Automated Planning for Administrating Role-Based Access Control

Simon Parkinson and **Saad Khan**
Department of Computer Science
School of Computing and Engineering
University of Huddersfield, UK
firstname.surname@hud.ac.uk

Lukáš Chrpa
Faculty of Electrical Engineering
Czech Technical University in Prague
Faculty of Mathematics and Physics
Charles University in Prague
chrpaluk@fel.cvut.cz

Abstract

Understanding how to implement change in security controls is heavily reliant on expert knowledge, both that intrinsic to how a system can be configured as well as how a current configuration is structured. Maintaining the required level of expertise in fast changing environments, where frequent configuration changes are implemented, can be challenging. The accuracy of a new access control permissions is essential, as inadvertently assigning rights that result in a higher than necessary level of access can introduce unintended vulnerabilities. To address these issues, a novel mechanism is devised for suggesting how to implement new access control rules, based on historic allocations and the use of Automated Planning. Throughout this paper, we focus on Microsoft's NTFS file system permissions as a continuing application example. The plans are evaluated in terms of their validity as well as the reduction in required expert knowledge.

Introduction

Access control is an integral security mechanism in multi-user computing environments, where there is a necessity to restrict user access to system resources (Sandhu and Samarati 1994). The assignment of permissions is a challenging process in large-scale systems, requiring constant maintenance (Cárdenas, Amin, and Sastry 2008). In a risk-averse organisation, every employee should have a precise set of permissions that does not exceed or fall behind the level of access required to perform their tasks. To fulfil this condition, access control management requires extensive knowledge and experience, which may not be readily available in all organisations. In addition, there is also the possibility of human errors that can lead towards creating or modifying permissions inaccurately.

Performing frequent maintenance on access control systems requires the administrator to have detailed knowledge on the structure of the access control policy and how new additions can be made, retaining the structure. The process of performing administrative tasks can be likened to *goal-based* deliberation whereby the administrator is pursing tasks to either prove or dismiss an hypothesis, which can be reduced to a discrete sequence of investigative actions, each with positive and negative consequences. It can therefore be established that system administration is a deliberation process whereby expert knowledge is required to determine the investigative hypothesis (i.e. the goal), as well as the investigative actions to be performed and their order. There is similarity here with Automated Planning (AP) which encompasses the study and development of deliberation tools that take as input a problem specification and knowledge about the domain in the form of an abstract model of actions (Ghallab, Nau, and Traverso 2004).

In many instances, users will acquire permission through group allocation, and a challenge is in the selection of the correct group to minimise any additional impact (e.g. over elevation of permissions). It may well be that only a single action (e.g. adding to one group) is needed to allocate the correct level of permission; however, all the possible actions and their effects need to be considered to determine the most suitable selection. In this paper, we focus on the process of examining available system information, which is essential for making future access control changes. We focus on developing a technique that does not rely on expert knowledge, and is capable of automating the deliberation process.

The paper is structured as follows: first a discussion of closely related work is provided, followed by a modelling section where an administrative process is modelled and its applicability to AP is presented. A section is then provided discussing the exploitation of the model using the Planning Domain Definition Language (PDDL). This leads towards the presentation of file system administration as an AP problem, and finally, we perform the empirical analysis to demonstrate and evaluate the developed system.

Related Work

The assignment of permissions is a challenging process in large-scale systems. It is an expensive job and requires constant maintenance (Cárdenas, Amin, and Sastry 2008). In a risk-averse organisation, every employee should have a precise set of permissions that does not exceed or fall behind the level of access required to perform their tasks. To fulfil this condition, access control management requires extensive knowledge and experience (Bauer et al. 2009), which may not be readily available in all organisations. In addition, there is also the possibility of human errors that can lead towards creating or modifying permissions inac-

curately. In existing solutions, software aids help to reduce the reliance on human expert for system configuration, auditing, and administrative processes (Al-Shaer, Ou, and Xie 2013). In the particular domain of file system administration, researchers have produced software solutions to assist with administration. For example, in using statistical analysis and instance-based learning to identify anomalies and suggest future allocations (Parkinson and Crampton 2016; Parkinson et al. 2019).

Previous work has witnessed successful exploration of the use of AP in different cyber security domains, mainly for developing attack plans for penetration testing (Boddy et al. 2005). The development of automated penetration testing has received such wide attention mostly because of the growing size and complexity of IT systems, which all require auditing for vulnerabilities. Performing this process manually is a resource intensive task and has a high potential margin of error (Steinmetz 2016). It also requires extensive knowledge for testing and understanding the results. Many studies have been conducted in this area. Riabov et al. present a technique where courses of action are generated based upon a system configuration (Riabov et al. 2016); however, the goal is adversarial in that the aim is to compromise the system, albeit by a trusted security professional. Current research also presents continued development of AP for penetration testing (Shmaryahu 2016) discussing the need to overcome scalability limitations. Researchers are also working on using AP for determining security threat mitigation plans for minimising attacker success (Backes et al. 2017; Khan and Parkinson 2018).

UbuntuWorld (Chakraborti et al. 2017) uses reinforcement learning to learn responses for the Ubuntu system to develop into a automated technical support system, which includes administration tasks. The solution demonstrates positive results from learning questions/responses from Ask Ubuntu. However, a key difference with the work presented in this paper is that the UbuntuWorld system is learning general responses for the Ubuntu system, and not those specific to an organisation's configuration.

It has been identified that there is a wealth of research in analysing Administrative Role-Based Access Control (ARBAC) systems. In one work, the authors utilise planning in the Artificial Intelligence Automated Planning for reachability analysis (Sasturkar et al. 2006; Stoller et al. 2007). The research is focused on developing algorithms to performing reachability, bounded reachability, and availability analysis. In their work, they are focusing on Administrative Role-Based Access Control (ARBAC), which allows for the formalisation of decentralised administration. Their work is based on an adaptation (named `miniARBAC`) of the ARBAC97, omitting the role-to-role administration requirement. The authors identify that analysis for ARBAC is PSPACE-complete and provide an implementation in SAS+ encoding to exploit automated planning (Bäckström and Nebel 1995)

Modelling

The process of performing an access control administrative action is naturally aligned to the abstract model of Automated Planning. The process is where a sequence of actions is performed and results in the transition of the system from an initial state, s_1, to goal state, g, where the system is operating with the desired configuration. In pursuing the goal, a state-transition system is a 3-tuple $\sum = (S, A, \rightarrow)$ where $S = (s_1, s_2, \dots)$ is a finite set of states, $A = (a_1, a_2, \dots)$ is a finite set of actions, and $\rightarrow$: $S \times A \rightarrow S$ is the transition function. A solution P is a sequence of actions $(a_1, a_2, \dots, a_k)$ corresponding to a sequence of state transitions $(s_1, s_2, \dots, s_k)$ such that $s_1 =\!\!\rightarrow (s_0, a_1), \dots, s_k =\!\!\rightarrow (s_{k-1}, a_k)$, and s_k is the goal state. A system's configuration is represented by a set of first-order predicates which are subsequently modified through the execution of each action, $a = \{pre+, pre-, \mathit{eff}+, \mathit{eff}-\}$, where $pre+$ and $pre-$ are sets of predicates representing positive and negative preconditions, i.e., what must and must not be true prior action application. Similarly, $\mathit{eff}+$ and $\mathit{eff}-$ are sets of predicates representing action's positive and negative effects, i.e., what becomes true or false after action application.

As example to demonstrate the relationship between AP and the presented application, consider a simplistic example where the administrator is required to modify the system configuration allowing bob to have $write$ access on $dir1$. The administrator will use an *assign* action to allocate the desired set of permissions, thus translating the system configuration via an *assign* action. Here we assume that the *assign* action has the precondition of *(exist S)*, and an effect of *permission(S,P,O)*, where S is the subject (*bob*), P is the permission level to be assigned (*write*), and O is the object (*dir1*), corresponding to the standardised definitions of Role-Based Access Control (Sandhu et al. 1996). This example is trivial as only one action is necessary to achieve the desired goal; however, in administrative tasks there is often a higher number of actions that can be executed to manipulate the system's configuration.

Domain Model Construction

The Planning Domain Definition Language (PDDL) is used, enabling the application of state-of-the-art domain-independent tools, aiding to provide higher quality solutions, and second, producing PDDL files creates the potential for the AP community to have a new application for bench-marking purposes. In this paper, we use PDDL 2.1 (Edelkamp and Hoffmann 2004). In the file system domain, we introduce the types of: *user*, *group*, *permission* and *directory*. These types instantiate the objects of file system domain. The domain contains six actions, which are used to control the creation of users, groups, and permissions as well as assigning relationships amongst the objects.

There are many different ways by which an administrator may implement new access control rules. Figure 1 provides an illustration of the domain actions and the order in which they can occur. It would be typical to see the ordering of the administrator creating a user, assigning any individual permissions, and creating or adding the user to any groups. This ordering contains ordering constraints observed by the authors. For example, that permissions are added to a group before a adding a user to a group. I.e, a role has been constructed before users are assigned. However, as there is little

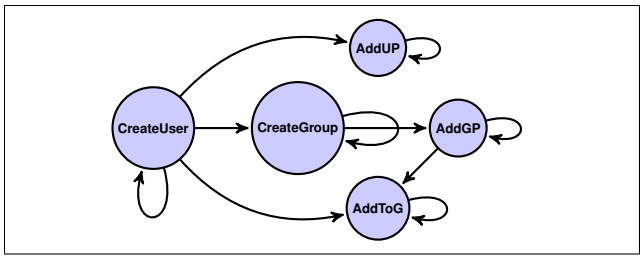

Figure 1: Diagrammatic illustration of the time-line of actions in the model. Key: AddToG = AddToGroup, AddUP = AddUserPermission, AddGP = AddGroupPermission

certainty over the way that the administrator may implement access control permissions, it is necessary to develop a sufficiently flexible domain model, capable of modelling many different orderings.

The following actions have been defined to account for the administrative actions performed during file system administration tasks (illustrated in Figure 1) in Role-Based Access Control (RBAC) models (Sandhu et al. 1996). Each action has a precondition and effect. The precondition is used to describe the state of the underlying system (in terms of objects and predicates) that is necessary for the action to be valid. The effect is used to model the outcome of performing the action in question on the system, and is also modelled in terms of objects and predicates. In the below actions, *u*, *g*, *p*, and *d* represent user, group, permission, and directory objects, respectively.

- `Create-user` is used to model an administrator creating a new user. The action has a precondition of $\neg$*user_exists(u)* and an effect of $\neg$*user_needed(u)* $\wedge$ *user_exists(u)*, where *u* is the new user to be created.
- `Create-group` is used to model an administrator creating a new group. The action has a precondition of *group_needed(g)* $\wedge$ $\neg$*group_exists(g)* and an effect of *group_exists(g)* $\wedge$ $\neg$*group_needed(g)*, where *g* is the new group to be created.
- `Add-user-to-group` is used to encode the process of assigning a user to be a member of a group. The precondition is *user_exists(u)* $\wedge$ *group_exists(g)* $\wedge$ $\neg$*user_needed(u)* $\wedge$ $\neg$*member_of(u, g)* and has an effect of *member_of(u, g)*. Here *u* represents the user been to be added to group *g*.
- `Add-user-permissions` is used to model the process of assigning permissions to a user. The precondition is *user_exists(u)* $\wedge$ $\neg$*user_needed(u)* $\wedge\neg$*u_permission(p, u, d)* and has an effect of *u_permission(p, u, d)*. Here *p* is the permission to be assigned to user *u* on directory *d*.
- `Add-group-permissions` is used to encode the process of assigning a group permission. The precondition is *add_to_group(g)* $\wedge$ $\neg$*group_exists(g)* $\wedge$ $\neg$*g_permission(p, g, d)* and has an effect of *g_permission(p, g, d)* $\wedge$ $\neg$*add_to_group(g)* . Here *p* is the permission attribute to be assigned to group *g* on directory *d*.
- `Calculate-effective` is used to model an assertion of effective permission of a user. The action has a precondition of *u_permission(p, u, d)* $\vee$ *(member_of(u, g)* $\wedge$ *g_permission(p, g, d))* and an effect of *effective_permission(u, p, d)*, where *p* is the permission attribute to be assigned to user *u* on directory *d*.

The `Calculate-effective` action is introduced to assert the effective permission of a user. The use of this action allows the concept of a subject's effective permission to be modelled as an effect and to be used as a requirement in the goal state. It asserts the effective permission, if a user has explicit permission on a directory, or they are acquiring permissions through a set of group memberships. To prioritise exploiting group permissions against assigning individual permissions, the cost of `Add-user-permissions` is higher than other actions, except `Calculate-effective` which has zero cost.

Problem Instance

In order to correctly model the creation of users and groups, as well as their involvement in the allocation of new permissions, it is necessary to extract and understand previous allocations. The solution presented in this paper takes as input the set of system event logs, and outputs the sets of objects and predicates necessary to construct the PDDL problem instance. Information stored within an event is used to convert the administrator's action into the problem instance. Note that such event logs are recorded by the built-in operating system functionality. As illustrated in Figure 2, the event holds the following essential information highlighted in bold: the subject (user or group), the object (directory), the object's old permissions, and the the object's new permissions. The permissions are represented in the Security Descriptor Definition Language (SDDL), but the individual permission attributes can easily be extracted[1]. Translating the event demonstrated in Figure 2 results in the construction of an *assign* action with a precondition of bob not having permission p on object o and an effect of bob having p on o. The new SSDL details the new permission *(A;OICIID;FW;;;bob)* added to the directory.

Software has been produced to process each event from the operating system log to determine their type through analysing the event ID. Once an event of interest has been identified, the event description is then processed to identify key information. For example, an event detailing the creation of a new user ($id = 4720$) contains the user account name.

Plan Generation

A sequence of actions is then identified which translates the system's security configuration from the initial state to the desired goal. For example, consider the situation where a new user (`USER3`) is required with Full Control to a `DPRT2` directory, and also permissions acquired through `ROLE5` to a `DPRT2G` directory. The produced plan will ensure that the following three actions are required to achieve the goal:

[1]Microsoft's SDDL language allows an Access Control List to be represented as a single string of characters `https://msdn.microsoft.com/en-us/library/windows/desktop/aa379567(v=vs.85).aspx`

```
Permissions on an object were changed.
Subject:
     Security ID:    admin
     Account Name:   BOB
     Account Domain: AD
     Logon ID: 0x9B3EC
Object:
     Object Server:  Security
     Object Name:    D:\
     Handle ID:      0x5bc
Process:
     Process ID:     0x1820
Permissions Change:
     Original Security Descriptor:
D:(A;OICI;FA;;;SY)(A;OICI;FA;;;BA)
     New Security Descriptor:
D:ARAI(A;OICIID;FA;;;SY)(A;OICIID;FA;;;BA)
(A;OICIID;FW;;;bob)
```

Figure 2: Example event showing the change in security permission for *bob* on *d*. The new permission level is that *bob* now has write ("FW" for FileWrite) on *d*

```
0:(CREATE-USER USER3)
1:(ADD-USER-PERMISSION USER3 FULLCONTROL DPRT2)
2:(ADD-USER-TO-GROUP USER3 ROLE5)
4:(CALCULATE-EFFECTIVE USER3 FULLCONTROL DPRT2)
5:(CALCULATE-EFFECTIVE USER3 FULLCONTROL DPRT2G
ROLE5)
```

To achieve the goal state, five actions have been identified, which are used to inform the administrator what they should perform the administrative actions to complete the task. First, the `CREATE-USER` is selected so that the user `User3` is created. Next the `ADD-USER-PERMISSION` action is selected to provide `USER3` `FULLCONTROL` on the `DPRT2` directory. Following this, the `ADD-GROUP-PERMISSION` action is selected, enabling `USER3` to acquire Full Control over the department directory (`DPRT2G`) by inheriting the group permissions of `ROLE5`. The use of the `CALCULATE-EFFECTIVE` action is used to satisfy the goal condition in the problem instance.

Empirical Analysis

A case study is now provided where both the ability to acquire problem instances, and the ability to provide plans on how to make configuration additions, are empirically evaluated on event logs generated from assigning an access control policy acquired from an end-user and collaborator of this research. The constructed problem instances are then used alongside the presented domain model to suggest new permission allocations. The suggestions are subsequently evaluated by an experienced (greater than 15 years experience) and independent security practitioner for suitability, as well as using built-in operating system functionality to compare the implemented permission against the level requested.

Environment and Methodology

We select the use of the LPG (Gerevini, Saetti, and Serina 2003) planner for its good support of PDDL language, and general good performance (Roberts and Howe 2009), and the availability and ease of configuration[2]. All experiments presented in this paper were performed on an Intel i7 3.50GHz processor running Ubuntu 16.04 LTS with 32GB RAM. Ten experimental systems of increasing complexity are generated with varying number of users, departments, and roles. Problem instance 1 has 50 users, 1 department, and 1 role. In each subsequent problem instance, the number of users is increased by 50, departments by 1, and also roles by 1, until problem instance 100 which contains 500 users, 10 departments, and 10 roles. This variation creates 10 problem instances that are representative of access control systems found in real organisations. In terms of testing, we use the domain model and introduce a problem instance for the following two use-cases:

a: Suggesting a new group membership allocation when creating a new permission entry allowing for previous group memberships to be utilised.

b: Introducing two new roles (group and directory), assigning the required membership (1 Read and Write, 1 Read), and introduce two new users.

The motivation for choosing these two tasks is that they are both common challenges that administrators face on a daily basis. Problems of type **a** are typically required when a user needs to acquire new permissions on another resource, and that resource already has suitable permissions implemented through group memberships. An example of this is when a user is changing job role and requiring permission on resources that the new role can access. Problems of type **b** represent where a new role is created within a department, which includes the creation of the directory, groups, adding of users to the group, and the assignment of permission on the directory structure.

It is worth noting that the generation of synthetic permissions for testing is common practise in access control research, where it is difficult to utilise benchmark data sets from previous research due to differences in architecture and implementation model. Furthermore, due to the security sensitive nature of the information held within the data sets, organisations are reluctant to share real-world data sets. In previous research, either parameter-based random data sets (Gal-Oz, Gonen, and Gudes 2019; Zhang et al. 2017; El Hadj et al. 2018; Talukdar et al. 2017), those based on business structures (Xu and Stoller 2014), healthcare (Ayache et al. 2016; Mocanu et al. 2015), or those representing a university system (Yang et al. 2015).

Results

The results from performing the experimental analysis are presented in Table 1. The results detail the construction phase describing the number of extracted events, number of

[2]LPG's MAX_RELEVANT_FACTS limit was increased to 40000 and MAX_TYPE_INTERSECTIONS to 10000 to accommodate problem instances with a high number of objects

Problem No.	Problem Instance Construction				Search Time (s)		Memory Consumption (GB)	
	No. of permissions *in*	No. of objects *out*	No. of facts *out*	Execution time *(s)*	*a*	*b*	*a*	*b*
1	57	42	48	0.27	1.21	1.24	0.24	0.26
2	107	63	98	0.49	2.12	2.33	0.78	0.82
3	150	84	141	0.74	3.09	3.12	1.25	1.31
4	179	105	170	1.11	4.06	5.09	1.79	1.83
5	232	126	223	1.27	6.23	6.41	2.37	2.44
6	274	147	265	1.38	6.06	6.88	3.05	3.14
7	304	168	295	1.67	7.0	8.27	3.78	3.85
8	346	189	337	2.79	9.23	10.44	4.54	4.68
9	396	210	387	2.72	9.46	11.32	5.43	5.56
10	425	231	416	3.14	11.99	13.71	6.39	6.50

Table 1: Empirical results from performing both problem instance construction and plan suggestion. Problem instance **a** is where a new permission allocation is required, and **b** is where a new role is required. Construction informs the quantity of both input (**in**) and outputs (**out**) as well as processing time

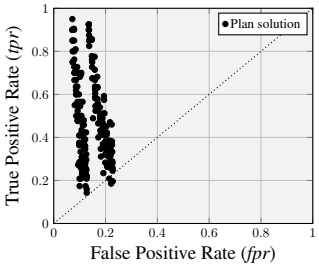

Figure 3: TPR & FPR

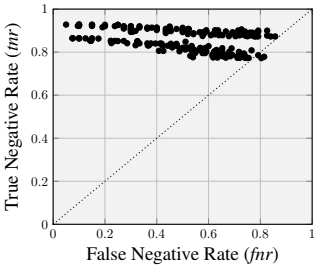

Figure 4: TNR & FNR

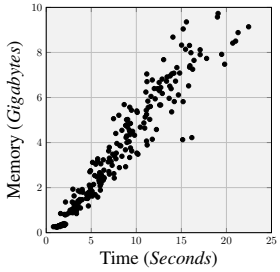

Figure 5: Time & Memory

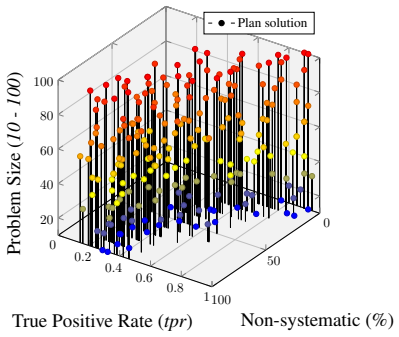

Figure 6: TPR & *ad hoc*

objects, and the number of predicates. From the table it is noticeable that the performance of the system decreased along with the number of initial events, resulting in a total of 3 seconds required for problem construction for the largest problem instance (10). This is significant as the short processing time is of little inconvenience to the user. Furthermore, the table also illustrates search time and memory consumption. It is evident that the required time to identify a valid plan (Search Time) is at maximum 13.7 seconds, and the maximum memory consumption reaches 6.5 GB. There is slight difference between the two different problem instances. For example, problem instance 10 requires 6.3 and 6.5 seconds for instance **a** (one allocation action) and **b** (2 $\times$ new group, 2 $\times$ permission, and 4 $\times$ group assignment). This time is of little inconvenience to the user and delivers the following significant benefits:

- It is a reasonable assumption that manual analysis will take considerably longer than the longest search time of 6.5 seconds. More specifically, viewing permissions on a per directory level would take at least a few seconds to view and understand permissions for one directory.
- More significantly is the potential for new allocations to be identified that are systematic with previous allocations. During manual analysis, the expert is likely to identify a way of implementing permissions which they believe is most suitable. However, there might be an unforeseen problem with their approach, or equally as significant, they have not identified a way of adhering to the structure of current allocations.

During search, many solutions are identified. For example, for problem instance "3" (150 users, 3 departments, 3 roles) there are 3 valid solutions for problem type "a" and 12 for "b". All solutions identified by the planner are valid; however many are sub-optimal in that they invoke the creation of groups when not necessary. A correct allocation is one whereby the effective permission meets the required permission. An example valid output for plan **b** is:

```
0: (CREATE-GROUP GROUP-ROLE-1)
0: (CREATE-GROUP GROUP-ROLE-2)
1: (ADD-GROUP-PERMISSIONS GROUP-ROLE-1
FILE-READ-DATA DIR1)
1: (ADD-GROUP-PERMISSIONS GROUP-ROLE-2
FILE-WRITE-DATA DIR2)
2: (ADD-USER-TO-GROUP USER1 GROUP-ROLE-1)
2: (ADD-USER-TO-GROUP USER2 GROUP-ROLE-1)
2: (ADD-USER-TO-GROUP USER1 GROUP-ROLE-2)
2: (ADD-USER-TO-GROUP USER2 GROUP-ROLE-2)
3: (CALCULATE-EFFECTIVE USER1
FILE-READ-DATA DIR1 GROUP-ROLE-1)
```

```
3: (CALCULATE-EFFECTIVE USER1
FILE-WRITE-DATA DIR2 GROUP-ROLE-2)
3: (CALCULATE-EFFECTIVE USER2
FILE-READ-DATA DIR1 GROUP-ROLE-1)
3: (CALCULATE-EFFECTIVE USER2
FILE-WRITE-DATA DIR2 GROUP-ROLE-2)
```

In the above example, it can be seen that two new groups are firstly created, followed by the allocation of two new permissions. The first is where `GROUP-ROLE-1` is assigned `FILE-READ-DATA` on directory `DIR1`. The second is where `GROUP-ROLE-2` is assigned `FILE-WRITE-DATA` on `DIR2`. The final stage of the plan is where the two users (`USER1` and `USER2`) are assigned to both `GROUP-ROLE-1` and `GROUP-ROLE-2`. This solution demonstrates the potential of using the encoded domain model and problem instances to suggest new allocations and no suitable permissions exist through group memberships.

Sensitivity

The results presented in the previous section demonstrate the suitability of the technique to suggest new access control allocations that utilise previous group allocations, suggesting the creation of new groups, if required. However, the underlying system used in this analysis is well-structured. More specifically, each user, group and permission level has been created and applied in a systematic manner following a clear policy. It is often the case that *ad hoc* permissions (i.e. those that do not follow the systematic structure) will frequently be made when undertaking administrative actions and the structure in relation to the policy will diminish. This section investigates the relationship between the degree of systematic structure and the impact on the technique's ability to suggest meaningful allocations. The purpose of this analysis is to establish how well the technique performs on systems of different size and structure, and thus account for variation found in live systems.

Experimental analysis is performed by taking the previous 10 problem instances and increasing the number of *ad hoc* permissions, thus simulating a diminishing structure. A percentage of additional and non-systematic allocation are introduced (i.e. 0%, 10%, ..., 100%) to represent those that do not follow previous systematic allocations. Ad hoc permissions are simulated by assigning a pseudo-random permission entry (user, directory, permission). This results in the construction of 220 problem instances.

During analysis, solutions plans are manually evaluated by a subject expert to determine their feasibility. More specifically, the following measures are used: True Positive Rate (*tpr*) is the fraction of valid administrative actions that are correctly included in the plan; False Positive Rate (*fpr = 1 - tnr*) is the fraction of valid administrative actions that have not been included in the plan; True Negative Rate (*tnr*) is the fraction of invalid administration actions that are correctly not included in the plan; False Negative Rate (*fnr = 1-tpr*) is the fraction of invalid administration actions that are incorrectly suggested in this plan; and Finally, the *accuracy* is reported as the fraction of all suggestion actions that are correctly identified and valid.

Discussion

Figure 3 illustrates the relationship between the *tpr* and *fpr*. It shows that the system results in a high *tpr* and a low *fpr*, meaning that the system is mostly capable of finding a suitable valid plan, with a the *tpr* decreasing in accordance with problem size. This is of significance as it demonstrates the ability of the system to suggest administrative actions that provide the user the required level of permission. Figure 3 also illustrates that there are two distinct clusters of plan solutions. The left cluster is type **a** problems and right cluster is type **b**. Furthermore, Figure 6 illustrates non-systematic (also referred to as ad hoc) permissions percentage (x-axis), *tpr* (y-axis) and problem sizes (z-axis). It is clear that *tpr* is inversely proportional to the percentage of non-systematic permissions and the same pattern can be observed in all problem sizes.

Figure 4 shows that the *tnr* is high while the *fnr* is low, which is of significance as it demonstrates the technique's ability to not suggest incorrect actions. Figure 4 also presents the two distinct clusters of plan solutions, as seen in Figure 3. The top cluster is of type **a** problem instances with relatively high *tnr* and bottom cluster is of type **b**. Figure 5 illustrates the relationship between memory consumption and computation time. From analysing the results, it is evident that larger problem instances require more memory and time (10GB, 24 seconds) to solve. Problem instances that hold information to represent a larger underlying system will contain a larger number of objects and predicates. For example, increasing the number of user objects also increases the amount of `user-exists` and `u-member-of` predicates, hence affecting the overall size of problem instances. In addition, the increased size of problem instances require more memory to hold the problem representation.

Conclusion

In this paper, we present a domain model for performing permissions allocation, followed by an discussion of how generic automated planning approaches can be utilised to provide assistive automation when planning for new permissions. The results demonstrate the applicability of the approach and the reduction in reliance on expert knowledge. This is of significance as it enables people with less specialised expertise to make configuration changes without adversely affecting the system. The proposed system is tested with different sizes of problem instances and ad hoc permissions percentages. Another important aspect of this research is that the tool enables the adherence to previously allocated permissions, where possible. This ensures that administrators do not make more allocations than are necessary to maintain consistence, and also minimise the potential to introduce unforeseen vulnerabilities. The average accuracy of the system is found as 85%.

In future work we plan to perform a larger experimental analysis through identifying suitable collaborative organisations and analyse the results from live systems. We further plan to perform extended end-user testing to determine the applicability of the generated plans.

References

[Al-Shaer, Ou, and Xie 2013] Al-Shaer, E.; Ou, X.; and Xie, G. 2013. *Automated security management*. Springer.

[Ayache et al. 2016] Ayache, M.; Erradi, M.; Khoumsi, A.; and Freisleben, B. 2016. Analysis and verification of xacml policies in a medical cloud environment. *Scalable Computing: Practice and Experience* 17(3):189–206.

[Backes et al. 2017] Backes, M.; Hoffmann, J.; Künnemann, R.; Speicher, P.; and Steinmetz, M. 2017. Simulated penetration testing and mitigation analysis. *arXiv preprint arXiv:1705.05088*.

[Bäckström and Nebel 1995] Bäckström, C., and Nebel, B. 1995. Complexity results for sas+ planning. *Computational Intelligence* 11(4):625–655.

[Bauer et al. 2009] Bauer, L.; Cranor, L. F.; Reeder, R. W.; Reiter, M. K.; and Vaniea, K. 2009. Real life challenges in access-control management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 899–908. ACM.

[Boddy et al. 2005] Boddy, M. S.; Gohde, J.; Haigh, T.; and Harp, S. A. 2005. Course of action generation for cyber security using classical planning. In *ICAPS*, 12–21.

[Cárdenas, Amin, and Sastry 2008] Cárdenas, A. A.; Amin, S.; and Sastry, S. 2008. Research challenges for the security of control systems. In *HotSec*.

[Chakraborti et al. 2017] Chakraborti, T.; Talamadupula, K.; Fadnis, K. P.; Campbell, M.; and Kambhampati, S. 2017. Ubuntuworld 1.0 lts-a platform for automated problem solving & troubleshooting in the ubuntu os. In *AAAI*, 4657–4663.

[Edelkamp and Hoffmann 2004] Edelkamp, S., and Hoffmann, J. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universitat Freiburg, Institut fur Informatik.

[El Hadj et al. 2018] El Hadj, M. A.; Khoumsi, A.; Benkaouz, Y.; and Erradi, M. 2018. Formal approach to detect and resolve anomalies while clustering abac policies. *ICST Trans. Security Safety* 5(16):e3.

[Gal-Oz, Gonen, and Gudes 2019] Gal-Oz, N.; Gonen, Y.; and Gudes, E. 2019. Mining meaningful and rare roles from web application usage patterns. *Computers & Security* 82:296–313.

[Gerevini, Saetti, and Serina 2003] Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in lpg. *Journal of Artificial Intelligence Research* 20:239–290.

[Ghallab, Nau, and Traverso 2004] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

[Khan and Parkinson 2018] Khan, S., and Parkinson, S. 2018. Eliciting and utilising knowledge for security event log analysis: an association rule mining and automated planning approach. *Expert Systems with Applications* 113:116–127.

[Mocanu et al. 2015] Mocanu, D.; Turkmen, F.; Liotta, A.; et al. 2015. Towards abac policy mining from logs with deep learning. In *proc. of the 18th International Multiconference, IS2015*, 124–128.

[Parkinson and Crampton 2016] Parkinson, S., and Crampton, A. 2016. Identification of irregularities and allocation suggestion of relative file system permissions. *Journal of Information Security and Applications*.

[Parkinson et al. 2019] Parkinson, S.; Khan, S.; Bray, J.; and Shreef, D. 2019. Creeper: a tool for detecting permission creep in file system access controls. *Cybersecurity* 2(1):14.

[Riabov et al. 2016] Riabov, A.; Sohrabi, S.; Udrea, O.; and Hassanzadeh, O. 2016. Efficient high quality plan exploration for network security. In *International Scheduling and Planning Applications woRKshop (SPARK)*.

[Roberts and Howe 2009] Roberts, M., and Howe, A. 2009. Learning from planner performance. *Artificial Intelligence* 173(5):536–561.

[Sandhu and Samarati 1994] Sandhu, R. S., and Samarati, P. 1994. Access control: principle and practice. *IEEE communications magazine* 32(9):40–48.

[Sandhu et al. 1996] Sandhu, R. S.; Coyne, E. J.; Feinstein, H. L.; and Youman, C. E. 1996. Role-based access control models. *Computer* 29(2):38–47.

[Sasturkar et al. 2006] Sasturkar, A.; Yang, P.; Stoller, S. D.; and Ramakrishnan, C. 2006. Policy analysis for administrative role based access control. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 13–pp. IEEE.

[Shmaryahu 2016] Shmaryahu, D. 2016. Constructing plan trees for simulated penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*.

[Steinmetz 2016] Steinmetz, M. 2016. Critical constrained planning and an application to network penetration testing. In *The 26th International Conference on Automated Planning and Scheduling*, 141.

[Stoller et al. 2007] Stoller, S. D.; Yang, P.; Ramakrishnan, C. R.; and Gofman, M. I. 2007. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM conference on Computer and communications security*, 445–455. ACM.

[Talukdar et al. 2017] Talukdar, T.; Batra, G.; Vaidya, J.; Atluri, V.; and Sural, S. 2017. Efficient bottom-up mining of attribute based access control policies. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, 339–348. IEEE.

[Xu and Stoller 2014] Xu, Z., and Stoller, S. D. 2014. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing* 12(5):533–545.

[Yang et al. 2015] Yang, P.; Gofman, M. I.; Stoller, S. D.; and Yang, Z. 2015. Policy analysis for administrative role based access control without separate administration. *Journal of Computer Security* 23(1):1–29.

[Zhang et al. 2017] Zhang, A.; Ji, C.; Bao, Y.; and Li, X. 2017. Conflict analysis and detection based on model checking for spatial access control policy. *Tsinghua Science and Technology* 22(5):478–488.