XacT: A Bridge between Resource Management and Access Control in Multi-layered Applications

Maarten Rits, Benjamin De Boe and Andreas Schaad
SAP Research
805, Avenue du Docteur Maurice Donat,
Font de l'Orme, 06250 Mougins
+33 (0)4 92 28 62 00
{maarten.rits,benjamin.deboe,andreas.schaad}@sap.com

ABSTRACT
In this paper we describe the eXtreme access control Tool (XacT) which provides an automated way to obtain access control information out of multi-layered applications. We believe that based on this information consistent access control policies can be specified to prevent over-privileged accounts. The main difficulty, that leads to these over-privileged accounts, comes from the distinction that must be made between identifying which users should perform a workflow task (resource management) and which users are allowed to perform a task (access control), as well as the fact that access control enforcement is typically spread over different layers in applications (e.g. database layer, operating system layer, workflow layer). In this paper, we present an automated way to obtain access control information out of multi-layered applications. We base our observations on recent insights into workflow controlled judicial information systems.

Categories and Subject Descriptors

General Terms
Management, Security, Languages.

Keywords
Workflow security, Role-based access control (RBAC), Aspect oriented programming.

1. INTRODUCTION
In general, most applications are implemented as services on top of one or more databases. Access control enforcement is both situated at the application (method-) level and at the database level. In case access control policies of a service are defined independently with regard to the database policies, authorization mismatches are likely to occur in practice. To avoid administrative overhead connections to databases are often established under over-privileged accounts, which shifts most of the access control responsibility to the higher service layers. Workflow systems even introduce a ‘virtual’ third layer (the workflow application) which is in fact an application running on top of other applications. Access control in this layer is specified and enforced on workflow activities. To avoid administrative overhead in workflow applications, the applied access control policies in the underlying layers are in practice also kept very permissive. The reason for this is that if a fine-grained access control policy would be applied in the underlying application- and data-layers, authorization exceptions would be very likely to occur on the higher layers, because they are composed of the underlying services. We believe that at the moment access control policies in different layers are defined independently of each other. This is the main problem we address in this context. Unfortunately, the authorization exceptions are in practice often solved by just extending the rights of the organizational role that has been assigned to execute the workflow activity, without further in-depth analysis of why the authorization was denied. Performing such an analysis for every case is in practice not feasible. These uncontrolled right assignments lead in the end to over-privileged systems.

In this paper we propose a methodology that allows to specify consistent access control policies. We use an automated tool called eXtreme access control Tool (XacT) based on aspect-oriented programming techniques [5]. This approach enables us to enforce fine-grained context-based access control on each layer, while preventing authorization exceptions on higher layers, because we can now make the access control policies dependent on each other. We base our observations on our recent insights into judicial information systems dealing with very sensitive information, subject to strict access control policies. The judicial processes supported by these systems need to be controlled through workflow systems.

The paper is organized as follows. In section 2 we discuss related work. Section 3 presents an overview of the Aspect Oriented Programming (AOP) language AspectJ, which is used in our tool. In section 4 we discuss the general methodology and architecture.
of our approach. Section 5 presents a practical case-study in which we used our tool. Section 6 provides a summary and discussion of further work.

2. RELATED WORK

In [19] two ways are discussed for reliably defining service policies to prevent authorization mismatches. One approach is to extract and adapt database policies. An alternative is to generate the service policies based on the predefined service functionality. Adapting database policies is practical for administering the access control of already implemented services. In this case the service functionality as well as the database interaction are already given. Access control rules that apply to specific user accounts employed to establish a connection to a database, are extracted. These constitute the maximum set of privileges that can be granted to the service users. The privileges are stored in so-called permission policy sets. The service developer is able to select the permission policies that are applicable to the service operation. The privileges can be further adapted and restricted by additional conditions. Finally, these possibly modified privileges are assigned to user profiles for the service domain. The concepts presented in this paper most closely resemble our work, however the authors did not provide in-depth technical details.

Authorization mismatches are also mentioned in [17]. In object-oriented systems, there is a tendency to consider a method as the unit of privilege. J2EE [1] and .NET [7] support role-based access control and permissions assigned to these roles are based on method invocations. However it is argued that in the majority of the applications in Medical Information Systems, privileges are more naturally expressed in terms of access to and modification of data. If the application deployer thus wants, for example, to enforce a data modification access rule, he needs to find out which methods may possibly modify the data. It is argued that this is hard to realize, since it can not always be determined beforehand whether or not a method will access and/or modify a given data-unit. Our solution provides a way to enforce fine-grained access control on the data-layer and also on the application layer. It is possible to make the different policies consistent with each other. For example, we are able to check which methods and data-units are accessed by a given service.

The use of process models for analyzing security requirements is discussed in [10]. During the analysis of business processes the following information is recorded: the place where data are kept and stored; where they are sent; who processes them; and who accesses them. This makes the flow of the data and who is able to access or change them more transparent. Therefore, this information can be used to determine which safeguards must be taken where and by whom to provide an appropriate level of security. Different steps are described to check the security of a business process: For example checking whether the person’s clearance levels are high enough to carry out a specific activity; checking whether a certain data structure might be handled during a certain activity; or checking whether the person to handle a certain data structure has high enough clearance levels. In case an inconsistency is found, the model has to be reviewed. Generally this can be resolved in two ways: i) reassignment of a clearance level ii) change of the process and assignment of an activity to a different actor; or splitting an activity into separate tasks. The discussed ideas are interesting, but the proposed methodology lacks a more formal description that would allow to automate these steps.

To summarize, the discussed solutions all try to set up some kind of interaction between the specification of access control policies on the application and data-layers. Their general shortcoming is, however, that still a lot of effort is required from an administration perspective, leading to an overhead that we believe makes their application very unrealistic in large scale systems.

We present an automated way to make access control policies between different layers consistent with each other. The tool is very useful for resource management and access control in workflow systems, because it can provide the administrator with a list of organizational roles that have all the required permissions to execute the methods and access the data-units that are used in a certain activity.

3. ASPECTJ

For many problems the modularity as offered by object-oriented programming is not powerful enough to clearly capture some important (non-functional) design concerns, which results in code that is scattered throughout the program. Aspect Oriented Programming (AOP) [5] promises to tackle this problem by offering several abstractions that help to reason about and specify such concerns one at a time.

AspectJ [4] is an extension to the Java programming language, and is currently considered as the most advanced implementation of the AOP methodology. In AspectJ, pointcuts define specific points in the dynamic execution of a Java program. Pointcut definitions are specified using primitive pointcuts such as the reception of a message call, the creation of a specific type of object etc. Primitive pointcuts can be combined using logical operators. On pointcuts, advice can then be defined in order to execute certain code. AspectJ supports “before” and “after” advice, depending on the time the code is executed. In addition, “around” advice enables the replacement of the original code, however, constructs are offered to continue parts of this original code. The definition of pointcuts and the specification of advice on these pointcuts together form an aspect definition. Besides these special constructs, an aspect is similar to a class and can as such contain data members, methods, etc. To deploy the aspects in a concrete application, AspectJ provides a special compiler that parses all application and aspect code and weaves them into normal Java code, which is then compiled using a standard Java compiler. The use of the pointcut and advice constructs will become clearer when we discuss XacT in more detail throughout the following section.

Several experiments have been undertaken with AspectJ for the enforcement of access control in applications [3]. In our approach we use AspectJ for the improvement of access control policy specification, but not for the enforcement of access control itself. We do this by obtaining fine-grained information about the access control protection out of an existing application. The “concern” we address with AspectJ could therefore be classified as the “monitoring” of an application.
4. ARCHITECTURE

In this section we present the architecture of our approach and some details about the concrete implementation of the tool. Currently, only a J2EE version of the tool is implemented, but the general architecture should be valid for other environments like Microsoft’s .NET [7] also.

4.1 Implementation Details

Figure 1 shows pseudo-code for the aspect MonitoringAspect that is used. The pointcut allMethods selects all method calls and all constructor calls within the ejustice package, but not within the test package. The ejustice package contains the business logic of an initial prototype implementation for a workflow system based on the electronic legislation process scenario described in [12].

In addition to plain Java code component technologies like Enterprise JavaBeans [15], and Web services [18] are also used. The test package contains classes with configuration parameters for our tool and therefore it is excluded by the pointcut. The second pointcut mainMethod selects the main methods of the application clients in the prototype workflow system. These clients are used to invoke specific services like for example an EJB method or a web service. They are separated from the ejustice package for reusability issues. The before advice on the allMethods pointcut gets the stack trace of the currently invoked method selected by the pointcut and compares this stack trace to the elements in the container provided by the getTargets method, which contains the signatures of the methods that are access control protected in the system. These can be obtained by consulting the designated access control policies. If a match is found, a PolicyMonitor instance is created based on the type of access control policy that is used for the selected method; this could be JAAS [16], EJB deployment descriptor [1], or XACML [8]. The PolicyMonitor instance will then inspect the policy and returns a set of roles that are allowed to execute the method. Both the stack trace and the set of roles are maintained in a report. To be able to inspect the policies, the PolicyMonitor instance needs read access to the access control policies in the system. This is not a problem because our tool will typically be invoked by system administrators and they should at least have read access to this information.

There are also two after advices on the mainMethod pointcut; they will generate the final report on completion of the application monitoring process. The first advice is for normal completion of the process, the second one when the monitoring process ends with an exception being thrown, in this case the error message will be printed but the intermediate report will also be generated. The code has been left out because of the minor importance.

4.2 Methodology

Now that we explained the working of the aspect, we will continue by looking at the architecture from a higher perspective. It should be clear, however, that although we used AspectJ and other Java technologies, this methodology is generally applicable and would give similar implementations when other technologies like for example .NET [13] would be used.

For XacT to work properly, we do not only have to weave the aspect into the original application code, we also have to execute the woven application. The advantage of dynamically inspecting or “monitoring” our application is that run-time context information relevant to the access control policies can be taken into account. This is not possible by statically examining the code. For example when a data-access method is invoked, it is important to know what the path is that is used in the method-call, because the access control policy will have different constraints on different data-paths. In most cases, according to the principles of modular and reusable code-design, this kind of information will not be hard-coded into the application, it will be read from configuration files. So these values will only be known at run-time.

Applications typically consist out of conditional branches. Based on certain conditions, mostly dependent on input values, different paths in the application code will be executed. In order to consistently calculate the subset of roles that have all the permissions to execute the methods that are called from a certain activity, we need to take every possible path into account that can be traversed in the application. We solved this in our prototype by using White Box test data from test cases as input for our application monitoring process. If this data is not available, other means should be used to force the execution of every application branch. However, because code testing and corresponding test cases are an integral part of almost all commercial available applications, we are quite confident that this approach could be

```java
Public aspect MonitoringAspect {
    Private pointcut allMethods() : (call(* *(..)) || call(*.new(..))) &.&
        within(ejustice..*) && ! within(test.*);
    Private pointcut mainMethod() : execution(static void main(String[])) &.&
        within(ejustice..*)

    before() : allMethods() {
        StackTraceElement[] trace = (new Throwable().getStackTrace);
        Downcall dc = new Downcall(thisJoinPoint, trace);
        for (int i = 0; i < getTargets().size(); i++) {
            if (getTargets().get(i).getTarget().matchesDowncall(dc)) {
                System.out.println("-XacT- match found!");
                PolicyMonitor pm = PolicyMonitorFactory.getPolicyMonitor(getTargets(),
                    get(i).getAccessControlType);
                Policy pol = pm.inspectPolicy(dc);
                getReportGenerator().addDowncall(dc, pol);
                calculateAllowedRoles(pol.getAllowedRoles());
            }
        }
    }
    catch (Exception e) {
        StackTraceElement[] st = e.getStackTrace();
        System.out.println(toString() + e.getMessage());
        for (int j = 0; j < st.length; j++)
            System.out.println(st[j]);
    }
    after() : mainMethod() {
        if (!finished)
            getReportGenerator().finishReport(allowedRoles);
            System.exit(0);
    }
    after() throwing (Exception e): mainMethod() {
        //
    }
}
```
generally applied. We do, however, admit that we did not use our tool for experimentation with large applications. In this case, it could take some time to examine every possible path in the code. Yet, the fact that launching the woven application with different test input data could take some considerable amount of time is not really such a disadvantage, taking into consideration that XacT provides in the first place only a recommendation and only has to be invoked one time on the ‘whole’ application in order to optimize the access control policies. After this is done, the tool only has to be used when changes are made to access control policies that affect the higher level policies, and then it will only be invoked on the specific parts of the application that are involved. Changes that affect higher level policies are:

(i) a permission is retracted from a role to execute a certain method and this method can be invoked by a higher-level service where the specific role is assigned as the responsible for execution from a “resource management” perspective. In this case, the tool has to check whether there is another role that has the permissions to execute all underlying methods, so that it can be assigned as the resource for the higher-level service. Only the part of the application that is invoked by the higher-level service has to be examined.

(ii) a permission is added to a role to execute a specific method that caused to split up a higher-level service into different services because no role was found in the organization to perform the service. This is one of the benefits of our tool; separating workflow activities into several steps, because of the lack of organizational roles that are allowed to execute all underlying methods. Based on this information we can correctly enforce separation-of-duties requirements. It could be argued that in this case there is not really a need for reusing the tool; we could leave the activities as they are. A more in-depth analysis on performance and system changes will be subject of future work.

5. JUDICIAL WORKFLOW

In this section we discuss an application scenario that formed the basis for a research implementation on which we used XacT. It concerns a scenario from the BKA in the context of the Austrian Legal Information System RIS. The study has been performed in the context of the EU FP6 project eJustice.

5.1 Background

Governments are now aware that offering their services online will help them to reduce costs [9]. One example of this evolution is the Austrian eRecht system, which aims at replacing all paper-printed laws in Austria by 2004. This system supports all legislative stages from drafting a document, its review and debate in the appropriate chambers, application of digital signatures, to final publication in an internet-based database. It allows each Austrian citizen to access the laws, regulations and other legal supporting information of their country. Here we present a case study of one process that uses this system: The process of changing a federal law concerning the Austrian Highway Code.

Since such processes and their results (the laws of a country), have an enormous impact on society, they need to be secured against external and internal alteration, be it inadvertent or malicious. This is even more important in the electronic world. In fact, strong evidence [6] suggests that internal alteration is far more critical than often perceived in the current age of Internet-connected systems.

The eLaw system of the Austrian Federal Chancellery is an electronic legal records processing system which certain ministries make shared use of. This system can, for example, be used to facilitate and manage changes to existing laws. As such, eLaw may be classified as a records management system for public administration. The workflows implemented in eLaw are enforced through Fabasoft’s eGov Suite 5.0.

Legislative information (e.g. gazettes, instruction edicts or tribunals) and the law that has been agreed upon by the involved political parties, is published in the Austrian Legal Information System RIS. The aim of this system is to replace printed law texts with digitally signed electronic documents, which are legally binding. The eLaw system is one of more than 30 public administration systems that feed data for publication into the RIS. The RIS currently provides services to more than 17,000 public administration officers over a nation-wide Intranet dedicated to the task. In addition, the general public may access the electronically published law via the Internet. RIS users access more then 6.5 million documents each month.

5.2 Process description: Updating a law

A typical scenario detailing the use of eLaw and its interaction with the RIS is that of a change to an existing law, e.g. a change to the law concerning the Austrian Highway Code. This process is illustrated by figure 2, and in the rest of this subsection each paragraph describes a step of the process (denoted by a rectangle in the diagram). Note that each step is based on a specific law which prescribes the exact legislative procedure, but such detailed consideration is not possible in this context.

A law clerk working for the ministry of transport prepares a draft of the proposed change (step 1 in figure 2). This draft is initiated within the eLaw system (i.e. it is given an initial classification and some MS Word Documents are created). It is then decided whether the draft should be reviewed by external stakeholders (step 2). In our case of a change to the Highway Code, the draft would normally be sent to the Austrian Automobile Club for comments. The stakeholders are identified and invited either electronically (via email) or by post to review and comment on the draft bill (step 3). The draft is made available to them by physically sending them a copy of the draft or through the RIS (but not using an electronically signed format at this stage).

The stakeholders then review the draft (step 4), and send back their review either electronically or by hardcopy. A deadline may have been set for the review. Once the deadline for the stakeholders’ review and comment has passed, the draft is prepared for further discussion (step 5) by a department responsible for coordinating the meetings of ministers, and the draft is eventually put on the agenda of the weekly meeting of the Austrian federal ministers.
If at the discussion (step 6) the draft is rejected, it has to be proposed again by the initiating ministry. In this case the draft is revised by a law clerk (step 7) and it is decided whether further review of the revised text is required. If the federal ministers agree to accept the draft without change, it becomes a government bill and is put into the RIS. At this stage there is no binding digital signature. The bill is then transferred to the systems of the parliament.

The government bill is now discussed by the national council and then by the federal council (step 8). Both chambers may either agree or disagree, and possibly change the government bill. If at the discussion (step 6) the draft is rejected, it has to be proposed again by the initiating ministry. However, the draft must not be revised by the same law clerk who initiated the draft.

Prior to final publication, the president must sign and approve that the change to the law has been performed according to the constitution (step 9). The president’s approval must also be countersigned by the chancellor (step 10). These two steps currently require a paper-based signature.

A final check of the new or changed law is performed by the constitutional service (which is a department of the federal chancellery), who also give the document the appropriate label with respect to the federal law gazette (step 11).

Finally, the changed law is published in the RIS after it has been digitally signed to provide authenticity (step 12).

5.3 XacT Experiment

After having described the legislative process, we now list some of its specific requirements regarding access control. It should be noticed, however, that even in this relatively small scenario already a lot of access control rules have to be taken into account, and we do not claim this list of requirements to be exhaustive.

Requirement 1: One of the most often ignored requirements, despite being one of the most obvious, is that a legal clerk should not work in two incompatible offices. For example, a clerk working in the ministry of transport should not have access to information in the department responsible for releasing public tenders.

Requirement 2: The clerk initiating a legal draft should not be the principal who decides whether reviews by stakeholders are required.

Requirement 3: A clerk should not be allowed to modify a document and upload it onto the RIS at the same time.

Requirement 4: A clerk should only be allowed to remove a document, already agreed upon, from the RIS if he has not been involved in its prior drafting.

Requirement 5: In case of a rejection (step 8) the draft has to be proposed again by the initiating ministry. However, the draft must not be revised by the same law clerk who initiated the draft.

Requirement 6: A clerk should not perform all workflow steps involving a legal bill.

We implemented a demonstration prototype that simulates the eLaw and RIS systems, and their interaction. Along the previous specified requirements we tried to take other general access control rules into account as well for experimentation. It became clear that a lot of these requirements are based on context information. For example, for requirement 4 we need to somehow know if this clerk has been involved in the prior drafting of the document. Not only role membership is needed as an input for the access decision rules but also some specific context information.

Figure 3 illustrates a selected part of our prototype architecture. We implemented the context-based access control using JAAS [16], EJB deployment descriptors [1], XACML [8], and stored procedures [14]. Only XACML offers a fully ‘declarative’ approach. Considering the other access control types, a part of the decision rule had to be hard-coded in the application. Figure 4 shows an example of the condition function in an XACML policy rule. For this specific access rule, the time has to be between 9AM and 5PM. An AttributeSelector element can be used to define where the values for a particular attribute can be found. This makes it possible to include any kind of context information.

Thus, through this condition function and the AttributeSelector element, XACML allows to fully declaratively specify context-based access control. The reason why we make this remark is because it is possible that a certain organizational role has all the required permissions to execute all of the underlying methods, but the ‘context’ parts of the access decision rules could be mutually exclusive. For example, when time of day is part of the decision rule, and for the execution of a specific method day-time is required in addition to some role-membership, while for another method night-time is required with the same role-membership. If these methods are located in the same conditional branch in an
activity, then nobody with that organizational role will be able to execute this part of the activity.

We would additionally like to capture these possibilities with our tool because they can lead to authorization exceptions and consequently to over-privileged accounts as explained previously. The more declarative the used access control enforcement, the more context information can be taken into account, so we strongly recommend to use policy specification and enforcement mechanisms like for example XACML [8] and Ponder [2].

Figure 5 shows an example of a report generated by our tool where we only take role-membership into account. It provides the administrator of the workflow a 'recommendation' for assigning an organizational role to an activity in the workflow. For this scenario the minister and admin roles have all the required permissions to execute the activity. The activity corresponds in this case to the MainClient module in figure 3. Only output for the EJB client BusinessClient and the EJB method doAction are shown, and the final result. Output for the other components has been left out for layout purposes. In this case the recommendation of XacT to the administrator of the workflow is to assign the activity to the organizational role minister.

6. SUMMARY AND CONCLUSION
In this paper we have presented a methodology that allows to specify consistent access control policies in different layers of an application. We have given implementation details of the eXtreme access control Tool (XacT) and we have discussed a more generalized architecture. The aim of the tool is to prevent authorization exceptions from occurring when fine-grained access control policies are specified on each layer in the application. This way over-privileged accounts can be prevented. Through an initial experiment with the tool on a prototype implementation developed in the context of the EU FP6 project eJustice we have concluded that our methodology is also feasible in practice. Through an initial experiment with the tool on a prototype implementation developed in the context of the EU FP6 project eJustice we have concluded that our methodology is also feasible in practice. In future work we plan to perform experiments on larger-scale applications and we would like to present a more formal performance analysis. Extension of our tool to other computing environments like .NET could then also be considered.

The Aspect-Oriented Programming (AOP) technology has shown great potential to us while solving this problem. The byte-code- and binary-weaving capabilities of some AOP tools would also make it possible to use XacT on applications that integrate legacy systems, so when no source code is available. We have also made abstraction of possible differences in naming schemes for roles that typically occur in heterogeneous applications. Finally, it could also be useful to determine the minimal set of access rights needed to perform a certain activity, instead of only retrieving the organizational roles that have at least this minimal set of rights assigned.

7. REFERENCES
Figure 5: Sample report generated by XacT


