = FileReadData$, on a locally mounted directory, $d = D:\backslash$. A diagrammatic illustration of this model is provided in Figure 1. This object model is then stored in Comma Separated Value form for future processing. In the provided example, the following data set would be output:

$$Dave, FileReadData, D:\backslash$$

The example provided is simplistic and only represents one individual file system permission. It is often the case that an interacting object would accumulate access permission from multiple sources (explicit, inherited, etc.), and therefore Algorithm 1 is used to calculate each objects' effective permission (i.e the permission the user actually receive) before the creation of the object model. The output object model (stored in text form) is then a complete representation of each interacting objects' effective permission on the specified directory structure.

4. Association Rule Mining

In this section, the area of Association Rule Mining (ARM) is discussed, and details of how it is used for detecting irregularities in file system permissions are provided. ARM is a method of discovering associations within data that frequently occur. For example, in the perspective of file system permissions analysis, determining that a user (e.g *Dave*) frequently has the same level of permissions (e.g *Read*) on a wide array of directories. In ARM, I is the set of binary items, and D are the data sets. In applying ARM to file system permissions, I is the total object set (users, permissions, directories) and D (previously defined as E) is the total set of effective permissions, $D = \{e_1, e_2, e_n\}$. Each permission entry e_n is a subset of I. For example, a potential item set could be $I = \{Dave, Mike, FileRead, FileWrite, D:\backslash\}$ and the following are example data sets:

$$E_1 = \{Dave, FileRead, D:\backslash\}$$
$$E_2 = \{Mike, FileRead, D:\backslash\}$$
$$E_3 = \{Mike, FileWrite, D:\backslash\}$$

Association rule mining is a way to discover interesting relationships in datasets, or in the case of this paper, infrequent relationships. The three following rules are used in association rule mining:

Definition 1 (Association rule). *$X \implies Y$ is an Association Rule (AR), if X and Y are item sets, X is the LHS (Left Handside) or body of the rule, and Y is RHS (Right Handside) or head of rule. A set of*

data entries, can be informally define an Association Rule (AR) as the rule of $X \implies Y$*, where* $X, Y \subset I$*. In the continuing example, a file system specific association rule would be* $\{FileRead\} \implies \{D : \backslash\}$*.*

Definition 2 (Support of an AR). *The support (supp*(X)*) of an AR is defined as the percentage of permission entries that contain both* X *and* Y*. It can also be defined as the probability* $P\big(X \cap Y\big)$*. In the example, the dataset* $\{FileRead, D : \backslash\}$ *has support of 0.6 since is occurs in one two thirds of the permission entries.*

Definition 3 (Confidence of an AR). *The confidence of an AR is defined as the ratio between the number of transactions that contain* $X \cup Y$ *and the number of transactions that contain* X*. It can also be defined as* $P\big(X \cup Y|X\big) = P\big(X \cup Y\big)/P\big(X\big)$*. In the example, the confidence (supp*$(X \subset Y)/supp(X)$*) would equal* $0.6/0.6 = 1$*, meaning that for 100% of the permission entries that have* $FileRead$ *are related to* $D : \backslash$

ARM procedures contain two stages: (i) the identification of frequent datasets, and (ii) generation of ARs. Piatetsky-Shapiro (1991) defines ARM as a method for the description, analysis and presentation of ARs which are discovered in databases using different measures to determine interesting data. ARM is concerned with the discovery, in tabular databases, of rules that satisfy defined threshold requirements. Of these requirements, the most fundamental is concerned with the support (frequency) of the item sets used to make up the ARs: a rule is applicable only if the relationship occurs sufficiently often in the data. Whether an item set is relevant or not, is determined by its support count confidence value. An item set is deemed to be relevant if its support count is above a user specified support threshold. Similarly, an AR is deemed relevant if the confidence value is above a user specified confidence threshold (Singer and Willett, 2003). Although minimum support and confidence thresholds help remove the generation of uninteresting rules, many of the remaining rules are still not interesting to the users. This work focuses on irregularities which in the context of Association Rule Mining can be assumed to be a non-frequent set of items. Therefore, in this work an interesting rule (I.e an irregular file system permission) is one with both low support and confidence.

Frequent data set mining is thoroughly studied by many researchers, but important rare items are often not discovered by these algorithms. In many cases, the contradictions or exceptions also offers useful associations. In the recent past researchers started to focus on the discovery of such kind of associations called rare associations. Rare data sets can be obtained by setting low support; however, this generates huge number of rules. Therefore the key idea is to inverse the support threshold criterion by using a maximum support threshold but also using the confidence as a second criterion to deal with the large amounts of rare relations between items. In the technique presented in this paper, the selection of the minimum support threshold value is based on the frequency of each item in the data set. E.g. how many times an item appears which is turned into a percentage and represents the threshold value. As regards to the confidence threshold, it is selected empirically and usually its value is selected to be equal to the minimum support threshold.

SOMA (Somaraki et al., 2010) is a trend-mining framework for knowledge discovery from large databases, the development of a validation framework for trend mining. SOMA was originally developed for the application of trend mining in medical data; however, SOMA itself is not domain dependent and can therefore be used with any type of data in different application areas. The framework exploits three steps: pre-processing, association rule mining, and trend mining; however, given the nature of the data and the aim of the investigation, both preprocessing and trend mining stages are not required. The preprocessing stage is superfluous in this application as the data is complete and discrete and there are no temporal requirements. In SOMA's original application of medical data, logic was utilised to aid with merging and time-stamping data subsets for analysis. Trend mining is also not required. For this application the most important phase is association rule mining, where rules are extracted by considering a the of variables extracted from the data.

In this paper, the Matrix (Yuan and Huang, 2005) algorithm which is part of SOMA framework was used for association rule mining. The algorithm is more efficient than the well-known Apriori (Tsai and Chen, 2004) algorithm as it only requires one pass over the data to generate the matrix. The algorithm is called a matrix algorithm as it creates a binary matrix with entries (0, 1) from passing over the database, resulting in the creation of a set of candidate items from which association rules are produced. In the work

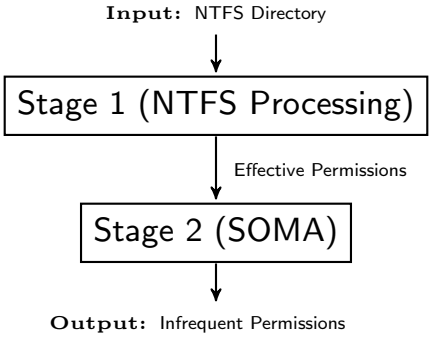

Figure 2: Schematic illustrating the integration of both the NTFS permissions extraction software and SOMA for association rule mining.

presented in this paper, the file system permission data is passed to the matrix algorithm in tabular form for processing and the output is a set of association rules, generated based on the support threshold and confidence value. The output is then processed to identify association rules for infrequent data. Full details of the matrix algorithm can be found in Yuan and Huang (2005).

There is an absence of literature detailing the use of ARM to detect anomalies in file systems; however, closely related research on the identification of anomalies and irregular data sets using ARM strongly motivate its use (Chandola et al., 2009). For example, Li et al. (2009) present the successful use of ARM to detect anomalies in identifying irregular behaviour in local area network traffic. In addition, intrusion detection is another successful area of research which has seen applications of ARM (Patcha and Park, 2007).

5. Produced Software

Software has been developed for the extraction, processing and association rule mining of file system permissions. For the extraction and accumulation of file system permissions (stage one), software 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. The output from the tool contains the ACL information for each directory in comma value separated form which is subsequently used for association rule mining. In the second stage, the SOMA framework was developed in MATLAB 20014a. In the experiments presented in this paper, both stages were executed on a Intel i7 Processor with a clock speed of 2.2 GHz. Figure 2 provides an overview illustration of the integration of the two stages. This integration of these two stages allows the user to extract NTFS permissions and then identify irregularities through trend mining.

6. Empirical Observations

This section contains two types of empirical observations through using the tools developed in Section 5. The first is empirical observations based on synthetically generated file system data, and the second is from analysing five real-world, multi-user directory structures of different configurations. The aim of this experiment is to determine the effectiveness of the proposed technique at identifying irregular and anomalous permissions within the directory structures. 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 irregular permissions incorrectly classified as regular; Finally, the *accuracy* is reported as the fraction of all samples correctly identified.

Algorithm 2: Algorithm for generated synthetic directory structures

Input: The maximum number of directories to be created, $MaxDir$
Input: The step size of the increasing number of directories, $StepDir$
Input: The maximum number of anomalies to be created, $MaxAnom$
Input: The step size of increasing the number of anomalies, $StepAnom$
Output: A set of directories, $S = \{s_1, s_2, ..., s_n\}$, where $s_n = \{d, p\}$ where d is the directory resource, and p is the permission level

```
1  Algorithm algo()
2  | for n ← 0 to MaxAnom do
3  | |  i = 0
4  | |  while i ≤ MaxDir do
5  | | |  S[j] ← createDirectory()
6  | | |  i+ = StepSize
7  | |  end
8  | |  j = 0, i = 0
9  | |  while J ≤ MaxAnom do
10 | | |  dirNo = genRandomInt(0, i)
11 | | |  pLevel = genRandomInt(1, 14)
12 | | |  i+ = StepSize
13 | | |  S[j] ← pLevel
14 | |  end
15 | |  n+ = StepAnom
16 | end
17 | return S
```

6.1. Synthetic Data sets

The motivation behind using synthetically generated test data is that although acquiring security information from real-world directory structures is easily achievable, it is difficult to acquire ground-truth knowledge regarding the irregularities within those systems. This is because the identification of irregular permissions is ultimately down to the auditor's knowledge, and although evaluating the system against such data is useful, it potentially limits the quality of assessment as both correct and incorrect permissions might have been incorrectly identified by the expert.

In order to generate synthetic file system data, a technique has been created to automatically generate directory structures, assign permissions, and insert irregular permissions (anomalies). From the created directory structures, it is then possible to use the proposed NTFS processing software to extract access control information, and then use SOMA to identify irregularities. Algorithm 2 describes the process to generate synthetic directory structures. The first aspect of the algorithm is to create a the directory structures by providing a maximum number of directories and a stepsize. For example, providing a maximum directory size ($MaxDir$) of 50 and a stepsize ($StepDir$) of 10 would result in the generation of five directory structures (10, 20, 30, 40, and 50). In the creation of directories, the security permission of their parent and system defaults will automatically be inherited. In addition, the same directory structure is created multiple times, from 1 until $MaxAnom$ in the step size of $StepAnom$. For example, if $MaxAnom = 20$ and $StepAnom = 5$, the number of the same directory structure created would be four with 5, 10, 15, and 20 anomalies. Creating the anomalies for each directory structure is performed by identifying a random number between 0 and and the current directory size (i). This random number ($dirNo$) will be used to assign an anomaly. The permission of the anomaly is determined by generating a random number between 1 and 14 (Table 1), $pLevel$. The number identified represents the number of permission attribute set. For example, if the generated number is 2, the first two permission attributes are set.

In the synthetic experiments presented in this paper, $MaxDir = 1,000$, $StepDir = 100$, $MaxAnom = 64$, and for each iteration is $StepAnom = StepAnom \times 2$. I.e the anomalies would increase in the sequence

User Group	Directory	Attribute	Support	Confidence
Administrators	100\100_16	Delete	0.05	0.27
Anomalyuser	100\100_16\0	Delete	0.001	0.01

Table 2: Example Association Rules generated when analysing synthetically produced file systems that has 100 directory and 16 anomalies.

of 1, 2, 4, 8, 16, 32, 64. Doubling the number of anomalies in each iteration will allow for the comprehensive understanding of the system's ability to recognise anomalies as their frequency increases. The random approach of assigning permissions, as well as the permission level, helps to simulate a file system that is becoming increasingly complex and helps to replicate the process of 'ad-hoc' permissions allocation. This results in the production of 70 different directory structures to test the proposed technique.

Figure 6 illustrates the average execution time for analysing each directory size. The difference in execution time for directory structures of the same size but with a different number of irregular permissions is less than 1 second. From the Figure 6 it is noticeable that execution time increases by around 50 seconds until the directory size reaches 800 where the time increases by around 100 seconds for the addition of 100 directories. It is also noticeable that the time required for the associative rule mining stage makes up a larger portion of the duration than the permission extraction stage.

The results from all 10 directory structures, and the 7 different permission levels, are demonstrated in three different graphs illustrates the Receiver Operator Characteristics (ROC) space (Hanley and McNeil, 1982). Figure 3 illustrates the *tpr* and the *fpr* for the first four directory structures, with a directory size of 100, 200, 300, and 400, respectively. Figure 4 illustrates the results for the directory structures with 500, 600 and 700 directories. Finally Figure 5 illustrates the results for the directory sizes of 800, 900, and 1000. These results are interesting as they demonstrate both an improvement in *tpr* and a decrease in *fpr* as the directory size increases. This reason behind this is that the proportion of anomalies in the directory structure containing 100 directories is greater than that of the directory with 200 directories. For example, the directory structure with 100 directories containing 32 anomalies consists of a total of 1008 permissions. The directory structure containing 200 directories and 32 anomalies anomalies consists of 1936 total permissions. This is interesting as it demonstrates that the tools ability improves with an increasing number of directories and a decreasing number of irregular permissions. In the first instance this might appear like a negative characteristic of the system; however, as real-world directory structures will often be well in excess of 200 directories, with a low proportion of anomalies, it can be seen that the tools performance would be suitable for application. It is, however, worth raising the point that the accuracy would reduce in directory structures where permissions have been managed in an ad-hoc manner for a prolonged period of time. However, considering the fact that a large portion of permissions will always be system generated, the technique should still be able to identify anomalies, even if accuracy is reduced.

The verbose results from all 70 experiments are omitted from the paper due to space constraints. However, the results and data sets are available from the author upon request. In addition, from the graphs it is not possible to identify the number of irregular permissions each directory structure has; however, from analysing the results it can be stated that the directory structure with fewest regular permissions has the highest *tpr* an the lowest *tnr* across all directory sizes. Another observation that is consistent for all directory sizes is that the *tpr* increases and the *tnr* decreases incrementally with the increasing number of irregular permissions.

Table 2 demonstrates two example Associate Rules for demonstrating both an irregular and regular permission extracted from a synthetically generated directory structure with 100 directories and 16 anomalies. The first rule is for a frequently occurring permission, and the second rule is for a irregular permission allocation. The first rule ($X \implies Y$) demonstrates that the *Administrators* group implies *Delete*, $X = \{Administrators\} \implies Y = \{Delete\}$. This rule is frequently occurring because the percentage (support) of all entries that contain both X and Y is at 5 %. This is high considering that there are in excess of five other users and groups within the system, and fourteen different attributes used in total. The confidence value is also quite high (27 %) which indicates that there is a strong relationship between both *Administrators* and *Delete*. The second rule demonstrates an irregular permission between

Directory Number	Directories	Permissions levels	Permission Entries
1	36	3	248
2	108	3	1142
3	427	2	1708
4	453	4	3184
5	407	4	3719

Table 3: Test NTFS directory structures specifics. The numbers are only considering directories with permissions different from their parent. I.e. not inherited.

the *Anomalyuser* and *Delete* permission. Here the support is less 0.1 % indicating that a low number of permissions contain both X and Y. In addition, the confidence value is also low at 1 % indicating that a low number of the total *Delete* permissions are related to *Anomalyuser*. From this example, it is easy to determine how the second permission has been identified as an anomaly.

In the results presented in this section, the average accuracy rate for each dataset is incrementally improving from 0.85 to 0.95 for the directory structure with 100 and 1000 directories, respectively. This further stimulates the assumption that the system performs best when the number of anomalies represents 1% or less than the total number of permissions (100 irregular and 9136 regular). However, even when the number of irregular permissions is around 10%, the accuracy rate is around 85%. This is interesting as it demonstrates the suitability of the technique for identifying file system permission regularity, which increases as the percentage of irregular permissions decreases. This is contradictory to the main aim of this work which is trying to eliminate the requirement for expert knowledge when auditing file system permissions. This is because as the complexity of a file system increases through more ad-hoc permissions, the tool's ability decreases and the requirement for expert knowledge increases. However, this tool does make progress on reducing the need for expert knowledge by making considerable progress towards removing the requirement for an exhaustive manual audit.

6.2. Real-world Data sets

In this section the proposed system is validated against five real-world file systems to identify irregular permissions. As this test is not synthetic, the ground truth is acquired by expert knowledge. The limitation of acquiring expert knowledge alongside access to real-world directory structures limits the number of available directory structures for analysis.

Table 3 shows details of the five previously unseen NTFS directory structures that are used to establish the developed techniques' performance. The diversity of different organisations permissions varies widely; some have good access control which is maintained through adhering to a rigorous structure, whereas some have inconsistent access control resulting from ad-hoc fixes. The number of directories presented in Table 3 is significantly lower than the total number of directories in the file system. This is because the algorithm considers directories which have permission different from their parent. For example, directory number 1 in Table 3 actually contains 2856 directories of which only 55 are are unique. I.e these 55 permissions are inherited to all their sub-directories. For each test case, the ground truth (I.e the correct identification of irregular permissions) is acquired through reports produced by an independent security auditing organisation.

Figure 7 illustrates the computation times for each of the directory structures presented in Table 3. From this graph, it is noticeable that the execution time for both stages of the algorithm increase as the size and complexity of the directory structure increase. However, the results demonstrate that the duration of the extraction and processing stage is more sensitive to an increase in both directory size and the number of assigned permissions, whereas stage two (association rule mining) is less affected by an increasing number of allocated permissions. For example, in the results it can be seen that the execution time of both stages increases greater than exponentially until directory structure 4. Interestingly there is a large difference between the number of permission entries for directory structures 3 and 4 (1708 to 3184) which results in a

Directory Number	**Irregular**	**Regular**	**True positive** (*tpr*)	**False Positive** (*fpr*)	**True Negative** (*tnr*)	**False Negative** (*fnr*)	*accuracy*
1	6	248	5 (*0.83*)	1 (*0.17*)	203 (*0.82*)	45 (*0.18*)	*0.82*
2	62	1142	50 (*0.81*)	12 (*0.19*)	977 (*0.86*)	165 (*0.14*)	*0.85*
3	24	1708	20 (*0.83*)	4 (*0.17*)	1538 (*0.90*)	170 (*0.10*)	*0.90*
4	14	3184	13 (*0.93*)	1 (*0.07*)	3037 (*0.95*)	147 (*0.04*)	*0.97*
5	144	3719	143 (*0.97*)	4 (*0.03*)	3619 (*0.97*)	100 (*0.03*)	*0.99*

Table 4: Accuracy results from empirical analysis. See paragraph 1 in Section 6 for a description of the measures.

large increase in computation time for stage one; however, the computation time for stage two only changes a small amount (less than one second). This demonstrates good scalability of SOMA (stage 2) for analysing file system permissions.

Following analysis of the performance, it is important to evaluate the ability to correctly identify both irregular and regular permissions. This is performed by comparing the results of processing permissions with SOMA with the results of expert analysis. Figure 8 illustrates the Receiver Operator Characteristics (ROC) space (Hanley and McNeil, 1982) for all five directory structures. The ROC graph demonstrates that although accuracy is lower on the smaller directory structures (directory structure 1 and 2), it is improving as both the directory size and complexity increase. For example, for directory structure 1 has a *tpr* of 0.83, a *fpr* of 0.17, and an *accuracy* of 0.82. Directory structure 5 has a *tpr* of 0.97, a *fpr* of 0.03, and an *accuracy* of 0.99 (full results can be seen in Table 4). Although directory structure 1 demonstrates a reasonable performance where 80 % of classifications are correct, it would still require a reliance on expert interpretation to improve confidence. Directory structures two and three are slightly better where the accuracy rate has increased to 0.85 and 0.90, respectively. It is interesting to discover that directory structures four and five have improved and have accuracy levels of 0.97 and 0.99, respectively. This is a significant finding as it suggests that the accuracy of the proposed techniques are improving as both the directory size and complexity are increasing. This would suggest that the directory structures one, two and three are not big enough to allow the association mining techniques to correctly distinguish between what is normal and irregular.

The empirical results presented above demonstrate the potential of the proposed technique to programmatically identify irregular permissions with a minimal reliance on expert knowledge. These results have demonstrated that the proposed technique is sensitive to directory size, and in general performs better on larger directory structures with diverse permissions.

7. Conclusion

This paper presents a technique to identify irregular file system permissions using Association Rule Mining. This research focuses on the vulnerabilities of Microsoft's New Technology File System (NTFS). Motivation for applying the presented technique to the NTFS is justified through a details discussion on the complexities associated with NTFS permissions administration. A detailed discussion is then provided where the structure of NTFS permissions is discussed and modelled in a suitable way to develop algorithms to autonomously identify irregular file system permissions.

The developed technique is a two-stage algorithm. The first stage is a permission extraction algorithm for acquiring all relevant information for analysis. Empirical analysis has demonstrated that this stage is sensitive to both the size of the directory structure as well as the complexity of the assigned permissions. However, given that analysis is performed off-line and that permissions are often not changing quickly, the time-scale that is presented (a maximum of just over 2 minutes for the largest directory) is not detrimental. The second stage is to use association rule mining to identify permissions which are irregular and can potentially result in vulnerabilities. Using association rule mining results in the production of a final rule set containing those which have low support and confidence. This stage of the produced system scales well and has an accuracy rate which increases with both directory size and complexity.

Empirical observations are then performed which included executing the produced technique against 70 synthetically generated and five real-world, multi-user directory structures. The synthetically generated directory structures created the possibility to rigorously evaluate the system's performance and understand the ability to identify irregular permissions which have been created in an 'ad-hoc' manner. The results from the synthetic directory structures demonstrated that the presented technique has an 90 % average accuracy. The main finding from these experiments was that the ability to correctly identify irregular permissions increases and the incorrect identification of regular permissions decreases as the directory size increases and the number of anomalies stays the same or decreases. It was identified that 95 % accuracy can be achieved when the percentage of irregular permissions is equal to or less than 1 % of the total permissions.

The real-world directory structures were different in terms of directory size, the number of assigned permissions, and the number of known irregular permissions. The irregular permissions have been predetermined through the use of expert knowledge. This expert knowledge was used as ground truth for the empirical analysis. The results are promising with an average accuracy rate of 91 %. It is interesting to discover that the technique has a better accuracy on larger and more complex directory structures.

Although an accuracy rate in excess of 90 % has been achieved, it still does not completely remove the requirement for expert knowledge. However, it significantly reduces the reliance on expert knowledge as well as the time consuming process of having to perform an exhaustive manual security audit of a file system. The contribution from this paper to the anomaly detection and file system auditing communities is significant and strongly motivates further research. All the data sets and software presented in this paper are available form the corresponding author upon request. Future research is to further develop the techniques to include different mechanisms of file system access control as well as identifying suitable mechanisms of further improving accuracy.

8. References

Barak, S., Modarres, M., 2015. Developing an approach to evaluate stocks by forecasting effective features with data mining methods. Expert Systems with Applications 42 (3), 1325 – 1339.

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.

Chandola, V., Banerjee, A., Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR) 41 (3), 15.

Cheng, Q., Lu, X., Liu, Z., Huang, J., 2015. Mining research trends with anomaly detection models: the case of social computing research. Scientometrics 103 (2), 453–469.

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

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.

Hu, H., Ahn, G.-J., Kulkarni, K., Nov 2013. Discovery and resolution of anomalies in web access control policies. Dependable and Secure Computing, IEEE Transactions on 10 (6), 341–354.

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).

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.

Khatib, E. J., Barco, R., Gmez-Andrades, A., Muoz, P., Serrano, I., 2015. Data mining for fuzzy diagnosis systems in LTE networks. Expert Systems with Applications 42 (21), 7549 – 7559.

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.

Li, X., Zhang, Y., Li, X., Sept 2009. Local area network anomaly detection using association rules mining. In: Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on. pp. 1–5.

Ma, B. L. W. H. Y., 1998. Integrating classification and association rule mining. In: Proceedings of the fourth international conference on knowledge discovery and data mining.

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.

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).

Patcha, A., Park, J.-M., 2007. An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks 51 (12), 3448 – 3470.
URL http://www.sciencedirect.com/science/article/pii/S138912860700062X

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.

Piatetsky-Shapiro, G., 1991. Discovery, analysis and presentation of strong rules. Knowledge discovery in databases, 229–238.

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

Singer, J. D., Willett, J. B., 2003. Applied longitudinal data analysis: Modeling change and event occurrence. Oxford university press.

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

Somaraki, V., Broadbent, D., Coenen, F., Harding, S., 2010. Finding temporal patterns in noisy longitudinal data: a study in diabetic retinopathy. Advances in Data Mining. Applications and Theoretical Aspects, 418–431.

Somaraki, V., Harding, S., Broadbent, D., Coenen, F., 2011. Soma: A proposed framework for trend mining in large uk diabetic retinopathy temporal databases. Research and Development in Intelligent Systems XXVII, 285–290.

Somaraki, V., Vallati, M., McCluskey, L., 2015. Discovering interesting trends in real medical data: A study in diabetic retinopathy. Proceedings of the 17th Portuguese Conference on Artificial Intelligence. EPIA 2015.

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.

Tsai, P. S., Chen, C.-M., 2004. Mining interesting association rules from customer databases and transaction databases. Information Systems 29 (8), 685–696.

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.

Yu, L., Chen, H., Wang, S., Lai, K. K., 2009. Evolving least squares support vector machines for stock market trend mining. Evolutionary Computation, IEEE Transactions on 13 (1), 87–102.

Yuan, Y., Huang, T., 2005. A matrix algorithm for mining association rules. In: Advances in Intelligent Computing. pp. 370–379.

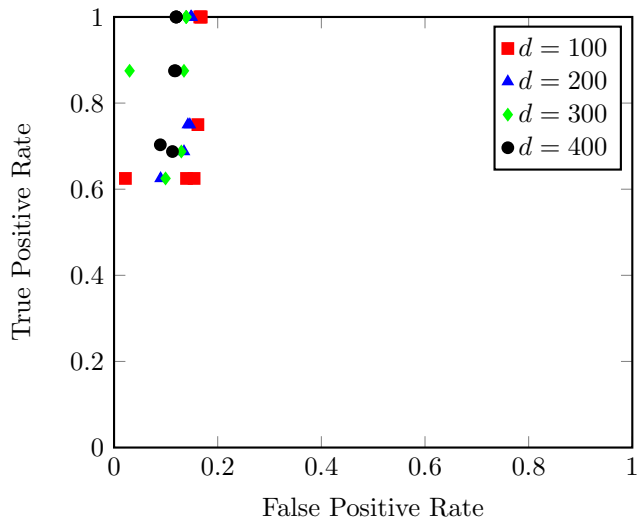

Figure 3: ROC space demonstrating the fpr and tpr of identifying anomalies in the synthetic directory structures where d is the number of directories.

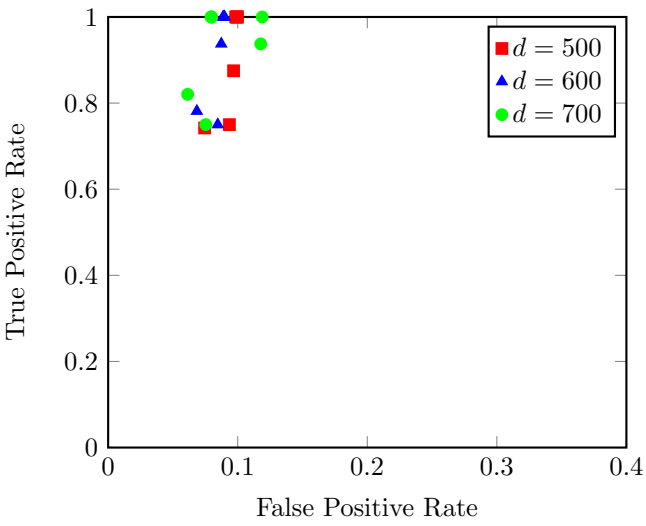

Figure 4: ROC space demonstrating the fpr and tpr of identifying anomalies in the synthetic directory structures where d is the number of directories.

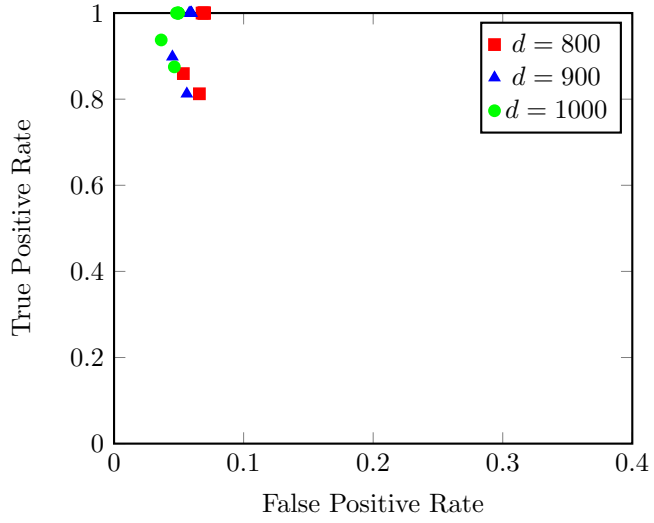

Figure 5: ROC space demonstrating the fpr and tpr of identifying anomalies in the synthetic directory structures where d is the number of directories.

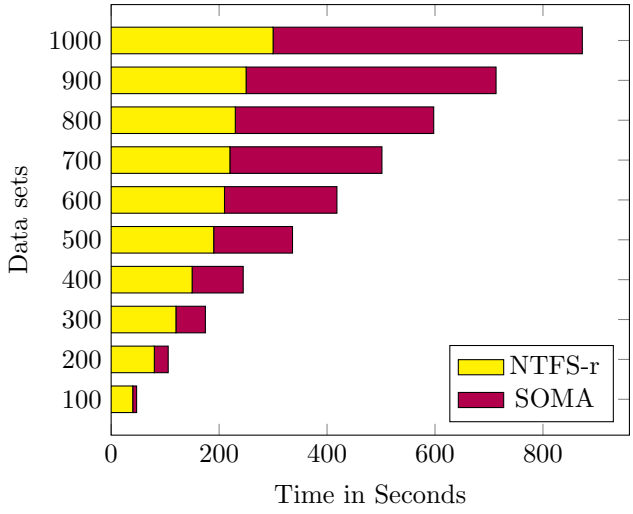

Figure 6: Computation time in seconds for (1) extracting and processing NTFS permissions, followed by (2) the execution of the trend mining (SOMA) algorithm to identify irregular permissions.

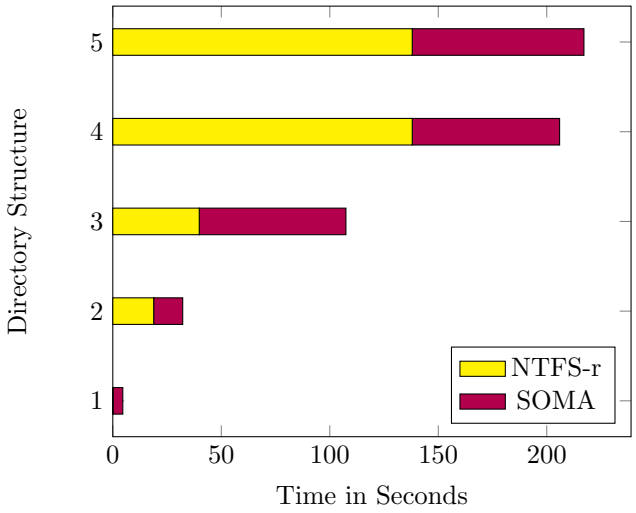

Figure 7: Computation time in seconds for (1) extracting and processing NTFS permissions, followed by (2) the execution of the trend mining (SOMA) algorithm to identify irregular permissions.

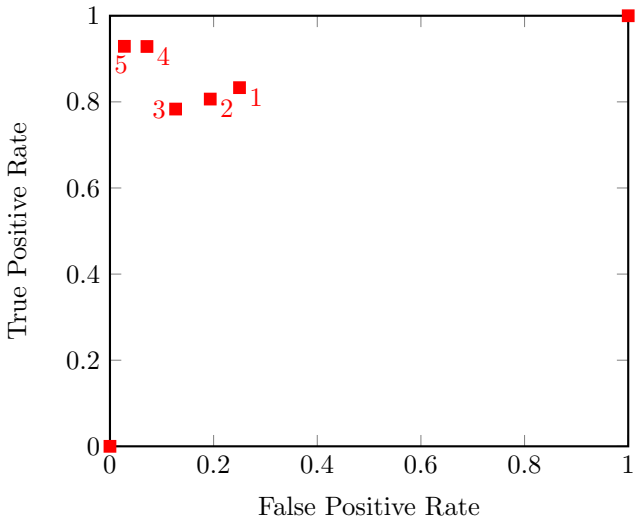

Figure 8: ROC curve for the analysis of the test file systems