


RESEARCH

Open Access

# *Creeper*: a tool for detecting permission creep in file system access controls



Simon Parkinson\* , Saad Khan, James Bray and Daiyaan Shreef

## Abstract

Access control mechanisms are widely used in multi-user IT systems where it is necessary to restrict access to computing resources. This is certainly true of file systems whereby information needs to be protected against unintended access. User permissions often evolve over time, and changes are often made in an ad hoc manner and do not follow any rigorous process. This is largely due to the fact that the structure of the implemented permissions are often determined by experts during initial system configuration and documentation is rarely created. Furthermore, permissions are often not audited due to the volume of information, the requirement of expert knowledge, and the time required to perform manual analysis. This paper presents a novel, unsupervised technique whereby a statistical analysis technique is developed and applied to detect instances of permission creep. The system (herein referred to as *Creeper*) has initially been developed for Microsoft systems; however, it is easily extensible and can be applied to other access control systems. Experimental analysis has demonstrated good performance and applicability on synthetic file system permissions with an average accuracy of 96%. Empirical analysis is subsequently performed on five real-world systems where an average accuracy of 98% is established.

**Keywords:** Permission creep, Access control, Auditing,  $\chi^2$  statistics

## Introduction

File systems are integral part 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) and fine-grained permissions are customised permissions created from a set of predefined attributes to represent highly customised access control policies.

Many organisations will implement and maintain access control systems in respect to the different jobs roles that

their staff undertake. Although role-based access control systems do exist (Sandhu et al. 1996), many implementations are built by using group membership allocations and discretionary access control models. A discretionary access control model allows for a subject to be capable of passing on permission to other subjects. More specifically is its use of groups to pass on permissions to other groups and users. This paper focuses on the discretionary model, largely due to its vast use in real-world systems; however, the techniques developed in this paper are based on the effective permission of each user. This means that the techniques can be extended and used to analyse different access control implementations.

Employees within organisations often change job role as their career progresses. During a change of job role, it is usual for ad hoc permissions changes (both additions and removal) to reflect the new required level. In a similar scenario, where a user is assigned a temporary organisational role (e.g. they are 'acting up'), their permissions may not be revoked once they default back to their original job role. Organisations are rigid in assigning user permissions when creating new user accounts and follow standard operating procedures. They often have a

\*Correspondence: [s.parkinson@hud.ac.uk](mailto:s.parkinson@hud.ac.uk)

Department of Computer Science, School of Computing and Engineering, University of Huddersfield, Queensgate, HD1 3DH, Huddersfield, UK

structured (and possibly automated) process for enrolling new users. On the contrary, elevating user privileges is often done by system administrators who make changes based on their experience and analysis to permit required actions.

The manner by which privileges are managed, changed and elevated as employees change role creates potential for a user having many unnecessary and redundant permissions, which are gathered over time. In terms of file system permissions, this could be that a user has access to many resources that are no longer required under their job role. The term associated with this phenomenon is *permission creep*. The security concern with permission creep is that a user can effectively end up with an accumulation of permissions that have both depth and breadth within the file system. Breadth is where the user has accumulated permissions on a large number of directories and depth is where they have acquired a high-level of permissions on a set of directories through accumulating permissions from many different group membership sources.

There are many potential ways to identify permission creep. For example, enabling logging mechanisms that record when a user has been allocated permission. Another solution maybe to take frequent snapshots of user permissions and compare them periodically to determine differences and look for changes that need revoking. These techniques should provide a skilled analyst with the necessary information to determine permission change over time. However, these mechanisms are heavily reliant on expert knowledge and acquiring a rich history of permissions allocation for the underlying system. There is a need to produce a mechanism capable of identifying instances of privileged creep without human interaction and prior knowledge or historical snapshots.

This paper investigates the hypothesis of: modelling file system permissions on a per subject level creates potential to use statistical analysis to identify permissions that appear irregular, and as such could be a result of privilege creep. This allows for the identification of privilege creep without historical allocation information. The aim of this paper is to build on existing research in identifying irregular permissions, where permissions are those that are identified irregular compared to other allocations Parkinson and Crampton 2016. This work includes significant differences in the way that permissions are modelled, processed and empirically evaluated. In earlier work, research focused on the identification of permission allocations for use when assigning new permissions.

Although there are many similarities in the implementation and configuration of different access control systems, this paper focusses on the Microsoft's New Technology File System (NTFS). The motivation for this focus is two fold: (1) the majority of infrastructure file systems are using NTFS, and (2) these systems are vulnerable to cyber-

based attacks, which may execute malware either under user credentials or attempt to gain privilege elevation. For example, malware such as Ransomware (Parkinson 2017; Parkinson et al. 2018) often executes under the user's credentials and therefore ensuring permissions are correctly managed could help minimise the impact of ransomware by ensuring the user cannot access networked resources where they do not require access. The primary contributions presented in this paper are:

- **Technique to extract an 'effective' permission representation.** This technique is capable of extraction a subject effective permission representation within discretionary access control systems. The implementation presented in this paper is for the Microsoft NT file system; however, it is easily extensible to incorporate other file system implementations.
- **Application of statistical analysis to identify instances of permission creep.** A combination of  $\chi^2$  and Jenks natural break analysis is used to identify instances of permission creep, unsupervised and in a generic manner without encoding prior knowledge. This is an important contribution as permission creep is subjective to each implementations access control model and it is not possible to develop a knowledge-base approaches that will work in all instances.
- **Software implementation.** A C# application (named *Creep*) embedding the novel techniques presented in this paper and capable of identifying instances of permission creep in Microsoft NT file systems.
- **Empirical testing** performed through large-scale synthetic instances of permission creep. Synthetic testing allows for a systematic comparison through the use of *Creep* and ground-truth analysis. Empirical analysis is then performed on 5 real-world systems to establish *Creep* accuracy using a comparative study (manual expert, *ntfs-r*, and *Creep*).

The paper is structured as follows: First a detailed analysis of related research is provided. Following on, a modelling section is provided detailing a generic model of discretionary access control systems, as well as providing a technique for translating a Microsoft NT access control implementation in to the provided model. The next section then details how the acquired model can be used to identify instances of permission creep using statistical analysis techniques. Information is then provided on the software implementation (the *Creep* application) of the presented technique. Empirical analysis section is presented providing and discussing both performance and accuracy characteristics of the technique

on synthetic systems where ground truth knowledge is utilised for benchmark analysis. Following on, empirical analysis is performed on five real-world systems where *Creeper* is compared with a human expert and another closely related technique (*ntfs-r*). A conclusion is finally provided, suggesting future avenues of research.

### Related work

Weak access control implementations and erroneous permissions management can introduce vulnerabilities in a file system that can violate data confidentiality, integrity and availability (Pfleeger and Pfleeger 2002). The threat level increases where sensitive data is stored in a distributed network, where multiple users are accessing data for business-critical operations. Weaknesses in controlling access can expose the system to insufficient or the over privileged and incompetent permission administration (Fang et al. 2014). This increases the risk of various attacks, such as aggregation of unauthorised computing resource access, malicious data theft or modification, malware attacks (Parkinson 2017) and others (Benantar 2006). Another type of threat, and that considered within this paper, is permission creep (Vidas et al. 2011). It occurs due to multiple privilege allocations that are accumulated and often go unnoticed by system administrators. Permission creep is challenging to detect due to their inadvertent nature. However, if they go undetected, they could present two significant security risk: (1) privileged users can take illicit advantage, and (2) an attacker can compromise a privileged user and perform a heightened level of malicious activities.

There are many tools and frameworks available that can be used to detect permission escalations and misconfiguration. One such framework is MulVAL (Multihost, Multistage, Vulnerability Analysis) (Ou et al. 2005), which is capable of identifying privilege escalation vulnerabilities in access control configuration data. It uses a knowledge-base containing definitions of security issue and incorrect configurations in the form of rules. The tool has been tested on large-scale systems, where it detected policy violations caused by software vulnerabilities. Another paper presents the Baaz system (Das et al. 2010) that can analyse underlying access control infrastructure to discover potential problems. It is based on group mapping and object clustering techniques that find possible inconsistencies in permission datasets; however, it is focused on analysing the relationship between groups and users, and does not take any consideration to the implemented (or the 'effective') permissions.

Other research presents a crowd-sourcing (knowledge sharing) model for identifying and eliminating access control misconfiguration in home networks. The developed tool, named as NetPrints (Agarwal et al. 2009), utilises a decision tree algorithm to learn configuration information

from users, and automatically suggest solutions for the given problems. In addition to these tools, software applications are available that can help in probing the permission management related issues, such as 'AccessEnum'<sup>1</sup>, 'Security Explorer'<sup>2</sup> and 'Permissions Reporter'<sup>3</sup>.

Many studies have proposed techniques to diagnose and rectify permission controls for mobile environments and applications. A static rule-based technique (Sbirlea et al. 2013) has been developed for Android platform that can detect three kinds of security vulnerabilities. The motivation behind their work is to remove permissions that can cause unauthorized access to sensitive mobile data. This technique was applied to 313 applications, which revealed several exploitable vulnerabilities. In a similar work, an ontology-based framework was created that can be used to regulate and verify the implemented privacy permissions over sensitive data in Android applications (Slavin et al. 2016). The ontology was manually crafted from 50 known security policies. The empirical analysis of the framework showed 341 permission violations in 477 applications. In another study, researchers study the 130 individual Android permissions and monitor how many applications request a higher level of access from the user than the application required (Vidas et al. 2011). Their work comprehensively shows that the majority of applications have greater access than is necessary based on monitoring API interaction. A key difference with work presented in this paper is that they use API call information as ground-truth knowledge of what the application actually requires. Therefore, a rule-based approach of identifying permission creep is possible, unlike in discretionary access control implementations. Web servers are also prone to a large array of attacks due to their exposed (both internal and external) nature. However most of the issues, such as malicious insiders, code injection and so on, can be eliminated by implementing appropriate access permissions. For this purpose, a scheme (Nosevich and Petukhov 2011) is proposed for web applications that applies use-case graph and differential analysis to conduct black-box access control testing. The use-case graph contains the definition of user access roles and dependencies, and is constructed with the help of human assistance.

Previous research shows that performing a test of independence over the access control data can reveal irregular permissions. In this study (Parkinson and Crampton 2016), the researchers used the  $\chi^2$  technique to separate out those permissions that failed the test of dependence and applied  $k$ -nearest neighbours algorithm to propose feasible access control rules based on the given system. The research presented in this paper is built-upon and motivated by this previous research. It should be noted that although the research utilises the same  $\chi^2$  technique, the modelling and representation of the underlying permissions is fundamentally different. In previous work, an

effective permission model was utilised to represent permissions mentioned explicitly in each Access Control List (ACL). The aim of the research was to identify potential irregularities within permissions allocated in the ACL alone. In this research we adopt a holistic model whereby the effective permissions of all users are extracted. This requires looking beyond the ACL, following group memberships to calculate a complete set of effective permissions. This ensures that permissions that are acquired through a complex link of group memberships and ACL entries are not missed.

In other research, association rule mining (ARM) methods have also been utilised to determine unnecessary permissions. Initially the ARM technique was employed to determine access-control misconfiguration and predict intended policies (Bauer et al. 2011). Following on, another study used a matrix algorithm (Yuan and Huang 2005) to extract infrequent rules that have low support and confidence values (Parkinson et al. 2016). This led towards the identification of anomalous and irregular permissions. Machine learning has also been used to determine irregular permissions. A recent article Shaikh et al. 2017 proposed decision tree and data classification based algorithms to identify incomplete and inconsistent (boolean allow or deny) policies within access control datasets. A key point in these studies is the representation of permission's data in a uniform format, such as object-based models, which helps in producing the useful results specific to the underlying system.

Recent studies suggest that the identification of elevated permissions is a complex task and requires extensive expert auditing knowledge and diverse experience. However most of the existing solutions use static knowledge, which is hard-coded and requires continuous addition (Bartel et al. 2014). The lack of expert knowledge and continuous investment of time and effort in encoding and representing the knowledge is a significant limitation. These issues motivate the need for an unsupervised classification techniques that can detect permission creeps in real-world environments with good accuracy and without manual acquisition and representation of expert knowledge.

## Modelling

In this section, a generic model is provided detailing how discretionary access control mechanisms are structured and how they can be represented using an effective permission model. This model is utilised throughout the technical developments presented in this paper.

### Access mask

A directory,  $D$ , can have a set of child directories where  $D = \{d_1, d_2, \dots, d_n\}$ . Each directory has an Discretionary Access Control List (dACL) containing a series of Access

Control Entities (ACEs),  $d_n = \{acl\}$ , such that  $acl = \{ace_1, ace_2, \dots, ace_n\}$ , which dictates the level of access given to a subject, where a subject can be any user or system component (e.g. software) that requires access. Each ACE has several key parameters, but those necessary in this paper are: a subject represents the subject that the ACE is assigned to, an access mask which contains information regarding the level of permissions and the inheritance flags,  $ace = \{s, p, i\}$  where  $s$  is the subject,  $p$  is the permission set, and  $i$  denotes the inheritance flags. The permission  $p$  is a set of standard attributes from the predefined set of attributes  $p \subseteq A, A = \{a_1, \dots, a_n\}$ . NTFS provides six levels (e.g., full control, modify, etc.) of standard coarse-grained permission that consist of a combination of predefined attributes. These attributes are drawn from the standard set of fourteen permission attributes, which detail that the subject can perform a fine-grained task. For example, "create files", "create folders", and "read permissions". NTFS also allows for the creation of special fine-grained permissions, consisting of any combination of the fourteen individual attributes.

### Propagation and inheritance

Within the dACL, there are two types of Access Control Entity (ACE); (1) Explicit and (2) Inherited. Explicit entries are those that are applied directly to the object's 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). For example, an Explicit allocation would ensure that a parent,  $d_p$  and child directory  $d_c$  have different ACLs ( $ACE_p \neq ACE_c$ ), where  $p$  and  $c$  denote parent and child directories and ACLs, respectively.

### Group membership

As previously mentioned, a subject can be a user, group or process within a system. The potential to assign a group permission on a directory allows the possibility for all the group's members to automatically acquire the same permission through group membership. There are several motivating factors as to why this is useful and widely used in real-world systems. The primary reason is that managing file system permissions on a per-user basis would be cumbersome and would result in large ACLs and would introduce additional computation overheads during processing. A secondary reason is that using group memberships allows the users to implement and operate a role-based access control system, whereby clear separations of duty are made within organisations and users are allocated to roles depending on the requirements of their job role. A subject  $s$  can either be a user or process, and as such is modelled as  $s = \emptyset$ . Alternatively, it can be a

group containing a set of other groups, users or processes,  $s = \{s_1, s_2, \dots, s_n\}$ .

### Accumulation

Accumulation is the possibility for the subject to receive an *effective permission* acquired from 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. This makes managing permissions easier for an administrator; however, it does introduce the potential for subjects to gain permission on any resource where the group is assigned.

Algorithm 1 details the process of how the effective permission is calculated from iteratively traversing a directory structure. The algorithm provides functionality beyond that of the operating systems standard mechanisms of calculating the effective permission as it identifies the effective permission for all subjects within the system. The algorithm takes as input an initial directory,  $d$ , and a set of subject relationships (i.e., group memberships). The output of the algorithm is a set of effective permissions,  $E = e_1, e_2, \dots, e_n$ , where each individual effective permission is a triple tuple,  $e_n = \{d, s, p\}$ , where  $d$  is the directory resource,  $s$  is the subject, and  $p$  is the permission level. The complexity of the recursive algorithm to determine effective permission is of  $O(|d| \times |acl|^2 \times |R|)$ . Here is the number of directories ( $|d|$ ) processed in total, multiplied by the number of access control entities ( $|acl|^2$ ) within each access control list, raised to the power two as is necessary to perform nested recursion, and finally, the number of members in each group ( $|R|$ ).

The algorithm works by iterating over each access control entity, identifying the subject,  $s$ , and their level of permission,  $p$  (line 4). Following on, the other ACE's are inspected to see if they contain another permission relating to the same subject or a group with which the subject is inheriting group membership. The `getAllGroups` function returns a set of groups that a specific subject is a member of. If another ACE is identified with a subject either matching the subject, or one that exists in the set of groups that  $s$  is a member of, the permissions granted to that ACE are accumulated with  $p$  (line 10). The output of this technique is that the permission object will be the union of all permission attributes allocated to  $s$ . After the effective permission has been allocated, it is the necessary to determine if subject  $s$  is a group or not (line 16). If they are a group, then effective permission entries for each group member are added to the list of effective permissions,  $E$  (line 18).

---

**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  $d$

**Input:** Subject relation set,  $S = \{s_1, s_2, \dots, s_n\}$  where  $s_n = \{s_1, s_2, \dots, s_n\}$  or  $s_n = \emptyset$

**Output:** Set of effective permissions

$E = e_1, e_2, \dots, e_n$  where  $e_n = \{d, s, p\}$ . Here  $d$  is the directory resource,  $s$  is the subject, and  $p$  is the permission level,  
 $p = \{a_1, a_2, \dots, a_n\}$

```

1 Algorithm algo (directory  $d$ )
2    $acl \leftarrow d(ACL)$ 
3   foreach  $ace$  in  $acl$  do
4      $\{s, p, i\} = ace$ 
5      $R \leftarrow \text{proc}(s)$ 
6     foreach  $ace'$  in  $acl$  do
7        $\{s', p', i'\} = ace'$ 
8       foreach  $s_g$  in  $R$  do
9         if  $s' = s_g$  then
10           $p = p \cup p'$ 
11        end
12      end
13    end
14  end
15   $E \leftarrow \{d, s, p\}$ 
16  if  $s \neq \emptyset$  then
17    foreach  $s'$  in  $s$  do
18       $E \leftarrow \{d, s', p\}$ 
19    end
20  end
21  foreach  $subdirectory\ c$  of  $d$  do
22     $algo(c)$ 
23  end
24  return  $E$ 
25
26 getAllGroups proc (subject  $s_{in}$ )
27    $R = \emptyset$ 
28   foreach  $s$  in  $S$  do
29     if  $s \neq \emptyset$  then
30       foreach  $s'$  in  $s$  do
31         if  $s' s_{in}$  then
32            $R \leftarrow s'$ 
33            $getAllGroups(s')$ 
34         end
35       end
36     end
37   return  $R$ 

```

---

### Detecting creep

The next stage is to process the set of effective permissions to identify permissions that are irregular and could indicate an instance of permission creep. This section describes how irregularities in permission allocation can



be identified through a statistical test of independence. A statistical test of independence approach has been adopted due to many previous successful implementations within security analysis (Ye and Chen 2001). Using statistical analysis to determine irregular permissions creates the potential to categorise irregular, but correctly assigned, file system permissions. This is because in large multi-user systems there are many file system permissions which are customised for only a few people, where the all remaining employees have permissions acquired by group membership. 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.

Previous research has seen the successful use of  $\chi^2$  analysis to identify anomalies in file system permissions (Parkinson and Crampton 2016). This research builds on previous work by modelling file system permissions as a collection of subject and effective permissions; however, with the addition of exhaustively identifying all subjects with permission over each resource. The below sections detail how this representation, which is output from Algorithm 1, is used by the presented analysis techniques to identify an instance of permission creep.

### Chi-square analysis

$\chi^2$  statistics are used to measure the lack of independence between  $a$  and  $s_j$ , which can then be compared to the  $\chi^2$  distribution with one degree of freedom to judge extremeness (Greenwood 1996). The  $\chi^2$  statistic is selected as it is an 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,  $\chi^2$  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  $s_j$  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. Here  $s_j$  is the subject in each effective permission entry,  $e = \{d, s_j, p\}$ , and each  $a$  is an individual attribute from the set of permissions,  $p$ .

From this a lack of independence measure between attribute  $a$  and object  $s_j$  by:

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

The  $\chi^2$  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  $s_j$  with a  $\chi^2$  value close to zero is either an anomaly or an irregular permission attribute. Following the calculation of  $\chi^2$  scores, it is then useful to compute the mean  $\chi^2$  for each permission using the following equation where  $l$  is the number of attributes specified for a permissions:

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

Once the average for each permission allocation has been calculated ( $\chi_{avg}^2(p, s_j)$ ), it is then necessary to calculate an average permissions allocation for each subject,  $s_j$ . This requires calculating mean  $\chi^2$  values that relate to the same subject. The following equation is used to calculate  $\chi_{subject}^2(s_j)$  values where  $k$  is the number of  $\chi_{avg}^2$  for the subject in question,  $s_j$ :

$$\chi_{subject}^2(s_j) = \sum_{j=1}^k \chi_{avg}^2(p, s_j) \quad (3)$$

This allows us to identify permissions that appear irregular. However, difficulty arises when deciding the cut-off threshold for  $\chi_{subject}^2$  that should be treated as potentially irregular and those that appear normal. Expert analysis would help separate the anomalies from regular permissions, but in many cases such expert knowledge is not available. Therefore, in this paper a technique is presented which attempts to classify  $\chi^2$  scores which are most likely to be anomalies or irregular. To perform this classification, Jenks natural breaks classification method (Jenks 1967) is used 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  $\chi^2$  scores) is the class of permissions which have failed the test of dependence and are to be treated as potential irregularities. To perform this, the following classification function is used:

$$I(x): \{1..n\} \mapsto \{1..k\} \quad (4)$$

where  $n$  is the number of data samples ( $\chi_{subject}^2$  values), 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 the sums of standard deviations ( $SDD_{n,k}$ ) is then calculated by:

$$SDD_{n,k} = \min_I \sum_{j=1}^k ssd(S_j) \quad (5)$$

$ssd(S_j)$  is the sum of the squared deviations of the values of any index set  $S$  calculated using the following equation

where  $O$  is an ordered set of  $\chi^2$  scores.

$$ssd(S) = \sum_{i \in S} \left[ O[i] - \left( \frac{\sum_{i \in S} O[i]}{|S|} \right) \right]^2 \tag{6}$$

In the first instance of computing  $SDD_{n,k}$  values,  $k = |\chi^2_{subject}|$ . Here  $|\chi^2_{subject}|$  represents the count of unique elements in the set containing all  $\chi^2_{subject}$  values.  $k$  is then decremented 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  $\chi^2$  scores, it can be assumed that the first class containing those  $\chi^2$  scores close to zero (set  $S_1$ ) are likely to be irregular or anomalous. However, the case may arise where multiple classes (e.g. sets  $S_1, S_2$ ) both contain permissions that are irregular or anomalies. It is also possible that in the case that no anomalies are detected, meaning the lowest class ( $S_1$ ) will be contain  $\chi^2$  values for correct permissions.

To aid understanding, an example is provided in Table 1. In this example, a ‘‘Test user’’ is added to hold permission on a computer’s  $C:\backslash$ . The allocated permissions are the default for all user groups (Users, System, and Admin) and the Test user is a member of the Users group. In addition, another permission entry has been entered for the Test user of Full Control on a subdirectory structure within the  $C:\backslash$ . More specifically,  $C:\backslash\Users$ .

As evident in Table 1, the  $\chi^2_{subject}$  value for the Test user is significantly less than the other three groups. In addition, it is evident that the Users group is less than both System and Admin. This is because Users have a lower level of permission, and therefore relationships to fewer attributes, on a majority of the directory structure. On the contrary, System and Admin have a high level of permission (Full Control) throughout the directory structure. It is noticeable that the Jenks analysis technique has identifies three classes for the  $\chi^2_{subject}$  values, and Test user is in the lowest and has been identifies as an instance of potential permission creep.

**Table 1** Scores form analysing a directory structure where ‘‘Test user’’ was deliberately modified to mimic an instance of permission creep

Subject	Permission	$\chi^2_{subject}$	Class No.	Potential Creep
Test user	Read & Execute, Full control	1	0	yes
Users	Read & Execute	468	1	no
System	Full control	1080	2	no
Admin	Full control	1120	2	no

### Implementation

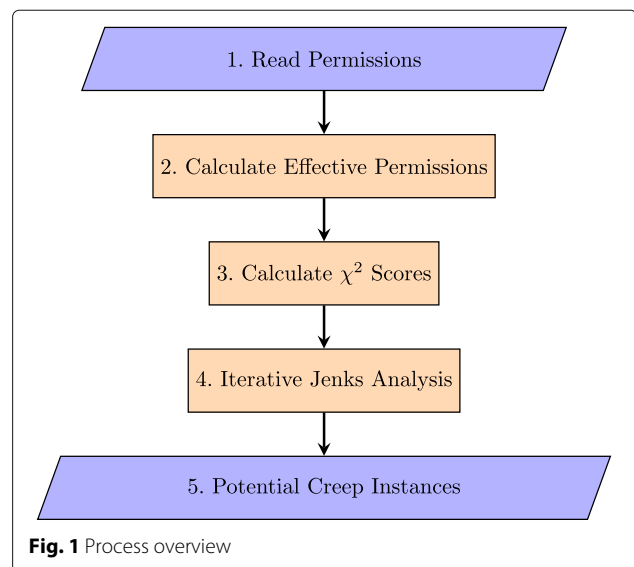
The technique presented in this paper has been implemented in a C# application, named *Creeper*. The motivation behind using C# language is due to its native integration with the Microsoft platform for extracting file system permissions. As this work targets the Microsoft NT file system, implementing the technique in C# – a Microsoft proprietary language – is not at detriment to both usability and impact.

Figure 1 provides a graphical overview of the process implemented in *Creeper*. The first part of the process is where the permissions are read using native system functionality and stored using the model presented in ‘‘Modelling’’ section. Following on, this acquired permissions information is the processed to establish the effective permission representation for each subject within the system. The next stage is to calculate  $\chi^2$  values based on the effective permission representation. Jenks natural breaks is then iteratively performed until the optimal classing is identified. The output of the software is the list of permission creep instances.

Table 2 illustrate the results of *Creeper* software and provides the interface shown to the user after analysis has taken place on a specified directory. The three columns represent the subject,  $\chi^2_{subject}$  scores, and whether or not that subject has been identified as ‘‘Of Interest’’ through performing Jenks analysis. ‘‘Of Interest’’ refers to the likelihood of an subject’s permissions being an instance of permission creep and that they warrant further investigation.

### Empirical analysis

In this section, empirical analysis is performed to determine the *Creeper*’s ability to detect instances of permission creep. The empirical analysis is performed in two



**Table 2** Results from Creeper demonstrating user "Extra1" has been identified as a potential instance of permission creep

User/Group	Average Score	Of Interest
CES000964708 \Extra1	98.267478318174	✓
NT AUTHORITY \SYSTEM	1056936.28738905	×
Administrator	1056936.28738905	×
bakman	1056936.28738905	×
bakman2	1056936.28738905	×
cmsxajr	1056936.28738905	×
cmsmig	1056936.28738905	×
Domain Admins	1056936.28738905	×
cmsxpjh	1056936.28738905	×
Enterprise Admins	1056936.28738905	×
Adminback	1056936.28738905	×
momadion	1056936.28738905	×
Glenlivet	1056936.28738905	×
cmsxbak	1056936.28738905	×
cmsxaps	1056936.28738905	×
SVACHVCBK	1056936.28738905	×

distinct phases. The first involves the generation of synthetic datasets where ground truth knowledge is available. The second includes the analysis on real-world datasets, where comparison is performed from manual analysis as well as a previous technique developed to autonomously identify irregular permissions. A comparative analysis is made to determine *Creeper()*'s ability to identify irregular permissions that are missed using more traditional approaches.

### Synthetic analysis

In order to empirically evaluate the proposed technique's ability to detect permission creep, an iterative approach of using synthetically generated datasets has been adopted. This technique takes as input the following:

- **Number of roles** is used to define the number of roles within the directory structure, which represent the number of organisational roles. For example, Management, Human Resources, etc;
- **Directory complexity** represents the depth and breadth of the synthetic directory structure. A directory structure will be created to the specified depth, with each directory containing the same amount of subdirectories. For example, a directory complexity of 4 would result in the creation of a directory structure with a maximum depth of 4, and a breadth of 4 for each subdirectory. This exponential growth would create a directory size of  $4^4 = 256$ ;
- **Total number of users** within the entire system would be equally distributed among the number of

roles. For example, in a system with 100 users and 5 groups would result in 20 users per role; and

- **Number of users** with artificially induced permission creep represents those users where additional privileges (i.e. adding to additional roles) have been added to mimic an instance of permission creep.

A process has been created to automate the construction of the synthetic directory structures that are utilised in this empirical analysis. In this work, the process was implemented in a Microsoft Powershell script. In addition to creating the synthetic directory structure, the script will also output the users that have been assigned permissions representative of a user experiencing permission creep. This provides the necessary ground-truth knowledge for analysing *Creeper*'s ability to accurately detect permission creep. The following ordered list details the process of creating a synthetic file system as used in this research:

- 1 **Setup** the file system structure, which includes creating the directories, users and groups within the system. At this point the necessary components have been created; however, no group and permission allocations have yet been made.
- 2 **Assign users to groups** is where users are allocated to permissions groups, which are used to represent user roles. In this allocation, an even distribution is made whereby the number of users is divided by the number of groups.
- 3 **Assign permissions** is where file system permissions are assigned to each group. In this research, the permission is assigned as combination of individual attributes where the first group gets the full set of attributes (i.e., Full Control) and subsequent groups get less expressive combinations. More specifically, the number of permissions and the power they hold on the file system is decreasing, which ensures each group has a different permission level.
- 4 **Assign creep instances** implements additional permissions directly to users (not their group) by pseudo-random selection. A user is first selected along with a directory. The next stage is to select a pseudo-random combination of permissions attributes and for the selected user and directory, and finally, the allocation is applied to the file system. This information is also written to a text file to be used as group truth knowledge when analysing the output from using *Creeper*.

The above process is iterative and executed using the minimum and maximum values for each parameter in this empirical analysis, as well as the incremental step size for each parameter provided in Table 3. These synthetic systems allow for a systematic empirical analysis of *Creeper*'s performance.



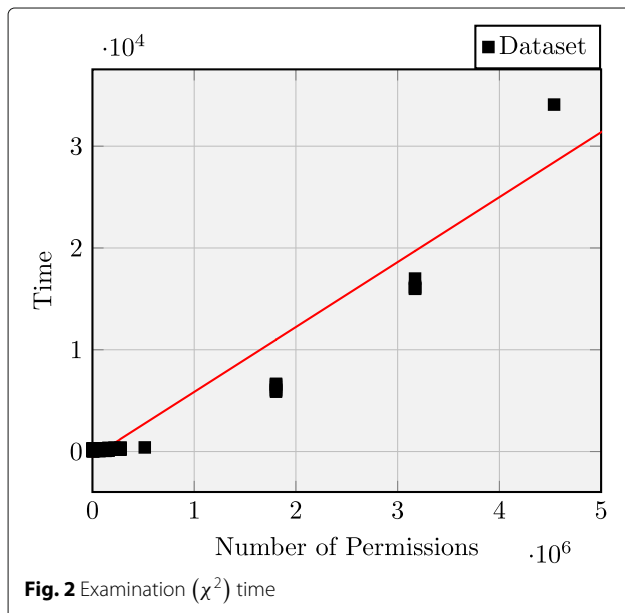
**Table 3** Experimental analysis parameter variation (minimum, maximum and stepsize)

Parameter	Min	Max	Stepsize
Number of roles	2	4	1
Directory complexity	2	5	1
Total number of users	100	500	100
Percentage of creep users	0	10	2

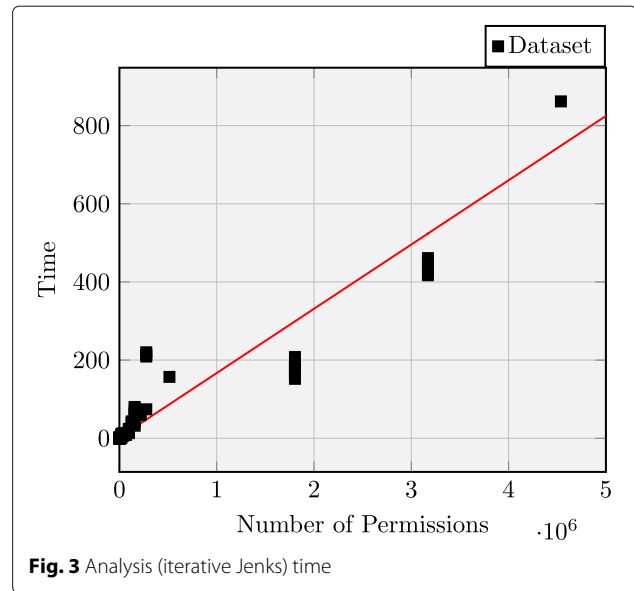
It is worth noting at this point that there is no guarantee that any inserted anomaly will be detected. More specifically, it could be that the random allocation made to represent an instance of permission creep is actually less expressive than those the user already has, and therefore would fail to represent an instance of permission creep. The use of synthetic directory structures is exploring the impact on the number of roles, directory size, number of users, and number of creep permission instances. In doing so, the implemented technique is inserting the instances of permission creep in the file system configuration and then detecting them in the set of extracted effective permissions. The interpretation (by the Operating System) of the file system configuration into the effective permission set is what is being explored. All test data sets, scripts, software and results are available for further research and details are provided in “Conclusion” section.

**Scalability**

Figures 2 and 3 illustrate performance characteristics of the implemented technique, in terms of performing  $\chi^2$  analysis (Fig. 2) and in performing iterative Jenks analysis for classing the results (Fig. 3). In both figures, the



**Fig. 2** Examination ( $\chi^2$ ) time



**Fig. 3** Analysis (iterative Jenks) time

amount of time required is shown against the number of permissions processes. In both figures there are clusters of data points which represent each directory complexity (2, 3, 4, and 5 in order of left to right). In addition, a linear line of best fit is included to show the rate at which time increases along with the number of permissions. This allows for the visual identification of the increase in computation time with the increase on the total number of permission entries. As demonstrated in both Figs. 2 and 3, the increase is linear and indicates good scalability as the number of permissions increases.

From analysing both Figs. 2 and 3, it is evident that the quantity of time required for extracting permissions and performing  $\chi^2$  analysis is significantly greater than required to perform Jenks classification. On average, a total of 4,650 seconds is required to complete the entire process, of which 92% is for  $\chi^2$  and 8% for Jenks classification. This is to be expected as  $\chi^2$  analysis as the calculation of  $\chi^2$  scores is quadratic and so has a complexity of  $O(n^2)$ , where  $n$  is the number of permissions to analyse.

**Identifying permissions creep**

To assess the performance in this empirical analysis, the following measures are considered:

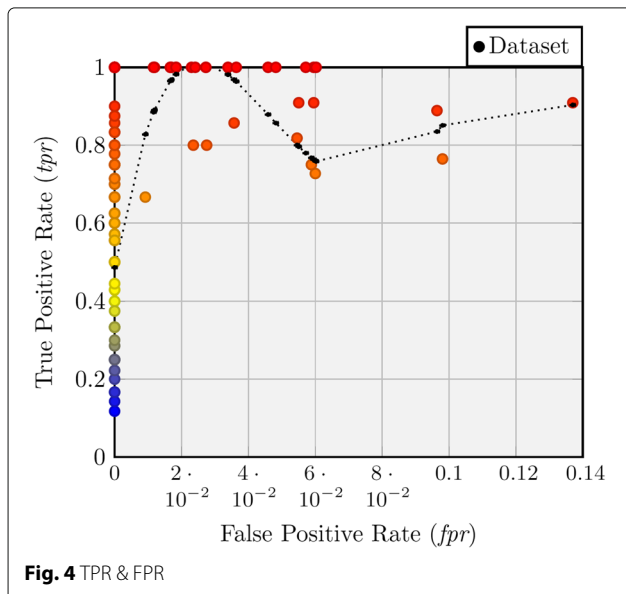
- 1 True Positive Rate (*tpr*): the fraction of creep permissions correctly identified as being part of an instance of permission creep;
- 2 False Positive Rate (*fpr* = 1 - *tnr*): the fraction of regular permissions incorrectly identified as being part of an instance of permission creep;
- 3 True Negative Rate (*tnr*): the fraction of regular permissions correctly identified as regular;

**Table 4** Results from performing experimental analysis

No. of Roles	Dir Complexity	fpr	tpr	fnr	tnr	Avg.
2	2	0.02	0.84	0.16	0.98	0.97
3	2	0.02	0.88	0.12	0.98	0.96
4	2	0.02	0.73	0.27	0.98	0.94
2	3	0.00	0.69	0.31	1.00	0.97
3	3	0.00	0.64	0.36	1.00	0.97
4	3	0.00	0.65	0.35	1.00	0.96
2	4	0.00	0.68	0.32	1.00	0.97
3	4	0.00	0.64	0.36	1.00	0.96
4	4	0.00	0.70	0.30	1.00	0.98
2	5	0.00	0.66	0.34	1.00	0.96
3	5	0.00	0.64	0.36	1.00	0.95
4	5	0.00	0.64	0.36	1.00	0.97
Average		0.00	0.70	0.30	1.00	0.96

- 4 False Negative Rate ( $fnr = 1 - tpr$ ): the fraction of creep permissions incorrectly classified as regular; and
- 5 Accuracy is reported as the fraction of all samples correctly identified. More specifically,  $Accuracy = \frac{tpr+tnr}{tpr+tnr+fpr+fnr}$ .

Figure 4 presents the  $tpr$  and  $fpr$ . Table 4 provides the more detailed results, which are averages for the number of roles and directory complexity combinations. Although there are slight variations within each combination due to differences in number of users and instances of permission creep, differences in the number of permissions changes significantly with any increase in directory complexity and number of roles.



**Fig. 4** TPR & FPR

As it can be identified in the Fig. 4 and Table 4, *Creep* has a high  $tpr$  and a low  $fpr$ . This is of significance as it demonstrates the technique is able to correctly identify instances of permission creep, whilst not incorrectly classifying correct permissions as instances of creep, which was specified to fit to the sixth degree. The Area Under Curve (AUC) calculated from Fig. 4 is 0.76. The AUC is calculated through applying using a best-fit polynomial curve fitting function. There are many instances that have a high  $tpr$  and have a  $fpr$  beyond zero and no greater than 0.18 (18%). Instances that have a low  $tpr$  do not have a high  $fpr$ . This demonstrates that the system is conservative, and is less likely to not identify instances of permission creep than incorrectly identifying regular permissions.

Table 4 details that smaller directory structures have a higher  $fpr$ , indicating that *Creep* incorrectly identifies some normal users as being those indicative of permission creep. From analysing the results it is evident that those with a directory complexity of 2 ( $2^2 = 4$  directories) have a  $fpr$  or 0.02 (2%). In the example, the average number of permissions (Access Control Entities) for directory structures with a complexity of 2 is 240. Although the number of directories is low (4), the number of permissions reflects each user within the system that can access a specific directory. The algorithm presented in Algorithm 1 creates a permission entry for each subject that has access on the directories, acquired through group memberships. This approach is adopted as *Creep* is seeking to identify subjects with irregular permission creep, which in many file systems permissions are managed through group membership allocations.

Furthermore, the  $fpr$  decreases to 0 with a directory complexity greater than 3 (27 directories), which demonstrates that the accuracy improves as the directory structure increases. This improvement is due to the fact that the number of permissions increases proportional to the number of directories, and that irregular permissions will become increasingly statistically insignificant as the number of directories increases.

The  $tpr$  as shown in Table 4 details that *Creep* has a good ability to correctly identify instances of permission creep. The average  $tpr$  for all experiments is 0.7 (70%). The rate is higher on smaller directory structures, greater than 0.7 for directory structures of complexity 2. All experiments with a directory complexity of 3, 4, and 5 have an  $tpr$  is greater than 0.6 (60%). Although the average  $tpr$  of 0.7 does mean that 0.3 (30%) of instances of permission creep are not correctly identified, it is worth noting that the system has a low  $fpr$  and the system is operating in a conservative nature.

The false negative rate ( $fnr$ ) details the fraction of regular normal that are incorrectly identified as instances of permission creep. As evident in Table 4, directory structures with a complexity of 2 have the lowest  $fnr$  of less than

0.3 (30%). All other experiments with a larger directory structure size have a *tpr* of between 0.30 and 0.36 (30% to 36%). This demonstrates that the *Creep* has a high false positive rate, and as such does fail to identify instance of permission creep and reports to the user that they are normal.

The true negative rate (*tnr*) is also high, with only results from a directory complexity of 2 being 0.98 (98%) and all other results 1 (100%). This is of significance as it demonstrates the ability of the technique to correctly identify permissions that are normal as not being part of an instance of permission creep.

Table 5 demonstrates average results for the number of users with permission creep, irrespective of directory size. As evident in the table, the *fpr* is 0 until the number of users with permission allocations representative of permission creep reaches 8, and at this point the average *fpr* increases to 0.01 (0.1%).

It is also evident that the *tpr* decreases along with the number of users, indicating that as the number of users with synthetic permissions increase, the ability for *Creep* to correctly identify instance of permission creep decreases. This is because as the number of users with permissions increases, the difference between average  $\chi^2$  class values for class 0 (those determined to have poor dependence) and class 1 (containing regular permissions) decreases.

The *fnr* increases as the number of users with creep permissions increases. The *fnr* is at 0% with 0 users with creep permissions, rising to 40% with 10 users representing instances of permission creep. The *fnr* represents the number of users that are incorrectly identified as normal, when they actually represent an instance of permission creep. This is of significance as it demonstrate that there is a significant potential to miss users with permission creep should the underlying system has many instances of permission creep. In terms of the practical use of *Creep*, the system would need to be run after each identified instance of permission creep is rectified, and thus allowing those that are incorrectly identified as normal to be more distinguishable.

**Table 5** Average results based on number of users with permission creep

No. of users with Creep	Avg. fpr	Avg. tpr	Avg. fnr	Avg. tnr	Avg. Accuracy
0	0.00	1.00	0.00	1.00	1.00
2	0.00	0.75	0.25	1.00	0.99
4	0.00	0.61	0.39	1.00	0.97
6	0.00	0.63	0.37	1.00	0.96
8	0.01	0.61	0.39	0.99	0.94
10	0.01	0.60	0.40	0.99	0.93

The *tnr* –fraction of permissions correctly identified as normal– is consistently high (100%) until when there are 8 users with an instance of permission creep, and at this point it decreases to 99%. Finally, the average accuracy is displayed for the number of users representing permission creep. It is noticeable that that accuracy is 100% with 0 instance of permission creep, and gradually decreases to 93% with 10 instances of permission creep. This shows good accuracy and validates the suitability of the technical approach to identify instances of permission creep.

### Real-world analysis

In the previous Section, an average accuracy of 96% has been established on synthetic datasets. Although these datasets are realistic and are generated to align with common access control implementations, it is still necessary to test the capabilities of *Creep* on real-world systems. This is particularly important as the diversity of implemented permissions within real-world systems may be different from those in synthetic systems. As well as analysing *Creep's* ability to detect instances of permission creep in real-world systems, we have created the possibility to make a direct comparison between *Creep* and a human expert. Furthermore, a comparison is made with a previous implementation of a similar technique (names *ntfs-r*), presented in (Parkinson and Crampton 2016), which models the underlying permissions differently to search for statistically irregular permissions. As previously mentioned, this earlier version only analyses permissions which are those directly allocated in the ACL and the technique does not calculate effective permissions for those receiving permissions through group memberships, meaning that there is a potential to miss users with an instance of permission creep. During this analysis, the following methodology is used:

- A human expert with extensive experience (greater than 10 years) in performing security audits will analyse the file systems using only traditional analysis method of examining access control rules using built-in operating system functionality;
- *ntfs-r* is used to identify irregular permissions, based on a previous implementation of  $\chi^2$  and Jenks analysis that utilises effective permissions from users and groups explicitly in the ACL;
- *Creep* is used to extract and analyse permissions, which specifically aims to identify for instances of permission creep; and
- Irregularities identified are evaluated by a separate human expert, and they are regarded as ground truth i.e. correct identifications used to determine the accuracy of other techniques irrespective of which technique identified them as a valid instance of permission creep.

Table 6 presents the characteristics of the five different file systems (referred to as datasets) analysed in this work. The number of directories provided in the second column is the total number of directories analysed. The number of ACLs examined in column three is significantly lower as it only contains permissions that are unique from their parent directory. More specifically, they are not inheriting their permissions from their parent directory. It should be noted that unlike in earlier synthetic analysis, ground truth knowledge is not available and thus the 'correct', 'incorrect' and 'missed' results are determined through expert interpretation of the results to identify those that are correct. For example, a correct instance of permission creep identified by the human expert would count as 1 correct classification, otherwise their incorrect count would be incremented by 1. Should any technique (including human expert) fail to identify a valid instance of creep which was identified by another technique, then their missed classification score will be incremented.

All directories used in this analysis have been acquired from real-world, multi-user and multi-device systems being used within commercial organisations and they are not part of the research infrastructure used in the development of the presented technique. All systems are hosted by organisations collaborating with the authors of this research; however, due to the security sensitivity of the data, we are unable to publicly make available the datasets or acknowledge the organisations. The file systems displayed in Table 6 are presented in size order and the number of different permission levels are also presented. Although this work is identifying instances of creep on a per user basis, it is interesting to note the number of unique permission levels used throughout the system. For example, Dataset 2 only has three unique levels whereas dataset 4 has 23. A higher number of unique permission level might be an indication that some users are obtaining an irregular set of permission attributes. Furthermore, an assumption can be made here that file systems with a higher degree of unique permissions levels will become more complex to manage.

For the purpose of aiding the reader in understanding an example instance of permission creep discovered, an

excerpt from dataset number 4 is presented. This dataset has been extracted from a large organisation, and as such there are many users frequently changing their job roles within the organisation. One instance of permission creep identified and verified was that of *user10* (anononymised due to commercial sensitivity) having *full control* on directory structures associated with human resources department (`\\shared\HR`) and the remainder of the user's permissions were allocated through the *finance* group. After further analysis, it was discovered that the user was undertaking maternity cover for an employee within the Human Resource department and gained new directly allocated permissions, which were never removed once their temporary role had finished. The allocation had not previously been discovered as the company has a policy of not making direct allocations; however, it was noted that the temporary allocation of job role coincided with the start of a new IT administrator, who it is believed made the allocation when new to their role and had not gained a full understanding the host organisation's security policies. This demonstrates the ability of the technique to discover instances of permission creep within an organisation that have not yet been discovered.

Table 7 presents the results from performing manual, *ntfs-r*, and *Creeper*. The first thing to highlight from the results is that all analysis techniques agree that there are no instances of permission creep in datasets 1 and 3. The results from calculating the average percentage of correctly identified instances of permission creep for manual analysis, *ntfs-r*, and *Creeper*, are 55%, 60%, and 98%, respectively. This demonstrates that *Creeper* has the best accuracy and was able to correctly identify the most instances of permission creep. It should be noted that as there is an absence of ground truth knowledge, there is potential that there are instances of permission creep that are not identified by any technique. This is because auditing real-world access control implementation to determine the correctness of each individual permission would require significant human effort. However, it should be noted that the identified 98% accuracy is consistent with synthetic analysis using *Creeper* where an average accuracy of 96% is achieved. The remainder of this section discusses the results presented in Table 7 to gain an understanding of why *Creeper's* accuracy is high when compared to other analysis techniques.

Human analysis is accurate in terms of not making incorrect classification; however, there are many instances where the expert has missed to identify valid instances of permission creep. This is most likely due to time constraints when performing manual analysis using standard operating system functionality. More specifically, when analysing permissions in Microsoft NT systems, the user has to inspect permissions on the individual object (e.g. directory or file) level. This is time consuming and the lack

**Table 6** Characteristics of the real-world file systems acquired for empirical analysis

Dataset	No. of Directories	No. of ACLs examined	No. of Permission levels
1	168	42	8
2	254	24	3
3	570	74	6
4	3,517	499	23
5	11,654	870	15

**Table 7** Comparison of techniques

Dataset Directory	No. of users with Creep (Manual)			No. of users with Creep ( <i>ntfs-r</i> )			No. of users with Creep ( <i>Creeper</i> )		
	C	I	M	C	I	M	C	I	M
1	0	0	0	0	0	0	0	0	0
2	8	0	7	8	7	7	14	1	0
3	0	0	0	0	0	0	0	0	0
4	2	0	8	6	0	4	10	0	0
5	0	0	3	3	79	0	3	0	0

**C** = Correct, **I** = Incorrect, and **M** = Missed

of an ability to retain and view a user's permission across multiple objects makes it challenging to identify permissions that are statistically irregular and could potentially indicate an instance of permission creep.

The *ntfs-r* implementation did correctly identify a higher number of creep instances than the expert; however, not only did it miss some instances, but it incorrectly identified many instances, particularly in dataset 5. After cross analysing with Table 6, it has been determined that a contributing factor to this high incorrect classification could be down to the large number of ACLs examined and the diverse usage of permission levels. An even distribution of permissions will occur if the system has a high number of permission allocations using the the same set of permission level. If the system has a relatively small number of permissions using different permission levels, then they will be identified as irregular as they do not fit the normal distribution and are statistically irregular.

*Creeper* identifies more correct instances of permission creep than other techniques. It does however incorrectly identify an instance of permission creep in dataset 2. After further analysis, it is identified that the technique has been too sensitive in this instance and the incorrectly identified effective permission is actually normal (a false positive), although it is statistically different and therefore could have represented a valid instance of permission creep. This demonstrates that *Creeper* can incorrectly classify permissions in some instances and still requires human expertise to analyse the results. The reason for this incorrect identification is because the correct permission is statistically different from the normal distribution and thus is identified as an instance of permission creep. However, it should be noted that it was able to correctly classify permission with a higher degree of accuracy than all other techniques, and thus demonstrates *Creeper's* capabilities.

Table 7 illustrates that *ntfs-r* identifies a higher number of permissions as potential instances of permission creep. In the analysis these are determined as incorrect classifications of permission creep; however, it is worth noting that these allocations have been identified by *ntfs-r* as they appear to be irregular and anomalous with the directory structure. Although *Creeper* uses a similar

underlying technical approach, the significant difference and improvement in accuracy is down to the difference in the way that effective permissions are modelled for all accessible users and not just those specified in the ACL.

As discovered through this analysis, datasets 2, 4 and 5 all have instances of permission creep. In terms of statistical significance, dataset 2, 3 and 5 have 6%, 0.2% and 0.02% of their permissions representing instances of permission creep, respectively. The percentage of permissions that represent instance of permission creep are within the range of those tested in the synthetic analysis presented in this paper. More specifically, in the synthetic analysis, the percentage of permission creep instances ranges from 100% to 0.3%. These percentages are calculated based on the fraction of normal permissions against the instances of permission creep. A contributing factor to the accuracy of the technique on real-world dataset is that the percentage of permission creep instances is lower than in the synthetic analysis. This contributes to the improvement in accuracy as it has previously been established that *Creeper's* accuracy is greater when fewer instances of permission creep exists. It also became apparent that the instances of permission creep have been identified due to the similarity of the user's permission distribution with users of similar roles; however, irregular differences were discovered in the additional permissions that are establish to be instances of permission creep. This validates the approach adopted in this paper whereby it is assumed that permission creep can be identified through statistical analysis.

## Conclusion

In this paper, a novel mechanism is presented to identify instances of permission creep in discretionary access control implementations. The paper presents the use of statistical analysis on an extracted model of subject effective permissions. This analysis includes the use of  $\chi^2$  analysis alongside Jenks natural breaks for the unsupervised identification of permission creep instances. The technique is presented, discussed and empirically tested on Microsoft's NTFS permissions. Empirical analysis has demonstrated good scalability, as well as good accuracy



on synthetic systems of different characteristics. Key findings have demonstrated better accuracy where there are fewer instances of permission creep amongst a bigger file system. This is as instances of permission creep become more irregular and statistically easier to identify.

A key finding of this research is that *Creeper* demonstrates an average accuracy of 96% on synthetic datasets and then an average accuracy of 98% on real-world systems. Furthermore, the real-world system analysis demonstrated the significant improvement in accuracy over two other analysis techniques; the first being a human expert, and the second being an earlier analysis technique using a similar, yet distinctly different, technical approach as presented in (Parkinson and Crampton 2016). Although this work provides a novel technique for detecting instances of permission creep in Microsoft NTFS systems, there are limitations and other significant opportunities of research which motivate future work. For example, the use of alternative unsupervised learning techniques to further improve performance and accuracy. Another key area of research is in applying the technique to other widely used access control implementations, including other desktop and server implementations as well as those implemented on mobile platforms.

## Endnotes

- <sup>1</sup> <https://docs.microsoft.com/en-us/sysinternals/downloads/AccessEnum>
- <sup>2</sup> <https://www.quest.com/products/security-explorer/>
- <sup>3</sup> <http://www.permissionsreporter.com/>

## Funding

This work was undertaken during a project funded by the UK's Digital Catapult Researcher in Residency Fellowship programme (Grant Ref: EP/M029263/1). The funding supported the research, development, and empirical testing presented in this paper.

## Availability of data and materials

All experimental datasets, scripts and software are available from the corresponding author upon request.

## Authors' contributions

SP performed the underpinning research and development presented in this paper. SK contributed towards to empirical analysis and paper writing. JB and DS contributed toward technical development (software testing and bug fixing) throughout this project. All authors read and approved the final manuscript.

## Competing interests

The authors declare that they have no competing interests.

## Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 11 May 2018 Accepted: 29 March 2019

Published online: 15 April 2019

## References

Aas K, Eikvil L. (1999) Text categorisation: A survey. Technical Report 941, Norwegian Computing Center

- Agarwal B, Bhagwan R, Das T, Eswaran S, Padmanabhan V. N., Voelker G. M. (2009) Netprints: Diagnosing home network misconfigurations using shared knowledge. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009. USENIX Association 2009. p 16
- Balakrishnan N., Voinov V., Nikulin M. S. (2013) Chi-squared Goodness of Fit Tests with Applications. Elsevier Science, Amsterdam
- Bartel A., Klein J., Monperrus M., Le Traon Y. (2014) Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing android. IEEE Trans Softw Eng 40(6):617–632
- Bauer L., Garriss S., Reiter M. K. (2011) Detecting and resolving policy misconfigurations in access-control systems. ACM Trans Inf Syst Secur (TISSEC) 14(1):2
- Benantar M. (2006) Access Control Systems: Security, Identity Management and Trust Models. 1st edn. Springer, Cham
- Das T., Bhagwan R., Naldurg P. (2010) Baaz: A system for detecting access control misconfigurations. In: USENIX Security Symposium. USENIX Association, Washington DC. pp 161–176
- De Capitani di Vimercati S., Paraboschi S., Samarati P. (2003) Access control: principles and solutions. Softw: Pract Experience 33(5):397–421. <https://doi.org/10.1002/spe.513>
- Fang Z., Han W., Li Y. (2014) Permission based android security: Issues and countermeasures. Comput Secur 43:205–218
- Greenwood P. E. (1996) A Guide to Chi-squared Testing, vol 280. Wiley, Hoboken
- Jenks G. F. (1967) The data model concept in statistical mapping. Int Yearb Cartogr 7(1):186–190
- Noseevich G., Petukhov A. (2011) Detecting insufficient access control in web applications. In: SysSec Workshop (SysSec), 2011 First. IEEE. pp 11–18
- Ou X., Govindavajhala S., Appel A. W. (2005) Mulval: A logic-based network security analyzer. In: USENIX Security Symposium. USENIX Association is the Advanced Computing Systems Association, Baltimore. pp 8–8
- Parkinson S. (2017) Use of access control to minimise ransomware impact. Netw Secur 2017(7):5–8
- Parkinson S., Crampton A. (2016) Identification of irregularities and allocation suggestion of relative file system permissions. J Inf Secur Appl 30:27–39. <https://doi.org/10.1016/j.jisa.2016.04.004>
- Parkinson S., Crampton A., Hill R. (2018) Guide to Vulnerability Analysis for Computer Networks and Systems: An Artificial Intelligence Approach. Computer Communications and Networks. Springer, Cham. <https://doi.org/10.1007/978-3-319-92624-7>
- Parkinson S., Somaraki V., Ward R. (2016) Auditing file system permissions using association rule mining. Expert Syst Appl 55:274–283. <https://doi.org/10.1016/j.eswa.2016.02.027>
- Pfleeger C. P., Pfleeger S. L. (2002) Security in Computing. 4th edn. Prentice Hall Professional Technical Reference, Upper Saddle River
- Sandhu R. S., Coyne E. J., Feinstein H. L., Youman C. E. (1996) Role-based access control models. Computer 29(2):38–47
- Sbirlea D., Burke M. G., Guarnieri S., Pistoia M., Sarkar V. (2013) Automatic detection of inter-application permission leaks in android applications. IBM J Res Dev 57(6):10–1
- Shaikh R. A., Adi K., Logrippo L. (2017) A data classification method for inconsistency and incompleteness detection in access control policy sets. Int J Inf Secur 16(1):91–113
- Slavin R., Wang X., Hosseini M. B., Hester J., Krishnan R., Bhatia J., Breaux T. D., Niu J. (2016) Toward a framework for detecting privacy policy violations in android application code. In: Proceedings of the 38th International Conference on Software Engineering. ACM, New York. pp 25–36
- Vidas T., Christin N., Cranor L. (2011) Curbing android permission creep. Proc Web 2:91–96
- Yang Y., Pedersen J. O. (1997) A comparative study on feature selection in text categorization. In: International Conference on Machine Learning (ICML). pp 412–420
- Ye N., Chen Q. (2001) An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. Qual Reliab Eng Int 17(2):105–112. <https://doi.org/10.1002/qre.392>
- Yuan Y., Huang T. (2005) A matrix algorithm for mining association rules. Adv Intell Comput 3644:370–379