To Study the Performance of ACL (Access Control Model) to Identify the Security Issues and to resolve the Problem of Assign Authorization to Object Through SSO (Single Sign-On)

Dharmendra Choukse
Institute of Engg & Science
IPS Academy
Indore, India

Umesh Kumar Singh
Institute of Comp. Science
Vikram University
Ujjain, India

ABSTRACT

Access control models to identify the benefits and security challenges associated with them and then also discusses how can reduce the complexity of Web services development through access control model to resolve the identified issues. Information Security involves the activities that organisations, enterprises, and institutions undertake to defend the value and continuing usability of assets, the integrity, and continuity of operations. The term Access Control really mentions to the control over access to system resources after a user's account credentials and identity have been authenticated and access to the system granted. Several Data, Access Control models, have been introduced by keeping in view the requirements of an organisation, and the sensitivity of the data.

Keywords
Webservices,ACL,SSO

1. INTRODUCTION

A. Mandatory Access Control (MAC):

Mandatory Access Control utilises hard-coded security rules. Rules are coded into an application or operating system [1]. The security policy is centrally controlled and can be overridden by the users, and it is functional to various properties, objects, and requests. The data organisation of MAC security policy begins with complex, undisclosed, and private, and next to the organization of resources that will be making demands for data. MAC concept is integrated mostly in military and governmental applications where high-level security is essential.

The benefit of this model is that the rules are hard-coded into the software, so there are decidedly fewer chances of an administrative error or social engineering.

B. Discretionary Access Control (DAC)

Discretionary Access Control can be used as a centralised and distributed model [1]. DAC centralised model is administered by an administrator or a team of administrators, who are answerable to make security policies and allocate privileges as per policy, but this approach is time-consuming, particularly if the administrator is off or outsourced. In the distributed method, the data access is distributed to some answerable person such as managers, supervisors, or team. This method provides a way to avoid delays in the administration of accounts is dispersed.

C. Role-Based Access Control (RBAC):

In this modest atmosphere, the risk of dropping info is more for leading organisation. MAC and DAC model secure data, but they have boundaries. To overcome their shortcomings, RBAC has been proposed [1]. The common architecture of Role-based Access Control is shown in figure 1.

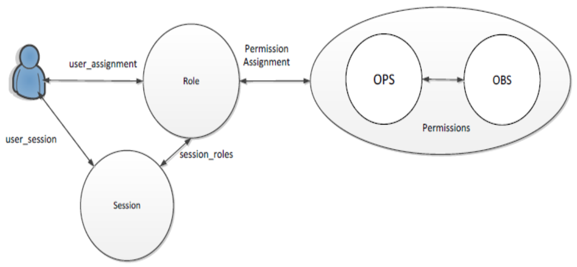

Figure 1: Role-based Access Control Architecture

As Role Based Access controls are in presence in last 20 years, particularly in UNIX and mainframe environments, but they lack some principles as each system use its own propriety elements. There was a need to design such a system which is standardised, scalable, logical in design, and non-system dependent.

RBAC0 is the first proposed model in this series, this model consists of separation of duties and providing lowest privileges to each role. It doesn't have the order mechanism, so the permissions were allocated directly to the users within a specific role or function. By considering the need of order as it exists in any organisation such as Administrator, Manager, and team members RBAC1 is presented based on RBAC0. It provides a standard distribution of tasks within an institute that is usually layered as senior and junior roles. This covered security distribution method is suitable for big atmospheres.

Constraints are presented in RBAC2 which offer more control over any network in large atmospheres. Constraints help to implement the policies while not having the order. Constraints work as limiters and ensure that the policies are being enforced. For instance, if an institute wants to give administrative rights to one user or role, the constraints confirms that only one user has the system administration rights.

D. Extensible Access Control Markup Language (XACML):

The Extensible Access Control Markup Language (XACML) is a general-purpose language for specifying access control policies [2]. In XML terms, it describes a core schema with a namespace that can be used to direct access control and authorisation policies for XML objects. Then it is based on XML, it is, as its name proposes, easily extensible. XACML supports a broad range of security policies [1], and uses a consistent syntax for arranging requests so that any one of the following responses to an access request will be effective:

- Permit: action allowable
- Deny: action disallowed
- Indeterminate: error or incorrect/missing value prevents a decision
- Not applicable: the request cannot be processed.

As shown in figure 2, XACML's standardised architecture for this decision making uses two primary mechanisms: the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). The PEP creates the request based on the user's attributes, the resource requested, the action specified, and other situation-dependent information through Policy Information Point (PIP). The PDP receives the constructed request, compares it with the applicable policy and system state through the Policy Access Point (PAP), and then returns one of the four responses specified above to the PEP. The PEP then allows or denies access to the resource. The PEP and PDP mechanisms may be embedded within a single application or may be distributed across a network.

In instruction to make the PEP and PDP work, XACML provides a policy set, which is a container that grips either a policy or other policy sets, plus links to other policies. Each policy is stated using a set of rules. Conflicts are resolved through policy-combining algorithms. XACML also includes methods of merging these policies and policy sets, permitting some to dominate others. This is needed because the policies may overlap or conflict. For example, a simple policy-combining algorithm is "Deny Overwrites," which causes the final choice to be "Deny" if any policy outcomes in an "Overwrite." Equally, other rules could be established to allow an action if any of a set of policies results in "Allow." Determining what policy or policy set to apply is accomplished using the "Target" component. A target is a set of rules or conditions applied to each subject, object, and operation. When a rule's conditions are met for a user (subject), object, operation combination, its associated policy or policy set is applied using the process described above.

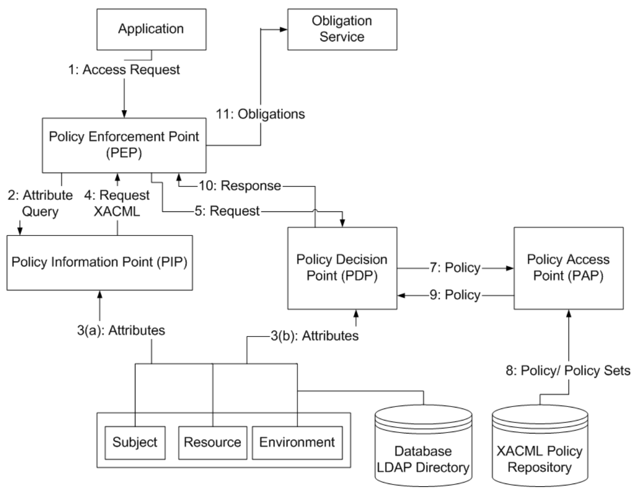

Figure 2: XACML's Standardized Architecture

The associated access control data for a specified enterprise domain can then be programmed in an XML document, and the conformance of data to the enterprise access control model can be obtained by validating the XML document against the XML schema that represents the enterprise access control model using XML parsers. These XML parsers are based on standard application programming interfaces such as the Document Object Model (DOM), and the parser libraries are implemented in various procedural languages to enable an application program to create, maintain, and retrieve XML-encoded data. Although XML-based and other access control languages provide capabilities for composing policies from scratch, allowing users to specify access control policies, together with the authorisations through the programming of the language, they lack a formal specification language for access control constraints (like historical based and domain constraints) that prevent assigning overlapping privileges. As an example, consider the case of constraints that require the manipulation and recording of access states (such as granted privileges). This is to avoid creating situations that result in users who were previously denied access to certain files being unknowingly granted access in a future state. Like most access control languages, XACML does not provide tools for the expression of historical constraints for historical-based access control policies, thus leaving the completeness of the constraint logic to the policy writer.

Domain constraints are based on the semantic information pertaining to an enterprise context; a grammar-based language cannot deal with content-based constraints. So, an XML schema is insufficient for a complete specification of the RBAC model for an enterprise since the latter contains content-based domain constraints. An example is not allowing more than one user to be assigned to the role of "security administrator" (role cardinality constraint) and not allowing the roles "viewer" and "uploader" to be assigned to the same user (separation-of-duty constraint).

Here, again we note as before that the specification languages assume a static environment where changes in access control policies are generally effected manually by a security administrator. So in essence, although XML-based access control languages provide features that enable them to specify a broad range of policies, a formal specification is still needed to define constraint rules adaptively.

2. LITERATURE SURVEY

The research has been done in the field of SSO to find out different solutions to secure the web services. These solutions are based on the different authentication schemes like Kerberos, X.509, SAML SSO and a single password. Driven by the demand for suitable security for Web services, there are several groups working on an extension of the basic Web service specifications [3]. The first browser-based authentication protocol was, to our best knowledge, Microsoft Passport [4]. Because the protocol is not published, we only refer to Microsoft's whitepapers such as [5]. The work has been done on the password based authentication; the protocol proposed a single sign-on protocol for distributed web applications based on standard internet mechanisms [6].

3. LIMITATION OF THE EXISTING SYSTEMS

E. Mandatory Access Control (MAC):

The limitation of this model is that:

1) The rules are hardcoded, so it takes interval to review
2) If the requirements change, then we need to modify the rules
3) MAC is best right for a group of users with related needs, so it is not common for all.

F. Discretionary Access Control (DAC):

The limitation of this model is that:

1) The consistency of data access for end-user with same job functions can be diminished as access to data is distributed at the discretion of the owner.

2) It is a time-consuming approach.

G. Role-Based Access Control (RBAC):

The limitation of this model is that:

1) The main disadvantage of RBAC is related to the role explosion: due to the growing number of dissimilar (real world) roles (sometimes differences are only very minor) we need a growing number of (RBAC) roles to correctly encapsulate the permissions (a permission in RBAC is an action/operation on an object/entity). Handling all those roles can become a complex affair.

2) It is also not very well suited to manage individual rights, but this is typically deemed less of a problem.

H. Extensible Access Control Markup Language (XACML):Problem with the existing system

1) The current system is identity-centric, i.e., it focuses on the user identity, the user role, and optionally the user group

2) Usually entirely managed by the Identity and Access Management (IAM) teams like Oracle Identity Manager, CyberArk, and IBM.

3) Administrator-time: roles and permissions are assigned at administration time and live for the duration they are provisioned for.

4. CONSTRUCTING RBAC BASED TEST MODEL

RBAC testing involves testing of role-permission assignments (i.e., rules) and testing of user-role assignments with SSOD and DSOD constraints. We present two methods for constructing role-permission test models, discuss modelling of user-role assignments, and describe the analysis of test models. Building Role-Permission Test Models from Functional Test Models RBAC rules are security constraints on system functions. If a functional test model is already available, we can integrate into it RBAC rules as constraints for access control testing. In our work, we demonstrated that test models nets could be used to build test models for automated functional testing of various applications. One approach to building a functional test model as a test model net (referred to as functional net) is to formulate a test design (or workflow) by using the building blocks of test model nets, including sequence, condition, repetition, concurrency, and modularity/hierarchy. This is similar to programming, which transforms a program design into code by using the building blocks (sequence, if-then-else, for/while/repeat, multi-threading, classes/ function calls) of the given programming language.

The RBAC model follows the more general representation of role permission assignments (i.e., RBAC rules to be defined below). It consists of the following elements:

- A set of roles R,
- A role hierarchy H□R×R, a partial order relation on R. <r1, r2> denotes that r1 is a direct super-role of r2 or r2 is a direct sub-role of r1 (r2 inherits all permissions of r1),
- A set of subjects/users (human or computer agents) Sub,
- Role assignments Sub-›2R (one subject can play a set of roles),
- A set of constraints on static separation of duties:
- SSOD□R×R, where <r1, r2>∈ SSOD means that r1and r2 cannot be assigned to the same subject,
- A set of constraints on dynamic separation of duties:
- DSOD□R×R, where <r1, r2>∈ DSOD means that r1 and r2 assigned to the same subject cannot be activated within the same session, and
- A set of role permission/prohibition rules R.

Let O be a set of objects (or resources), A be a set of operations (called activities related to the resources), C be a set of contexts (representing Boolean expression constraints, for instance, temporal contexts, location-based context, etc.), and {Permission, Prohibition} be a set of authorization types.

In a library management system (LMS), for example, the set of roles is {student, teacher, director, secretary, admin, borrower, personnel}, the role order is {<borrower, student>, <borrower, teacher>, <personnel, director>, <personnel, secretary>} (borrower is the super role of student, whereas teacher and personnel is the super-role of director and secretary), SSOD ={<borrower, personnel>, < admin, borrower>}, DSOD={<admin, director>}, the set of objects is {book, borrower Account, personnelAccount}, and the set of activities is {BorrowBook, ReserveBook, GiveBackBook, AdminActivity, ManageAccess, CreateAccount, ModifyAccount, DeliverBook, FixBook}, and the set of contexts is {day(WD), day(HD), day (MD)}, where WD, HD, and MD refer to working day, holiday, and maintenance day, respectively. In Table 1, rules 1-6 are specified for the borrower role. day(HD) can also be understood as day(d) and d=HD, where d is a variable.

According to law 1, a borrower is not permitted to give back books on holidays. According to rule 3, a borrower is allowed to borrow books on working days. Given a set of specified RBAC rules, there can be situations under which neither permission nor prohibition is specified. We treat these situations as "undefined conditions" and extend the set of authorisation types to {Permission, Prohibition, and Undefined}. From security assurance perspective, the undefined conditions must be tested because they likely lead to security holes in an implementation.

To generate tests for these conditions, test modelling needs to cover both defined and undefined access control conditions. Our approach can automatically find such undefined conditions for a given set of RBAC rules.

In the following, we discuss rare examples. In Table 1, rules 1-6 are the specified access control conditions whereas rules 7-10 are added according to the undefined conditions. Among the specified rules 1-6, rules 2 and 3 are the only ones that are related to activity BorrowBook for the borrower. Their contexts are a day(HD) and day(WD). They do not cover maintenance days (MD) – whether a borrower can borrow books on maintenance days is not defined. This rule 7 is added. This is similar to ReserveBook (rule 8) and GiveBackBook (rule 9). Consider FixBook for the borrower. There is no specified rule for FixBook under any context because it is a responsibility of secretary.

From a testing perspective, we need to test whether a borrower is allowed to perform FixBook. Thus we add rule 10, where day(d) is true for any d ϵ{HD, WD, MD}. Applying all activities to each role may require many rules to complete the specification. To deal with the complexity, our method allows tests to be generated concerning several coverage standards and can reduce the search space by using partial ordering and pairwise combination techniques.

Table 1 Rbac Rules For The Borrower/Student Role

No.	Object	Activity	Context	Auth _Type
1	Book	GiveBackBook	day(HD)	Prohibition
2	Book	BorrowBook	day(HD)	Prohibition
3	Book	BorrowBook	day(WD)	Permission
4	Book	GiveBackBook	day(WD)	Permission
5	Book	ReserveBook	day(HD)	Prohibition
6	Book	ReserveBook	day(WD)	Permission
7	Book	BorrowBook	day(MD)	Undefined
8	Book	ReserveBook	day(MD)	Undefined
9	Book	GiveBackBook	day(MD)	Undefined
10	Book	FixBook	day(d)	Undefined

Handling of role hierarchies in our approach will be discussed below. Constrained RBAC uses static and dynamic constraints to deal with separation of duties. In our approach, SSOD and DSOD specify the pairs of roles that cannot be assigned or activated together. Symmetric RBAC adds the notion of role permission review, which allows determining permissions of operations on objects assigned to specific roles.

This is addressed by RBAC rules defined roles, operations, and objects. As a more general formalism of permission specification, the RBAC rules also allow the specification of access contexts and prohibitions (i.e., negative permissions).

In a role hierarchy, each role r inherits all permissions (i.e., RBAC rules) from its super roles. Let S(r) be the set of all super-roles of role r, and (r) be the set of all rules concerning r, including the rules defined for r and its super roles. (r) = {<r, o, a, c, t> : <r, o, a, c, t>ϵR} {<r', o, a, c, t> : <r', o, a, c, t>R ϵ r' and S(r)}. Here, we use (r) to build role-permission test models that involve role r.

In the above LMS example, student, as a sub-role of the borrower, inherits all the RBAC rules in Table 1. These rules will be used to build the role permission test model for the student as a running example.

Let us consider building a functional net for a subset of the student role activities in LMS – borrow, reserve, and return the book. A student may borrow an available book and return a borrowed book. This consists of a sequence of two activities. A student may reserve a book and then borrow it. This is also a sequence of activities. When a borrowed book is returned, it can be borrowed again this implies a loop structure. When interested in a book, a student may borrow or reserve the book this is a conditioning structure. Putting the above structures together would result in the functional net shown in Fig. 3.

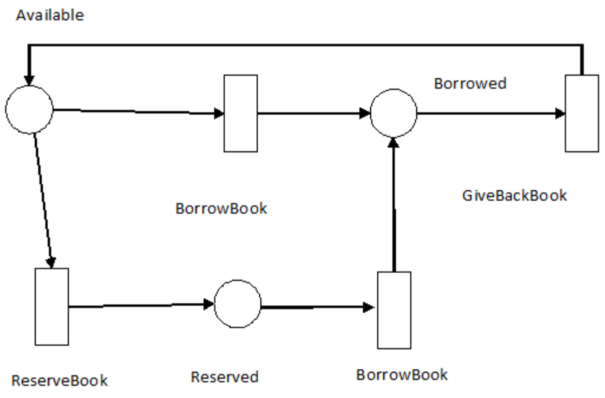

Figure. 3. Test Model net of student activities in LMS

Now we discuss how to build a role-permission test model by integrating RBAC rules into a functional net.

For the sake of easiness, let us first assume that a functional net involves the activities of a single role. An RBAC rule <r, o, a, c, t> is related to a functional net only if activity appears as a transition in the functional net. For example, rules 1-9 in Table 1 for the activities BorrowBook, GiveBackBook, and ReserveBook are related to the functional net in Fig. 3. Suppose the RBAC rules in (r) related to a functional net are <r, o1, a1, c1, t1>, <r, o2, a2, c2, t2>, …, <r, om, am, cm, tm>. We integrate each RBAC rule <r, oi, ai, ci, ti>

(1≤i≤m) into the functional net as follows:

If ti=Permission and ci=true. Nothing is needed in that the rule is already represented by the activity transition ai,

If ti=Permission and predicates in ci have corresponding places in the net (new places may be created for the predicates if necessary). Add a bi-directional arc between each place, and the activity transition ai (because the access does not change the context) and add ci to the guard of ai. Consider rule 3 in Table 1 – the student is allowed to borrow book only on a working day. The context day(d) and d=WD becomes an additional constraint of borrow books in the functional net. As shown in Fig. 4, we create a place day, add a bi-directional arc between place day and transition BorrowBook, label the arc with variable d, and add d=WD to the guard condition of BorrowBook

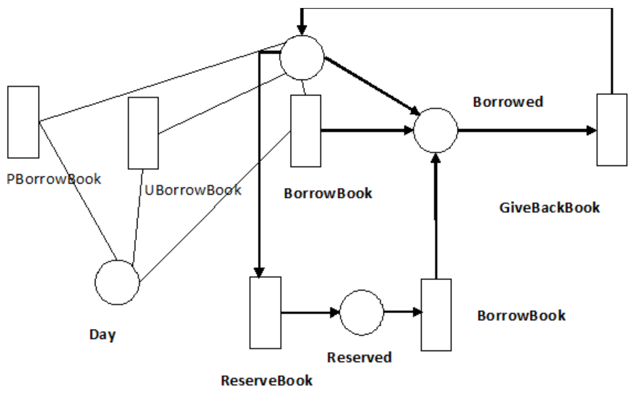

Figure. 4. Test permission model of student activities in LMS.

If ti = Prohibition, we add to the net a new transition Pai, which means ai is prohibited. This transition shares the input and inhibitor places (i.e., functional preconditions) with ai. If the predicates in ci are corresponding to places, we add a bi-directional arc between each place and transition Pai (because it does not change the context) and add ci to the guard of ai. Consider rule 2 in Table 1 – the student is not allowed to borrow books on holiday.

This is represented by transition PBorrowBook in Fig. 4.

If ti=Undefined, we handle the same way as Prohibition except that the new transition is named Pai. Consider rule 7 in Table 1 – borrowing books on a maintenance day is undefined. This is represented by transition UBorrowBook in Fig. 4. The role-permission test model in Fig. 4 results from integrating rules 2, 3, and 7 in Table 1 into the functional net in Fig. 3. Other rules can be handled similarly except for rule 10 whose activity Fixbook does not appear in the functional net in Fig. 3. Since Fixbook is an activity of the secretary role, we can integrate rule 10 into the functional net of secretary. To represent multiple roles in a model, we use a global place role, which is connected to each transition with a bi-directional arc labelled by a role variable <r>. Which role can or cannot perform an activity is then represented by such a guard condition as r=R or r!=R, Where R is a particular role. For example, rule 10 can be integrated into the secretary test model by using a new transition UFixBook, whose guard includes r=student. This transition means that fixed book is undefined for the student.

Building User-Role Assignment Test Models

A test model of user-role assignments specifies the test requirements related to assigning/assigning users to roles and activating/deactivating roles assigned to users. The assignment and activation must satisfy the static and dynamic constraints on separation of duties, i.e., SSOD and DSOD. As shown in Fig. 5, test model nets can be used to formalize the above test requirements.

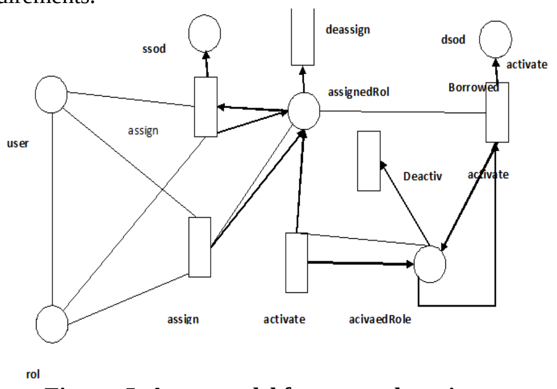

Figure. 5. A test model for user-role assignments.

In Fig. 5, places user and role represent users and roles, respectively. Places assigned role and activatedRole represent the roles that are assigned to users and the roles that are activated, respectively.

Places ssod and dsod represent the role pairs in SSOD and DSOD, respectively. Two "assign" transitions intend to assign roles to users. The lower "assign" transition assigns role r2 to user u if u is not yet assigned to any role. The upper "assign" transition assigns role r2 to user u which already plays role r1 only when <r1, r2> ϵ SSOD (i.e., the inhibitor arc from ssod to assign) and r1≠r2 (i.e., the guard condition). Similarly, the lower "activate" transition activates role r2 assigned to user u when u has no activated role yet. The upper "activate" transition activates role r2 assigned to user u when u has an activated role r1, r1 r2, and <r1, r2> ϵ DSOD. In addition, transitions design and deactivate remove role assignment and activation relations, respectively.

5. SIMULATION AND RESULT ANALYSIS

Our case studies are based on three Dot net application program, LMS (Library Management System). Table 2 presents the main parameters of these programs. LMS offers services to achieve books in a public library. The books can be borrowed and refunded by the users of the library on working days. LMS differentiates three types of users: public users who can borrow five books for three weeks, students who can borrow ten books for three weeks and teachers who can borrow ten books for two months.

Table 2 Subjects Of The Empirical Studies

Subject	LOC	#Classes/Methods	#R	#O	#A	#Rules
LMS	3204	62/335	5	4	12	33

LMS is managed by an administrator who can create, modify, and remove user accounts. Books in the library are managed by a secretary who orders books or adds them when they are distributed. The secretary can also fix the broken books on certain days dedicated to maintenance. When a book is damaged, it must be fixed. While it is being fixed, this book cannot be borrowed, but a user can reserve it. The director of the library has the similar accesses than the secretary and can consult the accounts of the employees. The administrator and the secretary can refer all user accounts. All users can consult the list of books in the library.

The results of our research are summarized in Table 3. For LMS, there were 207 test cases in 3,185 lines of code. 56.2% of the test code was generated. The tests killed 233 out of 243 mutants, with an overall detection rate of 95.9%. The ten outstanding mutants not killed by the tests have the same environment – they contain a new rule created by the adding-rule operative but can never because security harms because the useful prerequisite of the activity in the added rule is not satisfied. These mutants do not violate the required security policies. Consider a mutant with the following added rule that allows the admin role to return books on any day: (admin, Book, GiveBackBook, true, Permission). According to the required access control policies, none of the Borrower's activities, BorrowBook, ReserveBook, and GiveBackBook, is intended for use by the admin role (no access control rules concerning these activities are specified for admin). The above-added rule can never allow the admin role to return books because of the prerequisite of GiveBackBook- "the book is borrowed" (by the same person) - is unsatisfiable. This prerequisite can only be fulfilled by BorrowBook. In the mutant, however, Admin is not able to borrow books (BorrowBook is undefined for admin). It is worth pointing out that our approach killed the mutant with the following added rule that allows admin to borrow books: (admin, Book, BorrowBook, true, Permission).

As part of our primary experiment, the request for transition reporting to the student role in LMS only killed about 50% of the mutants because many access contexts were not exercised.

Table 3 Results Of The Empirical Studies

	#T	LOC	GLOC	%GLOC	#M	# K	Score
LMS	207	3185	1789	56.2%	243	233	95.9%

#T: number of test cases generated; LOC: lines of executable Dot net test code; GLOC: lines of Dot net test code; %GLOC: percentage of

Dot net test code; #M: number of access control mutants; #K: number of mutants killed by the generated test cases; Score: mutation score = #K/#M.

In the literature on RBAC specification and analysis, the number of roles and depth of role hierarchies is important factors for measuring the complexity and scalability of RBAC systems. For a complex real-world RBAC system with a large number of roles and a deep role hierarchy, our approach relies on the “divide and conquer” strategy and builds a number of test models to deal with subsets of roles and access control rules (rather than a single model for all roles and rules). Building test models (e.g., contracts and functional models) is essentially a manual process. It is also different from system modelling for design and verification. The former focuses on what needs to be tested with carefully selected test data, whereas the latter often deals with system-wide behaviours and input spaces. Thus, the complexity and scalability of test generation for individual test models in our approach is not directly related to the total number of roles and the depth of role hierarchy in the SUT. Instead, it depends on the number of access control rules, number of objects, number of activities, number of access contexts, and test data involved in a given test model. In theory, the complexity of our approach for reachability coverage is exponential to the sizes of these factors because it aims to cover every possible state transition. These factors determine the number of states and state transitions in the model.

The main outcome of our study is that our approach is highly effective in detecting access control defects. The key features that have led to this result include formalization of function nets and contracts, generation of access control tests with the reachability graph coverage, generation of executable test code, and mutation analysis of access control rules. In the following, we debate how these aspects can be affected when our approach is applied to general software applications where access control is an important security mechanism.

6. SUMMARY

We have presented the tool-supported, model-based approach to automated conformance testing of RBAC policies. It provides structured processes for building role-permission test models from functional nets and contract specifications. It also automatically generates executable access control tests from the test models. The empirical studies using Dot net based application programs have demonstrated that our approach is highly effective in detecting access control defects and that 56% - 82% of the executable test code is generated automatically. This study has focused on the testing of role-permission assignments and user-role assignments in RBAC, where users, roles, and permission rules are predefined.

7. REFERENCES

[1] R. Bhatti, J. B. D. Joshi, E. Bertino, A. Ghafoor, “Access Control in Dynamic XML-based Web-Services with XRBAC”, In Proceedings of The First International Conference on Web Services, Las Vegas, June 23-26, 2003.

[2] The eXtensible Access Control Markup Language (XACML), Version 3.0, OASIS Standard, January 22, 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-osen.pdf

[3] M. Hillenbrand, J. G. otze, J. Muller, and P. Muller, "A Single Sign-On Framework for Web-Services based Distributed Applications", 8th International Conference on Telecommunications, 2005, pp 273-279.

[4] T. Grob, "Security Analysis of the SAML Single Sign-on Browser/Artifact Profile", Proceedings of the 19th Annual Computer Security Applications Conference, 2003, pp 298-307.

[5] Microsoft. .net passport review guide, 2002.

[6] J. Gantner, A. G. Schulz, and A. Thede, "A Single Sign-On Protocol for Distributed Web Applications Based on Standard Internet Mechanisms", e-business and telecommunications networks Springer Netherland, 2006, pp I53-I59.

[7] T. Fleury, J. Basney, and V. Welch, "Single sign-on for Java web start applications using MyProxy", Proceedings of the 3rd ACM workshop on Secure web services, 2006, pp 95-I0I.