Policy Contexts: Controlling Information Flow in Parameterised RBAC

András Belokosztolszki[*], David M. Eyers[*], Ken Moody
University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge, United Kingdom
{firstname.lastname}@cl.cam.ac.uk

Abstract

Many RBAC models have augmented the fundamental requirement of a role abstraction with features such as parameterised roles and environment-aware policy. This paper examines the potential for unintentional leakage of information during RBAC policy enforcement, either through the exchange of parameters with external services when checking environmental conditions, or through a policy design which does not appropriately separate policy subsections with different basic purposes. We propose a simple, robust mechanism for handling these problems, and illustrate our approach with a current application of our OASIS RBAC system.

1 Introduction

Role-based Access Control (RBAC) has become a widely adopted security paradigm through its intuitive match to many real-world applications. The most fundamental RBAC notion is the *role* abstraction; namely that roles are an intermediate entity sitting between the users or principals in a given security setting, and the privileges that these users must be granted in order to use the resources of the system. This abstraction mirrors the way roles usually operate within human organisations; a particular person will have certain privileges derived from their job title, and thus only indirectly associated with that individual. Figure 1 depicts this simple scenario.

As suggested by Sandhu et al. [14], the role abstraction is merely a base for many more complex features (this base model is referred to as $RBAC_0$ in their work). Their other models either incorporate notions of constraint-guarded role-to-role transitions ($RBAC_2$), the concept of role hierarchy and delegation of authority ($RBAC_1$), or a combination of the two ($RBAC_3$).

RBAC models incorporating rule-based role activation (such as $RBAC_2$) will generally provide parameterised roles to determine the specific behaviour of role instances in a particular activation environment. In contrast to this 'internal' parameter environment, another common extension to rule-based role activation is a capacity to incorporate external environmental factors through some predicate mechanism communicating with other system components. In this paper we shall refer to elements that incorporate external environmental input as *environmental predicates*. Here we imply both that these predicates may be true or false depending on dynamic conditions, and that, as in Prolog predicates, parameters may be bound as a side-effect of a successful predicate evaluation.

Whilst these mechanisms certainly increase the expressiveness of RBAC policy, they also increase its design complexity, and consequently the opportunity for mistakes that may cause information to flow to inappropriate destinations. Some of our earlier work [2] performed an initial threat analysis on the underlying distributed infrastructure of OASIS (our particular RBAC implementation). In contrast, this paper focuses on policy-level rather than system-level issues, and is thus more widely applicable to any parameterised RBAC model, including those that allow a two-way interaction with the dynamic environment.

We propose a system for classifying the constituent elements within parameterised RBAC role-activation constraints into *contexts* defined within the policy store. In this paper we say that a *context* names and identifies a particular section of an access control application. Each of the parameters, roles, privileges, predicates and rules used to specify policy in that access control application will exist within some number of contexts. Sometimes it will be appropriate to identify a *policy administration context*, and we discuss this example in section 7. Our primary goals are to control *information flow*, that is the passage of data throughout the system, and to support *distributed policy administration*. In our examples these constraints will be OASIS role-activation rules, which are described in section 2.

We can apply context classification for a numerous purposes. For a start, we can use contexts to restrict the pos-

[*]These authors contributed equally to this paper.

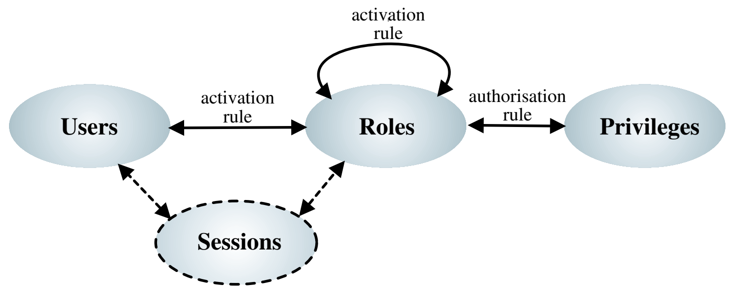

Figure 1. The basic Role-Based Access Control components.

sible specification of dependency paths between RBAC elements occurring in role-activation conditions. This style of classification is useful to protect against undesired information flow into and out of the access control environment through environmental predicates. In addition, it provides assistance in specifying the expected semantics of such predicates.

We can also use contexts to provide a basis for control over specific groups of privileges. In this case contexts help prevent undue information flow between different aspects of a single policy specification (i.e. one policy segment may define rules that relate to quite different uses of a given service than another).

Contexts can be used at a finer granularity to restrict information flow between the parameters of RBAC elements occurring in role-activation conditions, including flows to and from specific parameters of environmental predicates. The specifics of the OASIS rules and their parameterised elements are described in section 2 below.

Another use of policy contexts is to control the overall behaviour of specific groups of RBAC role-activation constraints. This allows better distributed administrative control over the deployment of policy relating to such conflicting privileges. Finally, contexts provide a basis for access control over the policy store itself.

Note that our proposed mechanism for shielding the RBAC infrastructure from the external environment is unnecessary if the environmental predicates being used are provably correct, or if they can be appropriately sandboxed. However we feel that such conditions are highly unrealistic at present. Even if other ways of ensuring information flow security exist, we believe our work is still useful in other regards, for example supporting the sub-delegation of policy administration.

This paper is organised as follows. Section 2 provides an introduction to some of the research that has been done on parameterised and context-aware RBAC. Section 3 introduces how we define policy contexts, including the specification of context hierarchies. The next three sections describe where policy context restrictions may be placed, and why each case is useful. First, Section 4 uses context specifications to limit the overall acceptable dependency properties for particular policy elements. Note that these restrictions will hold for the evolution of policy (i.e. the definition of new RBAC constraints). Restrictions on privileges are also described. Section 5 discusses applying contexts at a finer level of granularity, namely on the individual parameters of policy elements as they participate in OASIS rules. The mechanism is similar to the preceding case of applying context classification to policy elements, but designing contexts for this level of specification will focus much more on information flow properties rather than the higher level design of dependency restrictions. For example, rather than using contexts to indicate the semantics of a role for the sake of policy evolution, we might just want to “fire-wall” a particular parameter to ensure that it is only passed around within a controlled context. This is particularly relevant when the computation of RBAC constraints involves calling out into an external environment.

Following this, section 6 describes the use of context specifications to provide dependency restrictions on specific sets of rules. A striking application of all of the above styles of policy classification is to support distributed policy administration, as described in section 7. Contexts provide a useful granularity at which to define reflexive policy maintenance privileges.

Section 8 illustrates our approach via description of an application for which OASIS has currently been deployed, namely to provide a policy-based access control system within a prototype electronic health record programme for the United Kingdom National Health Service. Finally, section 9 summarises our contributions and concludes this paper.

2 Related Work

This section describes some of the research which has been done into context-aware, parameterised RBAC. As discussed in the introduction, most real-world RBAC systems now provide parameterisation; early examples include [1, 9, 10, 12]. This trend has been further reinforced by the specific parameterised models present in the NIST RBAC standards [15].

Equally, many researchers have examined how to include dynamic environmental interaction in their RBAC models [1, 5, 6, 11]. Note that some authors refer to the dynamic environment as 'context', using the word in a different sense from this paper. One common motivation for such environmental interaction support comes from a desire to enforce dynamic separation of duties (SoD) constraints within role hierarchies. For example, if a clerk role can be a prerequisite role for activating either a cheque signer role or a cheque counter-signer role, we may well want to enforce that they cannot activate both such roles for any one cheque object. Policy cannot be statically designed to uphold these dynamic requirements. The more restrictive so-called static SoD constraints can enforce such conditions, but they are overly restrictive, often crippling the expressiveness of policy. Employing static SoD in the clerk example just presented would preclude each individual clerk from being able to perform both signing and counter-signing roles, and would thus represent a significant mismatch between the expressiveness of policy and the requirements of applications. Another motivation for dynamic environmental interaction is to provide support for temporal conditions. Many models support such policy externalisation, among them TRBAC [5], OASIS [1] and Ponder [7]. Externalised policy components can clearly form a potential security problem, thus certain amount of semantic information about these components must be known to the policy engine. In the case of temporal conditions such information can be formally specified, but our approach helps to address this problem for more general cases.

An important goal we set for our context model was to address distributed policy administration. Initial work in this area can be found in [13], wherein Youman et al. introduce the ARBAC model that uses RBAC to manage access to the RBAC policy itself. Our work extends theirs as it provides additional means to group policy components together, thus adding to the expressiveness and simplifying such administrative RBAC policies.

The basic information flow control notion we employ is certainly not new in computer science. For example, a precise and formal model is presented in [8]. We use a slightly different terminology, referring to contexts instead of security classes or security labels. Also, we generally do not need most of the more complicated features of some of these models, such as dynamic security label binding or analysis of implicit information flow.

The OASIS system [1] developed at the University of Cambridge is a distributed RBAC model that incorporates parameterised roles, appointment (which generalises delegation), and rule-based role activation. There is extensive support within these rules for checking context by supplying parameters to environmental predicates.

Whilst appointment is a useful feature of OASIS, for the purposes of this paper an appointment can simply be thought of as a special type of role with a lifetime exceeding that of a session, whose only privileges are to enable the activation of standard (session-bound) roles. We shall often use the term *'rule element'* when referring to a role, appointment, privilege or environmental predicate, these being the possible preconditions for, or targets of OASIS rules. Note that special care must be taken when considering information flow relating to environmental predicates. The reason for this is that environmental predicates can set parameter values (these are the *out* parameters), and they can do so by considering the values of *in* parameters - indeed this particular type of information flow provided a strong motivation for the research presented here.

OASIS role activation rules

OASIS policy is specified via two types of parameterised rules: role activation and authorisation rules. A *role activation rule* allows a user to be assigned an active role within the context of a session, and is of the form:

$$r_1, r_2, ..., r_{n_r}, ac_1, ..., ac_{n_{ac}}, e_1, ..., e_{n_e} \vdash r$$

where r_i, ac_j and e_k represent the n_r prerequisite roles, n_{ac} appointment certificates and n_e environmental constraint predicates in this rule respectively, and r is the target role. In fact, any of these rule prerequisites and the target rule may be parameterised. An example rule might be:

$$\begin{array}{rcl} \text{doctor}(doctorID?), & & \\ \text{currentShift}(shift?), & & \\ \text{isOnDuty}(doctorID,\ shift) & \vdash & \text{doctorOnDuty}(doctorID) \end{array}$$

Here doctor is a prerequisite role, and currentShift and isOnDuty are environmental predicates. Note that the appended '?'s indicate that those parameters are bound to the values already set when the rule prerequisite elements in which they appear are presented to the rule. The alternative is parameters whose values are bound via the successful evaluation of an OASIS rule. These parameters are typed; this type information is recorded in the definitional section of the policy, and thus does not need to be repeated in the rules themselves. Often parameter modes do not need to be specified either, since they can be inferred from the specification of a particular environmental predicate; but this is not always the case, since parameters of some predicates can occur in both 'in' or 'out' mode. It is therefore good practise to define the mode when specifying the rule.

It should be clear from the above that an OASIS role activation rule can pass parameters into and out of the rule evaluation system via environmental predicates. These predicates may be as simple as type conversion functions, but they may also implement such complex operations as external database queries. In addressing information leakage, we differentiate explicitly between local functions and environmental predicates. Functions are blocks of code that run entirely within the OASIS service. They may provide facilities such as type conversion and other useful tools for evaluating expressions within rules – they are trusted pieces of code within the OASIS infrastructure that do not risk leaking information externally. On the other hand, environmental predicates extend the functionality of OASIS by incorporating an external computation. Examples might include performing database lookup, or incorporating time into OASIS rule evaluation. Potentially they can have undesirable side-effects, and the information flow between them and the OASIS infrastructure must be considered carefully.

OASIS authorisation rules

The other main type of OASIS rule specification is the *authorisation rule*. Authorisation rules assign a privilege to a role, and have the following form:

$$r, e_1, \ldots, e_{n_e} \vdash p$$

where r is the role in question, p is the privilege, and e_k represent the n_e environmental constraint predicates. As for role-activation rules, authorisation rule components may be parameterised. The following example describes an authorisation rule wherein someone active in the role of a doctor currently on duty, who is near a patient, and is assigned to be treating that patient, can gain the *read* privilege on the patient's electronic health record (EHR). We assume that the patientZone environmental predicate works using some sort of automatic location sensor in a hospital, and that the treatingDoctor environmental predicate confirms the mapping between doctors and their current patients via a database lookup.

$$\begin{array}{r}
\text{doctorOnDuty}(\mathit{doctorID}?), \\
\text{patientZone}(\mathit{patientID}?), \\
\text{treatingDoctor}(\mathit{doctorID},\ \mathit{patientID}),
\end{array} \quad \vdash \text{readEHR}(\mathit{patientID})$$

We can impose context restrictions on rule parameters and rule elements because of the comparative simplicity of rule specifications compared to arbitrary programming languages. General purpose programming languages are currently too expressive to be sand-boxed effectively. OASIS policy rules strike a balance – they are not designed to provide Turing completeness in themselves, but they can include programmable predicates. OASIS rules are declarative, and simple enough to check statically to ensure that no cyclic dependencies arise during parameter evaluation.

Before we discuss our parameter and role context model, we should mention web-services. The original OASIS prototypes were developed before the notion of web-services had been considered. However, web-services provide a very convenient mechanism through which access control systems might incorporate context. Our most recent Enterprise JavaBeans implementation of OASIS does in fact use web-service technologies (notably SOAP and WSDL) to provide environmental predicates to OASIS policy servers. Evidently such public forms, and indeed the risks involved with calling services "at arm's length", emphasise the need for mechanisms to avoid information leakage.

3 Context model

Our model for classifying policy components into particular contexts can be viewed as an extremely simple form of *meta-policy*, namely a policy on the use of policy. Our examples use the OASIS architecture, since it is the environment in which we have implemented and tested our ideas, however it is important to realise that much of what we propose is relevant to any RBAC system. All RBAC systems can use contexts to classify roles, and parameterised RBAC systems can additionally use contexts to classify role parameters. The OASIS-specific context comes when we use policy contexts within the semantics of OASIS rules. Generally our classification contexts are much simpler - indeed are really just a form of static type checking.

Note that the context extensions that we have presented in this paper are not directly integrated into our current OASIS implementation. Instead, the XML policy files understood by OASIS are used as a conduit between it and the *'Desert'* policy authoring and analysis tool, first introduced in [4].

As mentioned in the introductory section, there are a number of independent uses for the policy context framework we describe. The following four sections specifically examine some of these applications. This section, however, is focused entirely on the actual definitions of contexts. First we discuss how context definitions interleave into current OASIS policy specifications. We then extend the notion of contexts to include context hierarchies and discuss the semantics of sub-context relationships.

3.1 Context definitions for segregation

Currently OASIS policy specifications are divided up into two main areas; a dictionary that defines access control entities, and a rule repository that specifies the conditions under which role activation and privilege authorisation may take place.

We describe here how basic policy contexts are defined, and thus are augmenting the first region of OASIS policy specifications. Note that currently our OASIS implementation stores its policy in XML files contained within each OASIS role-membership certificate issuing service. The structure of these files provides a convenient framework in which to present the concepts and examples introduced in this paper. However, subject to a threat analysis, we intend to shift from a policy file methodology to one employing a more sophisticated, secure, shared repository. This is particularly relevant to the requirement that distributed access control should be applied to the access control policy itself. See [3] for further discussion.

Because the dictionary section also specifies types for the parameters, these types need not be repeated when later defining the prerequisite conditions and the targets of rules. Information flow for parameters should be considered alongside parameter types, in much the same way as when analysing the use of variables in a strongly-typed programming language. However, in our current OASIS implementation, parameter types are specified in terms of Java types, and thus do not explicitly appear in the policy definitions. An XML fragment from the example presented in section 8, is shown in Figure 2. The `definitions` node contains a set of context definitions (which for simplicity are simple, flat contexts in this example). Each of the parameters in the other elements that are defined provides, through the `contexts` attribute, a list of the contexts in which it may be used.

Some aspects of policy may not care to handle contexts explicitly, yet policy elements without explicit contexts should be able to participate alongside policy elements that define them. We therefore specify that unless explicitly overridden, all policy elements belong in the `default` context.

Note also that the context definitions presented in figure 2 do nothing beyond providing a means for the segregation of policy elements. Implicitly information flow is permitted within elements of the same context. In this case, elements which participate in multiple contexts (such as the role `doctor`) thus allow information flow within their context union. However, in this case the context definitions have not explicitly described anything about their desired information flows. We present our mechanism for allowing contexts to describe these intended information flows in the next section.

```
<?xml version="1.0"?>
<!DOCTYPE policies PUBLIC
 "//CBCL//DTD policies//EN"
 "http://www.cl.cam.ac.uk/
  Research/SRG/opera/oasisPolicy1.1.dtd">

<policies>
  <definitions>
    <context name="web"/>
    <context name="nhs"/>
    <context name="audit"/>

    <role name="doctor">
      <param name="doctorid"
             type="java.lang.Long"
             contexts="web nhs"/>
    </role>

    <role name="patient">
      <param name="patientid"
             type="java.lang.Long"
             contexts="web nhs" />
    </role>

    <environment name="current_duty"
       class="nhs.environments.IsCurrentlyOnDuty">
      <input name="id"
             type="java.lang.Long"
             context="nhs"/>
    </environment>

    <environment name="emp_db_user"
       class="nhs.environments.QueryEmployeeDB">
      <input name="gmcid"
             type="java.lang.Long"
             context="nhs"/>
      <output name="doctorid"
              type="java.lang.Long"
              context="nhs"/>
    </environment>

...
```

Figure 2. An example XML definitional policy fragment

3.2 Context definitions with information flows

The example presented in figure 2 showed contexts being used as a mechanism to segregate policy elements. However, there will often be cases where information flow is necessary between two contexts, but the policy element wishing to use this flow is not itself permitted to participate in both (e.g. in the manner of the `doctor` element in figure 2). In such cases, we need to augment our context definitions to describe the explicit information flow. An ex-

```
<context name="WAPgateway">
<source name="*" />
<target name="webForms" />
</context>

<context name="webStats">
<source name="webForms" />
</context>

<context name="webForms">
<source name="*" />
<target name="*" />
</context>

<context name="logging">
<source name="*" />
</context>
```

Figure 3. XML context definitions with explicit information flows specified

ample is shown visually in figure 4, and our current XML representation is shown in figure 3.

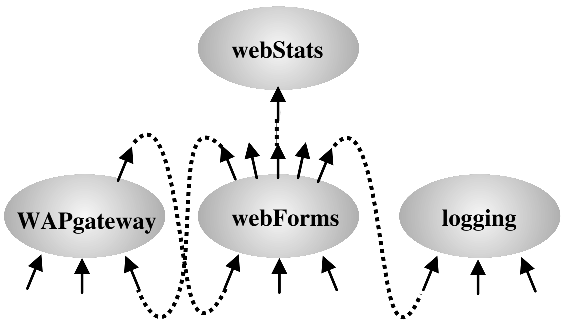

Figure 4. Example information flow among contexts.

Note that there is no default information flow unless explicitly specified. Thus the context `webStats` does not have any specific acceptable external information flows currently defined.

It may be desirable that a context define a general notion of information flow without referring to explicit sources or targets. Although we have not included complex use of this feature in our example, we suggest that the `name` attributes of the context specification actually contain XPath expressions relative to the root node of the context definitions, subject to the additive graph construction conditions below. An alternative is simply to include multiple `source` and `target` nodes. Thus context `webForms` specifies that in fact any information flow will be accepted either in or out of this context. Such a context may be used for distributed policy administration, which we discuss in section 7. The context `logging` can be considered as a "mode in" context. Unless other contexts specifically allow information flows from `logging` it will remain a context to which information only flows inwards. Of course we are assuming that the administration of the context definitions themselves does not permit lower-level distributed administration to break higher-level context information flow constraints.

Altogether, the context definitions specify a directed graph in which the context entities form the graph nodes, and the directed edges between nodes are specified within each context definition. Note that it is acceptable to have cycles in the information flow graph, since it defines all the potential, permissible flows. If an actual information flow suggested by a policy rule involves two contexts which are directly connected in a cycle, the two contexts effectively become synonymous.

At the moment this edge handling process is entirely additive; that is, we do not permit a context to explicitly reject graph node interconnections that may already have been set up by other context definitions. Our work so far has not yet uncovered sufficient need to justify this extra complexity, although we remain open to the possibility. Similarly, we shall be examining the potential utility of parameterising the contexts themselves.

3.3 Hierarchical context definitions

The previous section described a simple implementation of our context specification approach. This section extends the model to include hierarchical context definitions, and describes how to determine whether a parameter spanning two specific contexts is in fact acceptable. We feel that many applications might utilise notions of hierarchical context. For example, considering a web context, one might define a sub-context within this web context for when communications occur over a specifically secure web connection. It also enables a hierarchical structure for distributed policy administration.

The requirements for context hierarchies are similar to Java interfaces, and thus avoid the difficulties of overlapping routes to multiple inheritance tree roots. An example of a hierarchical context might be:

```
<policies>
  <definitions>
    <context name="webPublish">
      <target name="*"/>
    </context>

    <context name="secureForm"
            parentcontext="webPublish">
      <target name="secureWeb"/>
    </context> ...
```

When we organise contexts in a hierarchy, it is important to consider where the `default` context sits. Predictably, since we are trying to facilitate newly defined contexts that start with no accepted information flows, the `default` context is not the root of the context hierarchy. By omitting the `parentcontext` attribute, any specified context is assumed to have the root context as its parent context. Note that for security reasons the root context is not addressable or definable in the manner of other contexts, and may not have information flows directed to or from it. Similarly, the `default` context is not addressable in context definitions, but it can be used as a source or target of information flows. Were the `default` context to be definable in the policy file itself, it might cause existing policy to be accidentally "retro-fitted" with unintended context characteristics.

The semantics of a context hierarchy are that sub-contexts are automatically permitted to define information flows to and from their context parents. Note that these potential information flows are not defined automatically – it is not possible to generalise whether context users consider sub-contexts more specific with respect to information flows in or out, and thus we must leave this decision to the policy architects.

Context hierarchies are also highly useful for controlling sub-administration of policy. We discuss this in section 7.

Note that context hierarchies in an RBAC system are independent of any notion of role hierarchy which might exist. An RBAC system supporting role hierarchies would probably choose to have lower-roles inherit contexts from their parents, but we have not examined this question in detail – OASIS uses the more general notion of appointments to replace role hierarchies.

4 Context specifications for rule elements

In this section we discuss the application of the context concept to rule elements, which in the case of OASIS are prerequisite and target roles, appointments, environmental predicates and privileges. Our context assignments do not preclude assigning element participation to multiple contexts. We feel such a mechanism will greatly assist in managing information flows in large-scale policy specifications. In particular, it is worth noting that by using the context-based specifications of permissible information flows over rule elements, we are able to ensure that policy evolution will maintain these constraints, including if the contexts in which such elements exist change.

We provide a simple example of two roles which are both tagged as members of certain contexts.

```
...
<role name="doctor" contexts="web nhs">
  <param name="doctorid" type="java.lang.Long"/>
</role>
<role name="HIVconsultant" contexts="secureWeb nhs">
  <param name="doctorid" type="java.lang.Long"/>
</role>
...
```

Clearly the doctor role is in two contexts, `web` and `nhs`, and the HIVconsultant role is in both the `secureWeb` and `nhs` contexts.

If the relevant roles, appointment certificates and environmental predicates are assigned contexts, then the OASIS rules which represent policy constraints can be checked against the information flow defined for these contexts. To determine if information flow constraints are met at this level, we treat all prerequisite elements as being information flows into this rule, and the rule target as an information flow outwards.

In the case of role activation rules, target roles, and all prerequisite elements must satisfy their context information flow constraints. We take the union of all the prerequisite elements' contexts. The activation rule will not be accepted unless there is a mapping through defined context information flows from this combined source context into the context of the target role.

In a similar manner technically, although the application semantics are different, authorisation rules require that the union of the prerequisite elements' contexts can be mapped through defined information flows to the context of the privilege targeted by this rule.

For an example, we take the roles we presented above and assume that information flow from the `secureWeb` context to the `web` context is not permitted. A rule such as:

$$\text{doctor}(x) \vdash \text{HIVconsultant}(x)$$

would be disallowed in our policy store since the left hand side (prerequisites) belong to a context which cannot have information flow into the context of the right hand side (target role).

Using this context mechanism, we can express and ensure conformance to static separation-of-duty constraints. To do so, we define two target contexts to contain the mutually exclusive privileges or roles, and ensure that there are no specified information flows between them. Hierarchical context definitions (see section 3.3) will make it easier to manage this when policy administration is distributed.

This simple mechanism based on context classification enables the information flow permitted between different rule elements within a policy store to be specified, and these constraints are enforced statically. The restrictions on information flow will be maintained automatically as policy evolves.

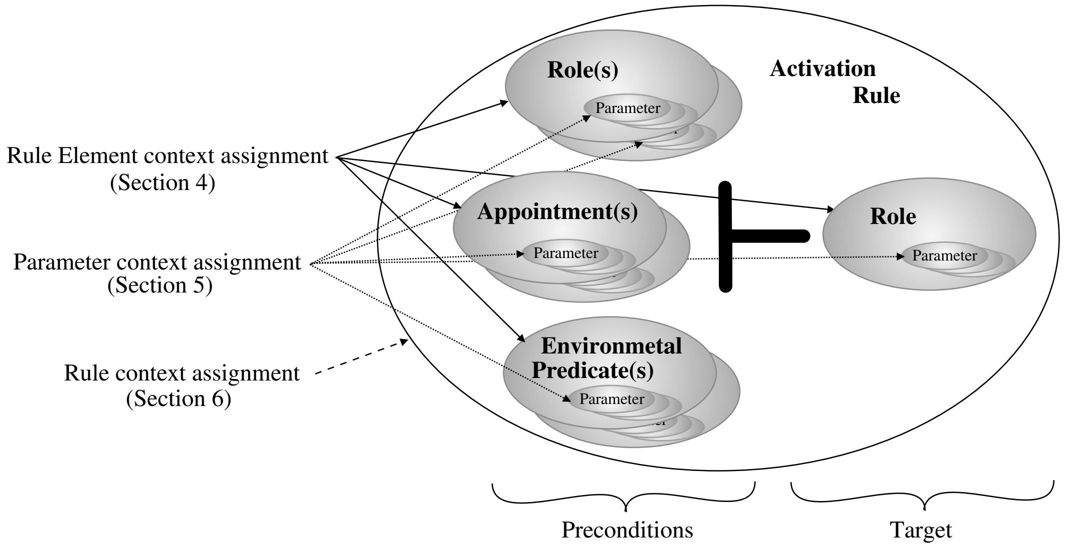
Role(s)
Parameter
Activation Rule
Rule Element context assignment (Section 4)
Appointment(s)
Parameter
Role
Parameter
Parameter context assignment (Section 5)
Environmetal Predicate(s)
Parameter
Rule context assignment (Section 6)
Preconditions
Target

Figure 5. Role activation rule components that can be assigned a context.

5 Context specifications for parameters

This section examines another application of the context specifications presented in section 3. As described in section 2, OASIS roles, appointments and environmental predicates can, and for any non-trivial deployment usually do, have parameters. So, in addition to specifying contexts for the rule elements themselves, one can assign contexts to the parameters, thus increasing the control over the information flow that takes place during the enforcement of OASIS rules. This is particularly pertinent when considering that environmental predicates evaluated entirely within the left hand side of an OASIS rule may carry information, via their parameters, into external contexts. This notion of fine-grained information flow is not captured in our analysis of policy elements in rules (presented in section 4). Also, rule-elements are specific to a particular RBAC architecture; the context specifications for parameters may be applied to other non-OASIS parameterised RBAC architectures.

The mechanism through which we specify the contexts of parameters is very similar to that adopted for the policy elements themselves. From an XML viewpoint, we can simply add a `context` attribute into the `param` nodes in the role definition section of the policy document. For example:

```
<role name="doctor" contexts="nhs">
  <param name="doctorid"
         type="java.lang.Long"
         context="secureWeb" />
</role>
```

We discuss the mechanism by which parameter contexts are checked through a worked example in section 5.2. Note also that the definition of a rule may enforce context restrictions on the parameters used within it – see section 6.

The checking of parameter context conditions is orthogonal to checking the policy element context conditions in a rule. Whilst policy architects are encouraged to use all possible context constraint approaches, they are free to choose any combination of the styles of context specification we have presented. The various context specifications only increase the restrictiveness of policy checks – there are no unexpected side-effects of using them together.

5.1 Taxonomy of environmental predicates

As environmental predicates can have side effects in addition to being able both to read and to set parameter values, it is prudent to provide a means to specify their information flow characteristics. For example, take an environmental predicate $\mathrm{eq}(a,b?)$ that sets the value of b to be equal to the value of a. There are two context perspectives from which to view this. From the perspective of an OASIS rule that uses this predicate, we do not consider the mechanics of the predicate itself; it is simply a contributer to the union of contexts on the left hand side of the rule, which must be mapped through defined information flows to the rule target.

From the perspective of parameter context checking, though, the default case for this predicate will be to assume that any parameter bound by the evaluation of this predicate will carry information from all the source contexts of its input parameters. To show why this is sensible, note that

```
<activation role="doctor" context="secure">
   <condition name="qualified">
      <match  name="gmcid"    value="\$x" />
   </condition>

   <condition name="emp_db_user">
      <with   name="gmcid"    value="\$x" />
      <match  name="doctorid" value="\$doctorid" />
   </condition>

   <condition name="is_not_struck_off">
      <with   name="gmcid"    value="\$x" />
   </condition>
 </activation>
```

Figure 6. An OASIS role-activation rule (in XML form).

a buggy, hacked or poorly-written environmental predicate could indeed copy data seen in any of its inputs to any of its outputs. In effect, we are saying that all out parameters of an environmental predicate are "tainted" or unsafe, since its code is potentially untrusted, and that in addition there is an implicit information flow from each input parameter context to each output parameter context.

However, an environmental predicate is just a type of policy element from the perspective of policy definition. Either through trust, or hopefully, through some concrete heuristic derived from the computational semantics of the predicate, the actual context information flows from input to output can be determined, and thus explicitly specified in the parameter contexts given for this environmental predicate's definition.

5.2 Rule evaluation

The semantics of parameter context checking in our rule evaluations should be intuitive, but we include an example for completeness. Note that in the current implementation of OASIS, the local parameter names (whose scope is just the rule itself) are prefixed with a '$' symbol.

Standard type checking merely involves binding each local parameter's type when their value is first set, and ensuring that all subsequent occurrences of this parameter have a compatible type. In current implementations this is trivial - the types must match directly.

In parallel to this type checking, we augment the evaluation process by also binding each local parameter's permitted contexts when their value is first set (provided this, in itself, is consistent with the mode of that parameter's context). Each subsequent appearance of that parameter is checked to ensure that the hierarchical context is compatible, as is the context mode. The only notable difference from parameter type checking is that each parameter has only one type, but may participate in numerous contexts.

For evaluation of the rule shown in Figure 6, parameter context checking will guarantee that information flow into the `doctorid` parameter will be a valid information flow into the `secureWeb` context (as in the definition of the doctor role above). Without any specific parameter context definition of the `emp_db_user` predicate, the `gmcid` parameter will also need to be in the `secureWeb` context, with consequent static restrictions over the `qualified` appointment, and the `is_not_struck_off` environmental predicate. As mentioned above, explicitly specifying the information flow in the definition of `emp_db_user` could lessen this restriction, with a subsequent increase in the dependence on the actual predicate implementation!

6 Context specifications for rules

This section examines assigning rules themselves to particular contexts, rather than relying on the context classification of their constituent policy elements and parameters. We described how the classification of policy elements is viewed as in and out information flows on the left and right hand sides of rules in section 4. Applying context classification to rules provides further overarching restrictions on what may be included within them.

For example, the rule in figure 6 has been placed in the `secure` context. This then affects the permissible contexts of this rule's policy elements and their parameters, including possible policy evolution resulting from sub-administration. Note that as for parameter contexts, the default case of not having an explicit context specification merely means that no extra context checks are performed.

Since rules are essentially containers for other policy elements, we employ two parallel context checking semantics. Each policy element and parameter within the rule must pass a test based on either:

Context hierarchy. If this element or parameter exists in a sub-context of the rule's context, it passes our static context check. Note this cannot ever be true for policy elements or parameters which do not specify a context.

Explicit information flow specifications. Alternatively we require that there is an explicit information flow between the context of this policy element or parameter and that of the rule. For elements and parameters on the left hand side of a rule, the flow must be toward the rule's context. For right-hand side rule constituents the flow must be from the rule's context.

Note that the rule checks presented here augment the information flow checks presented in section 4.

7 Context specification for distributed administration

The final application we present for the use of context tagging within policy specifications is to support distributed policy administration. The OASIS project aims to support policy-based access control for systems of a sufficient scale to require hierarchical, distributed policy administration.

Contexts used within our policy store provide a convenient handle through which we can grant sub-administrators privileges to maintain sub-sections of policy independently from one another. Since the administration contexts themselves may be hierarchical, this makes it possible for a higher-level administrator to ensure that all sub-administrators receive privileges appropriate to their needs, and to their levels of trustworthiness and competence.

We can thus permit sub-administrators to create, view and modify their own policy rules, see [3]. In terms of information flow, the local administrators themselves are indeed an element within a particular policy context.

Consistent with the hierarchical context notions presented in section 3.3, any given local administrator will be able to define new information flows, but only within contexts that they control, and in any sub-contexts. Clearly they would be in a position to modify and augment existing rules within their context, but they would also be able to create their own definitions for rule elements, including the incorporation of particular environmental predicates. All of these privileges would be subject to the context restrictions already mentioned. Creating information flows that go beyond their internal environment would require the modification or creation of definitions under the control of higher-level administrators.

Of course during policy evolution new contexts themselves may be added to the system, or existing contexts may have their information flow rules changed. This type of policy evolution is more expensive than the addition or modification of policy rules or rule elements, since all affected policy must be re-validated in light of this new context information. We thus expect that administrators would try to keep such modifications to a minimum. Various caching techniques could drastically reduce the cost of this re-validation, but we have not yet implemented such an approach.

We believe that by aggregating the inter-context flow specifications into hierarchies, we can decrease the risks of policy flaws that arise because higher-level administrators have to maintain detailed, ad-hoc connections between policy sub-sections and the privileges of those allowed to maintain them.

8 An example application from our health service deployment

This section describes our research performed in collaboration with Clinical and Biomedical Computing Limited. We are deploying OASIS to provide access control within a prototype widely-distributed Electronic Health Record infrastructure for the United Kingdom National Health Service.

The goal of the CBCL OASIS project is to provide a web-based interface to distributed health record fragments. The nodes on this NHS network are highly autonomous; indeed this fact motivated much of the research behind this paper. The main three types of elements within the system are:

The NHS Portal. From the user perspective, the NHS Portal is essentially a web server through which patients and medical staff may gain access to records they are authorised to examine. However, the NHS Portal also has an OASIS-internal side. The internal OASIS network links the NHS Portal with the other two pieces of network infrastructure described below, and securely transports fragments of electronic health record data.

The Index Server. A crucial feature of our Electronic Health Record infrastructure is that there is little reliance on standard, centrally-defined data-structures; the health record fragments may be heterogeneous in semantics (although probably homogeneous in some sort of web-oriented syntax, cf. HTML), provided that they can be keyed and located appropriately. We feel that it is highly unlikely a single record standard will be adopted across all nodes linked in a wide-area EHR network, thus the Index Server maintains information about the whereabouts of EHR fragments, without any knowledge about the medical significance of these fragments. Our current prototype uses NHSID as part of the key for record fragments. The contents of these fragments are not stored in the Index Server.

Clearly performance may be significantly boosted by some form of intelligent cache which does understand the medical significance of different EHR fragments, but this also induces significant security design issues. We have focused on core functionality.

HCO Servers. The Health Care Organisation (HCO) servers actually contain electronic health record data. These systems may be individual GP practices' electronic records systems, large-scale hospital administration systems or other lab or medical organisations' on-line databases. We require that the records of any such system can be keyed by our Index Server in terms of an approximate local time-stamp, and that the local system supports a web-style interface to which we can control access via OASIS privileges.

Note that there is not a free flow of patient record data throughout this EHR infrastructure. Each of the three types of nodes presented above has its own local OASIS policy store. Thus the policy of a particular HCO site, when presented with a request for a fragment, may be to first request further credentials from the Index Server before permitting the Index Server to acquire the privilege to read this fragment on behalf of the NHS Portal. Whilst there is some extra network traffic caused by such a design, there are the significant advantages of identity hiding and non-disclosure.

There are numerous ways in which this infrastructure would benefit from the specification of information flow contexts:

- Firstly, we can examine the problem from a network traffic perspective. From the point of view of the NHS Portal, it has two very different types of network connection. On the user side, we are dealing with mainly HTML documents travelling using the HTTPS protocol through the Internet (or an NHS subset of it). On the Index Server side, our traffic consists of SOAP messages travelling using HTTPS over an internal OASIS network. Different high-level security and confidentiality conditions would exist on these two networks.

- From the policy perspective, we ideally want to segregate information flow *within* the NHS Portal too. For example, it is desirable that there be only carefully controlled information flows between staff and patients. Indeed in our current design, there is no need to provide any such information flow directly between these two groups. Our research has already been faced with the problems caused by the fact that doctors are usually also patients within any given overall healthcare infrastructure. Contexts can assist enforcing in information flow restrictions on a user who can be seen either as a patient or a doctor.

- Finally, contexts provide a significant administrative handle for controlling policy evolution in such an NHS infrastructure. Not only are there a large number of distributed policy stores, but it is likely that the policy store within a hospital EHR system will require sub-administration. Section 7 discussed how delegation of administration can be safely controlled using contexts.

As our NHS EHR prototype moves into later stages, we hope to further incorporate context specifications into our policy management tools and privilege sets (for reflexive policy) currently under development.

9 Conclusion

In this paper we have introduced and discussed the potential for undesirable information flows in parameterised RBAC systems, particularly those allowing a two-way interaction between the internal access control system and its dynamic external environment. We have presented a simple, robust mechanism for increasing the semantic structure of policy definitions, by means of the explicit specification of permissible information flow. This research was motivated by our deployment of OASIS in a prototype electronic health record architecture, examples from which are used through the paper. Beyond information flow protection, we discuss application of our context technique to facilitate the distributed administration of policy.

Research into dynamic policy context checks is ongoing. Our future plans also include examining possible parametrisation of contexts and extending information flow specification with negative paths, i.e. explicitly forbidding information flow from one context to an other.

10 Acknowledgements

This research is supported by the Cambridge Australia Trust, King's College Cambridge, the John Stanley Graduate Fund, and the United Kingdom Overseas Research Students Awards Scheme. Thanks to the staff at Clinical and Biomedical Computing Limited, particularly Tim Mills and Ian Boston for design of the software and also to John Norman and Jem Rashbass for their continued support of our research collaboration, and also to the other members of the Cambridge Opera Research Group (`http://www.cl.cam.ac.uk/Research/SRG/opera/`), for their work and feedback.

References

[1] Jean Bacon, Ken Moody, and Walt Yao. A model of OASIS role-based access control and its support for active security. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):492–540, November 2002.

[2] András Belokosztolszki and David Eyers. Shielding the OASIS RBAC infrastructure from cyber-terrorism. In *Proceedings of the IFIP WG 11.3 Conference*, 2002.

[3] András Belokosztolszki and Ken Moody. Support for self-administration of OASIS RBAC policies. University of Cambridge, Computer Laboratory.

[4] András Belokosztolszki and Ken Moody. Meta-policies for distributed role-based access control systems. In *Policy 2002: IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 106–115, 2002.

[5] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233, August 2001.

[6] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20, 2001.

[7] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks,International Workshop, POLICY 2001, Bristol, UK*, pages 18–38, 2001.

[8] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[9] Luigi Giuri. Role-based access control: a natural approach. In *Proceedings of the first ACM workshop on Role-based access control*, pages II–33–37, 1995.

[10] Cheh Goh and Adrian Baldwin. Towards a more complete model of role. In *Proceedings of the third ACM workshop on Role-based access control*, pages 55–62, 1998.

[11] Cheng Hian Goh, Stuart E. Madnick, and Michael D. Siegel. Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic environment. In *Proceedings of the third international conference on Information and knowledge management*, pages 337–346. ACM Press, 1994.

[12] Emil Lupu and Morris Sloman. Reconciling role based management and role based access control. In *Proceedings of the second ACM workshop on Role-based access control*, pages 135–141, 1997.

[13] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: preliminary description and outline. In *Proceedings of the second ACM workshop on Role-based access control*, pages 41–50, 1997.

[14] Ravi Sandhu, Edward Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[15] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, 2000.