International Journal of Cyber-Security and Digital Forensics (IJCSDF) 1(1):1-14
The Society of Digital Information and Wireless Communications, 2012 (ISSN: 2305-0012)

Rationale and Design of the Access Specification Language RASP

Mark Evered
School of Science and Technology
University of New England
Armidale, 2351, Australia
mevered@une.edu.au

ABSTRACT

In this paper we describe the formal specification language RASP for expressing fine-grained access control constraints in information systems. The design of the language is motivated by a number of IS case studies which demonstrate the complexity of the access constraints which arise if minimal (need-to-know) access is to be strictly enforced. RASP supports modularity, parameterization, role acquisition, constraint expressions and a symmetrical approach to role transitions and attribute transitions. No existing access control specification language supports all of these complex, realistic requirements.

KEYWORDS

security, access control, specification, roles, attributes

1 INTRODUCTION

In general, each of the users of an information system needs to be able to view or manipulate only some of the information stored in the system. Ideally, the appropriate access for each user will be specified in the form of an access policy during the analysis phase of the software development and then enforced via access control mechanisms during the execution of the implemented system.

An access control specification language of this kind can be used for a variety of purposes. These include:

- **Analysis**: The specification language can capture the requirements for a new information system and be used to validate the design.
- **Generation of test cases**: The formal specification can be used to generate test cases for verifying whether a new system fulfils the access control requirements.
- **Testing of existing systems**: The desired access requirements can be formulated for an information system already in existence so that test cases can be generated to determine whether the access control is adequate.
- **Generation of access control code**: In some cases, the access control code may be able to be generated automatically from the specification.
- **Proof of access control properties**: A formal specification can serve as the basis for formal proofs that certain security or privacy properties are satisfied by a system which implements that specification.

An access control policy can be understood as consisting of two components. The first is control over the membership of the subject groups of interest in the application domain. The second is a mapping from each of these groups to permissions which allow certain operations to be performed on the data by members of the groups.

In database systems this second component has traditionally been restricted to a simple read/write permission on the fields of the records. An object-based approach has the benefit of allowing access to be based on the permission to execute methods corresponding to high-level operations on objects meaningful in the application domain. This object-based approach dates back to [8] and has now been integrated into most common component models.

The first component of access control has also been approached in a number of different ways. In the simplest case, an access control list for each object contains an entry for each subject or group of subjects. The owner of the object (or a system administrator) can assign subjects to groups. More recently, Role Based Access Control models have been defined which allow this first component of access control to be based on the roles played by individuals in the organisations making use of an information system. This means that there is a (dynamic) mapping from subjects to roles and then a (relatively static) mapping from roles to permissions. These models recognise the complex nature of permissions in real organisations and have been shown to subsume both conventional discretionary access control models and mandatory access control models such as Bell-LaPadula [1].

A number of different RBAC models have been proposed with varying degrees of additional complexity. These additions include role hierarchies [13], parameterized roles [7], constraints on role acquisition [15] and roles for objects and environments as well as for subjects [3]. As with any kind of model, there is a tension here between simplicity on the one hand and expressive power on the other. A lack of expressive power in an overly simple security specification model may lead to either compromises in the specification of the required security or to specifications which are artificially long and complex and therefore difficult to validate with stakeholders. Since both of these outcomes are undesirable, we argue here for expressive power over simplicity as long as the specification language itself does not become too complex to validate and verify as a result. This is of course a somewhat subjective assessment and must therefore be informed by real case studies.

In this paper we use four case studies to illustrate requirements for an access control specification language which has sufficient expressive power to adequately support the principle of least privilege for complex information systems such as Health Information Systems. The more comprehensive of these is an information system for managing the data associated with residents at an aged-care facility. This case study was used in a previous paper [6] to investigate requirements for access control *mechanisms* but the focus here is on an abstract access control specification language. This case study was chosen because of the complex constraints which arise from what appears to be a relatively simple system if the principle of least privilege is taken seriously. In this it is typical of the complexities which occur in the area of access control for Health Information Systems.

In the following section we describe the two case studies and some of the complex access control constraints which they imply. The third section constitutes the bulk of the paper and generalizes the constraints to identify requirements for an adequate access control specification language. The RASP (Role and Attribute-based Specification of Protection) notation for each of the requirements is given. The fourth section compares this work to related work on role-based access specification languages and shows that no existing specification language is sufficient in providing support for all of the requirements. We conclude with a summary of the findings and contributions of the paper.

2 ACCESS CONSTRAINTS

2.1 Case Studies

1. Aged-care facility

The information system in this case study manages the personal, financial, clinical and medical data of the residents at a small aged-care facility in rural New South Wales. Subjects include staff such as the manager, administrative staff, health care workers and volunteers and visiting professionals such as doctors and physiotherapists. A particular doctor is assigned to each resident but in unforeseen circumstances, a different doctor can be allowed access to the medical data of a resident. Residents themselves are also subjects since they have access to the information stored about them.

Access rules must cover the normal day-to-day running of the facility but also the admittance procedure for new residents, the death or departure of a resident and also emergency situations. In emergency situations, it is essential that the access rules in force for normal situations do not prevent relevant information from being accessed.

Some access is dependent on signed statements from subjects. This includes a statement of consent from residents and a confidentiality agreement from staff. In a fully paperless system, these signatures, and the witnessing of them, can be realized as operations within the information system.

For further details on this system, see [6].

2. eSteps

This is a system for the development of questionnaires for health data, the collection of the data on mobile devices and the accumulation of the data at a central point. The system also allows the importing and exporting of data in the standard EpiData format and the tranmission of the data to other parties for use in analysis.

The system was developed in conjunction with the WHO and has been used in the collection of information on non-communicable diseases. The original system was developed without consideration for access control issues and the case study involves a post-priori definition the constraints required for the system, including various levels of confidentiality requirements.

For further details on this system, see [16].

3. Multi-level security

This case study involves the classification of documents into security levels from unclassified to top-secret and access to the documents by subjects granted a certain level of security clearance. It is based on the well-known Bell-LaPadula model [1].

The main constraints are for permission to view only documents at a level equal to or below the level of clearance and permission to modify and create only documents at a level equal to or above the level of clearance (the *-property). ‘Trusted subjects’ are not restricted by the *-property. Documents can be declassified by trusted subjects and subjects can be cleared to a higher level by a security manager.

Documents can also be given attributes, such as being associated with a certain project and access is then conditional on subjects being assigned to that project.

4. Electronic funds transfer

This system is for managing a set of accounts for a banking system or for an e-commerce system such as PayPal. The main operation on the data is the transfer of funds from one account to another but there are further operations for administrative purposes. There are also 'deposit' and 'withdraw' operations for transferring funds to and from an external form (such as cash).

The subjects in the system are the owners of accounts and the administrative staff managing the system. As well as transferring funds from his/her own account to another account, an account owner may wish to give another account owner permission to transfer a certain amount to their own account on a regular basis (e.g. monthly).

For further details on this system, see [5] which discusses an access mechanism for the system but does not address the question of an adequate specification language.

2.2 Example Constraints

The following is a list of some of the more challenging access constraints which have arisen from the case studies. Over-simplistic RBAC models are not able to adequately handle constraints such as these.

- The manager of the aged-care facility will generally assign subjects to roles but should not be able to assign just anyone to the role of doctor. This should only be possible if a qualification for the subject is presented and is signed by an appropriate authority.
- A new staff member will initially have a role which enables them to sign a confidentiality agreement. Once they have done so, they can gain the role of 'staff member' but should thereby automatically lose the role of 'applying staff member'. This illustrates the need for role transitions.
- A doctor can be assigned to be the 'doctor for a particular resident' but only with the permission of both the manager and the resident (or the 'responsible person' for the resident).
- A person cannot be both the signatory and the witness to a confidentiality agreement.
- Only one person can be the doctor assigned to a certain resident. This illustrates a uniqueness requirement.
- Health care staff can view the recent medical information of a resident but not older information (realized in this particular aged-care facility as 'more than twelve months old').
- The manager and the resident should be alerted if someone invokes emergency access to the medical information of the resident. This illustrates the need for alerts in the system.
- The manager can delete the information about a resident but not until seven years after the resident has left the facility (or nine years for a resident of indigenous descent).
- Access to a document in a multi-level security system depends not only on the confidentiality level but on the particular patient/case with which the document is associated. This

illustrates the need for object attributes as well as subject roles.

- A subject in a multi-level security system can be raised to a higher clearance level but not dropped to a lower level.
- The documents as well as the subjects must be able to be assigned to new confidentiality levels. This shows the need for dynamic attributes as well as dynamic roles.
- Access to an instance of a questionnaire in the eSteps system is dependent on being the 'data collector' allocated to use the PDA device for that instance.
- An inexperienced staff member in the electronic banking system is only allowed to perform a limited set of tasks and can only perform these tasks with a limited set of parameter values (e.g. transfers of small amounts).

2.3 General Principles

As well as providing the expressive power to convey the above kinds of constraints, an access control specification language should conform to a number of general principles appropriate in all security applications. We consider most of these to be self-evident. A possible exception is the principle of positivity. This will be further discussed later in the paper.

- **Conciseness**: Commonly occurring kinds of constraints should be expressible in a straightforward manner.
- **Clarity**: Ideally it should be apparent at a glance that a constraint is expressing the correct intent.
- **No repetition**: A single constraint or access concept should only need to be expressed in a single place.
- **Modularity**: Since the total set of roles and constraints can be very extensive, it must be possible to formulate them as a combination of smaller, understandable sub-units.
- **Aspect-orientation**: The access control aspect of a system should be expressible in a way which is completely separate from other aspects such as functionality, synchronization and architectural aspects.
- **Positivity**: The constraints should be purely in the form of allowing access, not in the form of denying access. As well as preventing conflicts this prevents situations where the denial of access may be overlooked for some subject or role and therefore unwanted access inadvertently allowed.
- **Reasonability**: The semantics of the access language should be formally defined and allow reasoning about properties of an access control policy formulated in the language.
- **Efficient implementation**: A certain overhead will always be involved in performing access control checks but this overhead should be kept at a reasonable level. If this is not achieved then there will be a temptation to compromise on the degree of security in order to improve system performance.

3 LANGUAGE REQUIREMENTS

Based on our case study examples and general principles for security mechanisms, we now formulate some requirements for an adequate access control specification language. We will consider various aspects of the language definition in turn. We use a form of pseudo-code in the examples to communicate the concepts without the need for a discussion of syntax.

3.1 Modules and Parameterization

One basic question in the design of such a language is the construct or constructs used to group constraints and to form modules. One possibility, for example, would be to offer a construct which groups all constraints relating to a certain role. An alternative would be to offer a construct which groups all constraints relating to a certain object. A module could consist of one such construct or a number of them specified in a single source file.

In the above case studies, we have found that the kind of grouping which provides the clearest description for a certain application is neither strictly role-based nor strictly object-based but is determined by the particular rules and procedures of the organization. So, for example, in the aged-care case study, it is convenient to group most of the constraints associated with health care workers together with other constraints relating to the access required by staff during normal day-to-day operations. Other constraints for health care workers, however, will be grouped together with the constraints describing the special access that subjects should be granted in emergency situations. In other words there will be a module for 'normal operation' and a module for 'emergency situations'.

Because of this need for flexibility in the grouping of constraints, it is best if access constructs are simple clauses which can be freely combined into modules on whatever basis the software engineer determines to be appropriate. Some examples of such clauses in RASP are:

```
allow volunteer!
residents.addNotes;
```

(A 'Volunteer' may invoke the 'addNotes' operation of an object with the 'Residents' attribute.)

and:

```
allow
volunteerActingInEmergency!
residents.getMedicalEntry;
```

(A 'VolunteerActingInEmergency' may invoke the 'getMedicalEntry' operation of an object with the 'Residents' attribute.)

where 'Volunteer' and 'VolunteerActing InEmergency' are assumed to be roles in the system.

A second question relating to the basic constructs of the language is how much of the complexity of the constraints is built into the role acquisition component of the language and how much is expressed as more complex access rules for each role. In the above example, it is assumed that a subject acquires a new role to act in an emergency. Alternatively, we could have used a clause of the form:

```
allow volunteer !
residents.getMedicalEntry
#(isEmergency);
```

(A 'Volunteer' may invoke the 'getMedicalEntry' operation of an object with the 'Residents' attribute if there is an emergency.)

where a condition has been attached to the access clause. In the case studies, we have found that it is clearer both for specification and for auditing purposes to identify distinct roles wherever possible. In fact these often correspond to recognized roles within an organization.

A third basic question is whether roles need to be parameterized. Given that roles need to be defined as precisely as possible in order to accurately reflect the roles used in organizations and ensure minimal access, we have found it necessary that roles can be given parameters. So instead of just:

```
allow doctor!residents.
addMedicalEntry;
```

we need:

```
allow doctor(someone)!
residents.
addMedicalEntry(someone);
```

(The 'Doctor' of 'Someone' may invoke the 'addMedicalEntry' operation for that 'Someone'.)

which parameterizes the role and relates the role parameter value to the value of a parameter of the operation on the object.

3.2 Role Acquisition

In any access control scheme, a fundamental consideration concerns the question of who has the right to assign rights. More specifically, in a role-based scheme, who has the right to assign a subject to a certain role and/or to remove a subject from a role? In the simplest case, this is handled by a manager or even just a system administrator who has full control over access rights. In general, a much finer-grained approach is necessary. For example, it is not appropriate for the manager of the aged-care facility to be able to assign him/herself to the role of doctor.

One possibility for finer-grained control is a delegation-based approach in which a subject with certain rights can pass those rights on to other subjects. This is however not generally desirable. For example, it is not appropriate for a Health Care Worker to assign another person to be a Health Care Worker. That is the job of the manager of the facility.

In general it is necessary to specify for each role who (i.e. members of which role) has the right to appoint subjects to a role. So, for example:

```
appoint manager:someone ->
healthCareWorker;
```

(The 'Manager' can appoint 'Someone' to be a 'HealthCareWorker'.)

This does not yet, however, solve the problem of who can appoint a Doctor. The manager needs to make the appointment but should not be allowed to appoint just anyone. Ideally, we would want a constraint that the subject is qualified as a doctor, i.e. an appropriate certificate from a trusted third party. This idea of appointment combined with certificates was suggested in [15]. We can express this in RASP as:

```
appoint manager,
medicallyQualified{someone}:
someone -> doctor;
```

(The 'Manager' can appoint 'Someone' to be a 'Doctor' in the facility if he/she has a medical qualification certificate.)

where `'medicallyQualified{someone}'` signifies a check for the presence of a digitally-signed certificate from an appropriate authority.

Finally, approval from more than one person may sometimes be required for an appointment. For example:

```
appoint manager,someone:
doctor -> doctor(someone);
```

(The 'Manager' and a 'Someone' together can appoint a 'Doctor' to be the 'Doctor for that Someone'.)

3.3 Constraints and Reasonability

The appointment of a subject to a role may be dependent not just on permission and certificates, but also on further conditions involving parameter values of an operation or the current state of the objects in the system. For example, in a multi-level security scenario, the manager of the system can appoint someone cleared at a certain level to be cleared at a different level, but only if the new level is higher than the old level.

Other appointment constraints involve limits on the time for which the appointment is to remain valid. This may be an absolute time value such as one day or may be specified as lasting for a single session. For example:

```
appoint manager: doctor ->
initialExaminer(someone)
#time(session);
```

These kinds of more complex constraints have advantages and disadvantages for a formal access-control model. They can be used to more accurately and therefore more strictly express the constraints required for a certain information system. On the other hand, they make the model itself more complex and constraints which depend on the state of the system may make it difficult or even impossible to prove properties of the access-control by automatic reasoning. Since our aim is to support the expression of minimal required access, we maintain that such constraints must be allowed. (The consequence is then simply that more properties may be provable for systems which do not require such constraints than for systems which do.)

A further kind of constraint in role-based systems is associated with the notion of conflicts, whether this represents an actual 'conflict of interest' or simply a situation which does not make sense. So, for example, a resident cannot be allowed to witness their own signature.

```
conflict someone,
witnessFor(someone);
```

A certain combination of rules may constitute a conflict statically or dynamically. That is, it may be a conflict for a single person to be appointed to both roles or it may only be a conflict for a person to be acting in both roles in a single session.

A similar kind of constraint is that only a single person should be allowed to have a certain role. Again this may be static or dynamic. For example:

```
unique session manager;
```

(The 'Manager' role can only be adopted by one person at a time.)

3.4 Role Transitions and Hierarchies

A further key question in role acquisition is the relationship between a new role to which a person is being appointed and the roles that person already possesses. Sometimes this simply involves adding a role to the set of roles already possessed. In other cases, the new role may be a specialization of a role already possessed. This allows for role hierarchies. So, for example in:

```
appoint manager: doctor ->
initialExaminer(someone);
```

the general role of 'doctor' is retained but the subject now has the special role as well.

A third possibility not generally supported in RBAC systems is that the new role is replacing a role already possessed. This is the case in:

```
appoint manager:
intensiveCareNurse /->
maternityWardNurse;
```

where '/->' signifies that the former role is lost. An example from a multi-level security system is:

```
appoint accessManager:
secretCleared /->
topSecretCleared;
```

where someone with the role 'secretCleared' is given the role 'topSecretCleared'. In terms of a Bell-LaPadula-type scheme, it is essential that the old role is relinquished at the same time the new role is allocated so that the *-property can be properly enforced.

To accommodate these possibilities, an access control specification language needs two forms of appointment clause: one which specifies that a person with a certain role can gain a new role as well, and one which specifies that a person with a certain role can change to a different role. Note that the former provides for specialization as well as simple addition of roles.

This concept of *changing* roles can be used for other purposes as well. One example is a natural progression within an information system such as from 'ApplyingStaffMember' to 'StaffMember' to 'FormerStaffMember', each of which will have different access rights.

A clause expressing this concept can also be used to *revoke* some of the access rights of a person by changing them to a new role. For example,

```
appoint manager: nurse /->
restrictedNurse;
```

Used in this way, the clause can even be used for removing a role from a person entirely by indicating that the role be changed to a 'null' role as in:

```
appoint
manager:healthCareWorker /->
nobody;
```

It should be noted that this use of role transition obviates the need for 'negative' access rules which list the operations that someone is not allowed to perform.

3.5 Object Groupings

In the simplest case, an access clause will allow someone with a certain role to invoke certain operations on a particular object. If the number of objects in a system is large, however, this can lead to an explosion in the number of access clauses required and in the overhead involved in changing access rights. Often the access allowed for a certain role should be the same for a whole group of objects and it is laborious and error-prone to specify each separately.

In Ponder [4], objects can be grouped into hierarchical folders and access can be specified for all the objects of a folder (and its sub-folders). This is sufficient for some systems but not for others. In a multi-level security system, for example, the security level of a document may be independent of its location in a folder hierarchy. We propose a more general approach in which attributes can be assigned to objects and an access clause specifies the access allowed by a role to all objects with a certain attribute. In fact the attributes possessed by an object can be seen as analogous to the roles possessed by a subject and may require the same level of complexity, including 'appointment', parameters and transitions. So, for example, we may have:

```
appoint accessManager:
someone ->
confidentialClearedStaffMember;

attribute accessManager:
document ->
confidentialDocument;

allow
confidentialClearedStaffMember!
confidentialDocument.read;
```

This example illustrates the three main kinds of RASP clause. A subject is given a role, an object is given an attribute and a subject with a certain role is given access to an operation on objects with a certain attribute.

This similarity in the way roles and attributes are handled for subjects and objects makes the model more uniform and simpler to understand while greatly increasing its expressive power.

4 RELATED WORK

Both the object-based access control paradigm [8] and the role-based access control paradigm [12] are well-known approaches as is the combination of the two to define access to an object in terms of the methods which can be invoked by subjects acting in a certain role. A number of significant extensions to the basic RBAC model have been suggested in order to adequately handle the complexities of minimal access control requirements in real-world scenarios. These include role hierarchies [13] and role parameters [7].

A question which has received much less attention is how to group objects so that the access constraints for the whole group can be specified in a single place rather than repeating them for each and every object. The Ponder policy specification language [4] supports a hierarchical structure of domains and sub-domains of objects similar to a file system hierarchy. The leaves of the tree are references to objects rather than the objects themselves so that an object can appear in a number of different domains. This approach assumes that the domains are relatively static and that an administrator will place objects into domains via some mechanism external to the language. In our case studies, the domains of an object may depend on object attributes which change in the same way that the role of a subject may change. These transitions require the same level of specification as to who can effect the change as is required for role changes. The approach of Generalized Role-Based Access Control [3] recognizes the need for symmetry between subject roles and object roles but does so on the basis of a very simple model which does not support role parameters or pre-conditions for role transitions.

Attribute-based access control (ABAC) [17][18] was developed to support access to web services based on provable attributes of a user rather than the identity of the user. This is important for anonymity in using such services but is not appropriate for organizations or systems where fine-grained access-control policies are based on identity and roles. ABAC has been extended to include attributes for resources as well as subjects but does not address attribute transitions.

A further important question concerns the acquisition of access rights. Ponder is a delegation-based system. It provides for delegation policies which limit which access rights a subject can pass to another subject but the basic assumption is that the possessor of a right decides if and when another subject should gain that right. In our case studies, it is often necessary that access rights be granted by someone who does not possess them him/herself. The OASIS Role Definition Language [15] allows for this kind of appointment-based acquisition of access rights and for role acquisition pre-conditions based on external certificates known as auxiliary credential certificates. OASIS RDL does not however allow for a distinction between the case where a new role is replacing a previous role and the case where the new role is additional. In our case studies, this distinction has been found to be useful both for role transitions and for object attribute transitions. OASIS RDL also does not allow for the generation of new credential certificates as a result of operations performed within the system.

Ponder supports both positive and negative authorizations. In fact, it has two forms of negative access control clause: negative authorization policies and refrain policies. So, for example, a set of access rights can be granted to a group of subjects via a positive authorization policy and then one of the rights can later be revoked from a certain member of the group via a negative authorization policy. Negative authorizations lead to the problem of potential inconsistencies and loopholes in an access control system. A more elegant way to express this kind of partial revocation is to use role transition to transfer a subject from one role into a new role which has a more restricted set of rights.

Generalized Role-Based Access Control proposes that, in addition to subject roles and object roles, there should also be ‘environment roles’ [3] which classify the state of the environment in an access control scenario. So, for example, there may be time or location roles such as ‘Monday’ or ‘downstairs’. This provides a simple form of constraint for RBAC but more complex constraints are often necessary. OASIS RDL allows for the matching of a role parameter with a method parameter before a method is invoked. Ponder allows for the full expressive power of the UML Object Constraint Language [10] in formulating the additional access constraints of ‘when-clauses’. XACML [9] similarly allows constraints to be formulated in terms of expressions which invoke arbitrary object methods of objects in the system. Our case studies confirm that this expressive power is sometimes required for real access control scenarios even though it may hinder formal proofs of security properties.

The access control specification languages and mechanisms described in this section represent the state-of-the-art in fine-grained access control but, as demonstrated above, none of them can support all of the requirements which arise from the case studies.

5 IMPLEMENTATION

A proof-of-concept implementation of the access control specification language RASP is currently being undertaken. This implementation has been developed in the context of a web-based document retrieval system.

A RASP specification describing the roles, attributes and access rules associated with the users and documents is analyzed by a parser implemented using the jjtree compiler-compiler. If error-free, the parse tree is used to generate a Linux directory structure for access to the documents and the access rules are implemented via .htaccess files within the Apache web server.

When a user logs in to the system, he/she initially has only the role ‘someone’. Based on the constraints of the RASP specification rules, the user can then:

- invoke a program to access a document
- explicitly take on further roles
- appoint someone to a role
- label an object with an attribute

The first of these is accomplished by representing the programs as methods in the RASP rules and he last three are implemented as dynamic updates to the Apache access files.

6 CONCLUSION AND FUTURE WORK

This work is based on case studies of information systems which require very fine-grained access control. We have outlined two case studies and given examples of some of the complex access constraints which arise if minimal access is to be guaranteed. We have then generalized these to formulate specific requirements for an access control specification language. We have considered requirements in the areas of modules and parameterization, role acquisition, constraint expressions, role transitions and hierarchies and object groupings.

No existing access control specification language fulfils all the requirements resulting from this analysis and therefore none is adequate in specifying the minimal access rules for the case studies. In particular, a major contribution of this research has been to identify the usefulness of the concept of role transitions and the symmetry between roles and role transitions on the one hand and attributes and attribute transitions on the other.

The RASP specification language allows the formulation of minimal access rights for use in information systems analysis and design and for verification and the formal proof of security and privacy properties. A full syntax of the language can be found at in Appendix A.

Further work includes the completion of the proof-of-concept implementation and validating the specification language through further case studies.

7 REFERENCES

1. D.E. Bell and L.J. La Padula, “Secure computer systems: unified exposition and Multics interpretation”, MTR-2997, The MITRE Corporation, 1975.
2. B. Blobel, “Authorisation and access control for electronic health record systems”, International Journal of Medical Informatics, 73, 2004.
3. M.J. Covington, M.J. Moyer and M. Ahamad, “Generalized role-based access control for securing future applications”, Proc. 23rd National Information Systems Security Conference, Baltimore, 2000.
4. N. Damianou, N. Dulay, E. Lupu and M. Sloman, “Ponder: A language for specifying security and management policies for distributed systems”, The Language Specification Version 2.3, Imperial College Research Report DoC 2000/1, 2000.
5. M. Evered, “Opsis: A distributed object architecture based on bracket capabilities”, Proc. Conference on Technology of Object-Oriented Languages and Systems, Sydney, 2002.
6. M. Evered and S. Bogeholz, “A case study in access control requirements for a health information system”, Proc. Australasian Information Security Workshop, Dunedin, 2004.
7. J.H. Hine, W. Yao, J. Bacon and K. Moody, “An architecture for distributed OASIS services”, Proc. Middleware 2000, Lecture Notes in Computer Science, Vol. 1795, Springer-Verlag, Heidelberg/New York, 2000.
8. A. Jones and B. Liskov, “A language extension for expressing constraints on data access”. Communications of the ACM, 21(5):358-367, May, 1978.
9. T. Moses (Ed.), Extensible Access Control Markup Language (XACML) Version 2.0, OASIS Consortium, 2005.
10. Object Management Group, Object Constraint Language Specification Version 2.0, 2006.
11. J.H. Saltzer, “Protection and the control of information sharing in Multics”, Symposium on Operating System Principles, Yorktown Heights, NY, 1973.
12. R. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, “Role based access control models”, IEEE Computer 29 (2), 1996.
13. R. Sandhu, “Role activation hierarchies”, Proc. 3rd ACM Workshop on Role-Based Access Control, Fairfax, 1998.
14. M.C. Tschantz and S. Krishnamurthi, S. “Towards reasonability properties for access conrol policy languages”, Proc. 11th ACM Symposium on Access Control Models and Technologies, Lake Tahoe, 2006.
15. W. Yao, K. Moody and J. Bacon, “A model of OASIS role-based access control and its support for active security”, ACM Transactions on Information and System Security, 5, 4, 2001.
16. P. Yu and H. Yu, H., “Lessons learned from the practice of mobile health application development”, Proc. 28th Annual

International Computer Software and Applications Conference, Hong Kong, 2004.

17. T. Yu, X. Ma and M. Winslett, “Prunes: an efficient and complete strategy for automated trust negotiation over the internet”, Proc. 7th ACM conference on Computer and communications security.ACM Press, 2000.
18. E. Yuan and J. Tong, “Attributed based access control (ABAC) for web services”, Proc. IEEE International Conference on Web Services, 2005.

Appendix A – Formal syntax of RASP

```
clause: appoint_clause |
        attribute_clause |
        allow_clause |
        conflict_clause |
        unique_clause |
        log_clause


appoint_clause: 'appoint'
        precondition
        {',' precondition } ':'
        role transition role
        [ time ] [ alert ] ';'


precondition: role |
        certificate |
        condition


role: id [ '(' id {',' id} ')' ]


certificate: id '{' id
      {',' id} '}'


transition: '->' | '/->'


time: '#' 'time' '('
      [ constant ] unit ')'


unit: 'session' | 'day' |
      'month' | 'year'


alert: '@' 'alert' role
       { ',' role }


attribute_clause: 'attribute'
       precondition
       {',' precondition } ':'
       attribute transition
       attribute
       [ time ] [ alert ] ';'


attribute: id [ '(' id
       {',' id} ')' ]
```

```
allow_clause: 'allow' role '!'
        action {',' action}
        [condition]
        [ ':' certificate ] ';'


action: attribute operation


operation: '.' id '('
        [ par {',' par} ] ')'


par: id | constant |
     role | 'self'


condition: '#' '(' expression ')'


conflict_clause: 'conflict'
       [ 'session' ]
       role [ ',' role ] ';'


unique_clause: 'unique'
       [ 'session' ] role ';'


log_clause: 'log' [ role '!' ]
       attribute
       [ operation ]
       [ condition ] ';
```