Privacy & Authorization Policies - Validation & Assurance Techniques
(Two Chapters for a book titled "Identity and Security" by FutureText)

Ramaswamy Chandramouli

8 Identity-Enabled Technical Privacy Policies – Formulation & Validation

8.1 Introduction

The primary motivation behind formulation of any privacy policy (policy in the context of this chapter refers to technical policies defined, specified and enforced within the relevant information systems) is to restrict the disclosure of identity of an individual (in certain locations, events or transactions) due to the potential for economic, reputation or safety loss to that individual. Hence every enterprise dealing with individually identifiable information (IIF) has to have privacy protection policies and a supporting IT architecture for specification and enforcement of those policies.

Some of the drivers for enterprise privacy policies are maintaining market competitiveness, maintaining corporate reputation and complying with government regulatory requirements. The last one is the main driver for enterprises in some key economic sectors as the government has enacted sector-specific laws to protect the privacy of citizens participating in transactions in that sector. Examples of such laws in US are the Health Insurance Portability and Accountability Act (HIPAA) [8.1] for privacy protection of healthcare consumers and the Gramm Leach Bliley Act (GLBA) [8.2] for privacy protection of consumers of financial services.

Let us call the class of information for which an enterprise has responsibility for privacy protection as customer information. A customer information class may have many information types within it. For example, the healthcare customer class may have information types such as demographic information type, allergy information type, clinical information type etc. whenever an enterprise wants to formulate privacy policies for any customer information class it is dealing with, it usually undertakes the following steps (or its equivalents):

Step 1: Identify privacy labels that are appropriate for the domain (based on regulatory or corporate mission requirements) and the customer information class it is dealing with.

Step 2: Formulate policy statements associating the identified privacy labels with appropriate information types and the corresponding information lifecycle parameters (relating to collection, storage, user base, retention and dissemination).

Step 3: Enforce privacy labeling semantics and the privacy protection measures (based on lifecycle parameters arrived at in the previous step) through either access control mechanisms (for restricting read, modify or delete of those information types) or through

information flow control mechanisms (for restricting export of information types to outside entities). Here the term outside entities refers to those entities that are outside the circle of trust that is formed by a federation created by the enterprise.

The assurance that an enterprise is in compliance with a mandated set of privacy requirements comes from the effectiveness of the above 3 steps. The factors affecting Step 1 are the experience of the company in the domain or sector and its general awareness of the laws under which it should operate. The effectiveness of Step 3 depends upon the security mechanisms it has deployed in its IT infrastructure. While these two steps (i.e., Step 1 and Step 3) are important, the most critical factor that determines the integrity of an enterprise's privacy protection process is Step 2: proper assignment of appropriate information types to the set of privacy labels identified for the class of information under which the information types fall.

An enterprise associates information types with privacy labels through one or more of the following methods:

- Directly from user preferences (or consent options) associated with the information types
- Indirectly through business process analysis of the transactions in which the information type is involved and the privacy impact of its exposure.

In the first method above, there is high degree of confidence that the overall enterprise privacy policy framework is in compliance due to the fact that the user preferences (the single category of factor that determines privacy violations) are embedded in the information type. However, in the second method where information types are determined by transaction units, there is the possibility that the assigned privacy labels do not hold up their semantics. This is due to the fact that in many business transactions the data items within an information type and across information types are correlated and have varying degrees of dependencies (full, partial, etc.) The implication of this dependency relationship is that it is possible to infer a data item with an information type that is labeled with a stricter privacy label (e.g., Privacy Intrusive) using a combination of two or more data items from information types labeled with a less stringent privacy label (e.g., Pseudo Anonymous). Hence we argue that an inference analysis approach that identifies all the data dependencies within and among information types and using these as the basis for detecting violations of privacy labeling semantics and making the necessary adjustments to the privacy labels, if needed, is a necessary task in the formulation of compliant privacy policies.

In tune with the above perspective on the privacy policy formulation, we devote a major portion of this chapter to a description of an Inference Analysis methodology based on Disjunctive Logic Programming [8.3]. This methodology was developed by the co-author of this book and is described in [8.4]. To properly identify partial orders that may exist between privacy labels we first develop the privacy label taxonomy in section 8.2. We also illustrate the instantiation (customization) of this taxonomy in this section. In the next section (8.3), we demonstrate a way of expressing the data dependency relationships within and between information types using inference relations. In section 8.4 we describe a method for detecting violations of privacy labeling semantics using the set of identified inference relations as well as an approach for correcting the labeling semantics violations either by modifying the contents of information type or the associated label or both. In section 8.5 we illustrate the simple case of associating

information types with privacy labels when privacy labels are directly obtained from user preferences.

8.2. Privacy Label Taxonomy

The analysis of data dependencies as a means to detect privacy labeling semantics violations has meaning only in situations where there exists some partial order (hierarchy and mutual incompatibility) among privacy labels. Hence developing the privacy label taxonomy is the first step in our inference analysis methodology for validation of privacy label assignments.

8.2.1 Privacy Label Taxonomy Development Logic

The privacy label associated with an information type denotes the degree to which it reveals information about the identity of an individual or a group. If the information type by itself reveals all the data that can be attributed towards an individual, then we label the information type as “Individually Identifiable Information (IIF)” or “Privacy-Intrusive”. An example of an information type that is Privacy-Intrusive is the hospital’s patient demographic information record. This record contains the full name, SSN, insurance policy number and the full address of the patient, essentially revealing all the information that is directly attributable to the person with the given full name in the record. The information that is revealed by an information type labeled as “Privacy-Intrusive” is viewed as individually identifiable from the point of view of all constituents that receive the information (recipient). In other words it is not the case that the given information type reveals individually identifiable information from the point of view of some recipients (with better contextual and domain knowledge) and not others. Since revelation of IIF is total and absolute, there are no degrees of privacy exposure involved with respect to different recipient groups and hence the “Privacy-Intrusive” label has no sub classes.

If an information type carries information based on a pseudo identifier (e.g., mortgage account number in a mortgage payment slip), then it can be labeled as “Pseudo Anonymous”. An information type with privacy label “Pseudo Anonymous” needs external information (that is not part of the enterprise database) or a related record in the enterprise database in order to attribute an instance of that type to an individual. The pseudo anonymity property of the information type carrying the “Pseudo Anonymous” label is not absolute (not an intrinsic property of the information type). In other words, what is Pseudo Anonymous for one class of information recipient may be Privacy-Intrusive for another class of information recipient. For example, the list containing the patient ID, ward number and bed number that is given to the hospital’s facilities department is Pseudo Anonymous from the point of view of the facilities department alone. The same list given to a nurse station is not Pseudo Anonymous, because a ward nurse using the work station in the nurse station will be able to get the name of all patients assigned to that ward at any given time using patient IDs as search keys. Hence the label “Pseudo Anonymous” is relative to a recipient group and some selected data items are sanitized or masked when sent to that group so that group members cannot link the received information to a specific individual. This gives rise to different classes of “Pseudo Anonymous” labels depending upon which particular data item is masked. For example the “Pseudo Anonymous (Patient Name)” label class will contain information types where the “Patient Name” will be masked (by providing a pseudo identifier “Patient ID”).

Similar to an information type labeled “Privacy-Intrusive”, an information type that is labeled as “Fully Anonymous” maintains this property with reference to all classes of recipients. Hence there are no sub classes within “Fully Anonymous” label. Based on the degree of “Individually Identifiable Information” revealed, the privacy labels fall into a hierarchy with the following partial order:

Privacy Intrusive > ***Pseudo Anonymous >*** ***Fully Anonymous***

In the case of the “Pseudo Anonymous” label, there are multiple classes and the degree of “Individually Identifiable Information” revealed by information types in various classes depends upon target recipient groups (with different data items masked) which are mutually disjoint with no containment relationships among them. Hence there is no dominance relationship among “Pseudo Anonymous” label classes (similar to the partial order given above) and these classes are termed mutually incompatible. The single “Privacy Intrusive” label class, multiple incompatible “Pseudo Anonymous” label classes and the single “Fully Anonymous” label class, together with the partial order shown above, give rise to a lattice structure.

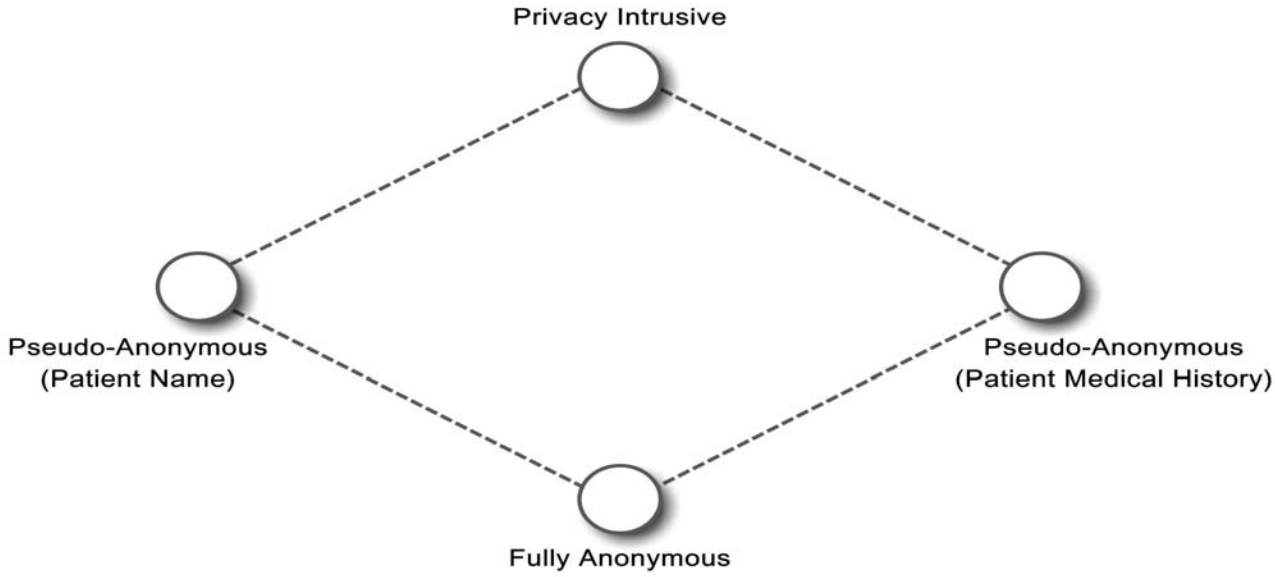

Figure 8.1: Addressing the extremes—privacy intrusive and fully anonymous

8.2.2 Instantiation (Customization) of Privacy Label Taxonomy

The generic privacy label taxonomy (a lattice) could be customized for each enterprise by identifying appropriate classes of "Pseudo Anonymous" label. Then, using a combination of text-based privacy policies for the enterprise and/or privacy attribute (e.g., purpose, recipient, etc.) values associated with logical data structures (e.g., relational tables and records), the information types associated with each privacy label class are determined. The information types associated with a privacy label (we use the terms privacy label class and privacy label interchangeably since in many instances there is only one class within a given label) could consists of the contents of an entire logical data structure (e.g., patient registration table), subset of records from a logical data structure (e.g., Records of patients from the patient registration table who checked into the drug rehabilitation ward) or one or more selected fields within a record (e.g., SSN field in the patient registration table).

The generic privacy label taxonomy customized for a large hospital enterprise is shown in Figure 8.1. The patient demographic information record, the patient registration information record, the patient clinical information record and the patient financial information record carry the "Privacy-Intrusive" label. Under the "Pseudo Anonymous" label, there are two classes – "Pseudo Anonymous (Patient Name)" and "Pseudo Anonymous (Patient Medical History)". The lab test request information record" and the patient referral information record come under these classes, respectively. The lab test request is submitted to an affiliated laboratory of the hospital using the pseudo identifier (patient ID). Hence it is anonymous or pseudo anonymous with respect to the patient name. In a referral request to a doctor outside of the hospital, a particular diagnosis is sent for expert opinion and not the entire patient medical record. Hence the patient referral information record is pseudo anonymous with respect to patient medical history.

The privacy label classification for all the information records (information types) in a hospital along with the name of the external entity (recipient) to which it is disclosed is given in Table 8.1. The privacy label associated with an enterprise information type is used solely for the purpose of restricting information disclosure to outside entities. For the purpose of information handling within the enterprise, it is the value of privacy attributes (purpose, recipient, retention, etc.) that come into play. Some times even information types that are not currently disclosed to outside entities are assigned privacy labels (see Table 8.1 – against Privacy-Intrusive privacy label) because the information contained in them may be required to be disclosed to outside entities as one-time requests or for special occasions.

Privacy Label	Information Types and External Recipients
Fully Anonymous	(a) Aggregate Data on Heart Ailments Information – Public Health Agencies (b) Aggregate Data on Cancer Treatments Information – Public Health Agencies
Pseudo Anonymous (Patient Name)	Lab Test Request Information – Clinical Labs
Pseudo Anonymous (Patient Medical History)	Patient Referral Information - Specialists
Privacy-Intrusive	(a) Patient Demographic Information - NONE

(b) Patient Registration Information - NONE
(c) Patient Clinical Information - NONE
(d) Patient Financial Information – Insurance Companies

Table 8.1 – Privacy Label Taxonomy

8.3. Formulation of Inference Relations

Let us assume that there are two facts: A and B. Fact A is explicitly known. Fact B is not known because it is not meant to be disclosed. On the other hand, if there is an inherent dependency between two facts A and B, knowing one fact A may help us to know completely or partially the fact B. In other words if we know A then we can infer B. If such a situation exists then we express the relationship between the two facts A and B using a mathematical expression called the inference relation represented as:

A => B

In the above inference relation, A is called the known fact and B is called the inferred fact. An inference relation can also be expressed as a logical implication using the predicate "*known*" as follows: *known (B) ← known (A).* A predicate with a parameter or term (the one inside the parenthesis) is called an atom. An atom is also called a logical assertion or fact. In a logical implication, the atoms to the right of the implication symbol are called antecedents and the ones to the left of the symbol are called consequents. The generic name for a logical assertion or a logical implication is called logic formula. This is due to the fact that a logical assertion can be looked upon as a logical implication without a consequent. A set of logic formulae constitutes a logic program.

A logic formula instead of containing a single atom in its antecedent and consequent could contain a conjunction of atoms as shown below:

known(D) & known(B) ←known (A) & known (C) …(8.1)

However, using laws of predicate calculus [8.5] the above logical implication can be expressed as two equivalent logic formulas:

known (B) ← known (A) & known (C) …(8.2)
known (D) ← known (A) & known (C) ...(8.3)

Based on the above properties, there will be no loss of generality if we assume that the set of logic formulae represented by logical implication of the type in (8.1) consists of a conjunction of atoms in their antecedent and a single atom in their consequent. Logic programs consisting of such types of formulae are called definite logic programs. An interpretation I associated with a definite logic program P (or for that matter any logic program) is a set of ground atoms (those that do not contain a variable as a term) that satisfy P. An operator that operates on one interpretation and maps it to another interpretation and whose repeated application leads to an invariant interpretation Ifp is called a closure operator Tc. The invariant interpretation obtained by repeated

application of a closure operator is the called the fixed point and contains all the logical consequences of the definite logic program P. The immediate consequent operator that uses the ground atoms in the current interpretation and generates ground atoms corresponding to the consequents in at least one other logical implication (not used in previous application of the operator) is one such closure operator. Since each logical implication in a definite logic program contains only a single atom in its consequent, the logical consequence or the fixed point for a definite logic program (that contains a set of such formulae) also contains only atoms (specifically ground atoms). The set of all logical consequences for a definite logic program is called a minimal model and since there is only one logical consequence or fixed point, the minimal model for a definite logic program is also unique.

When logical implications (or in general logic formulae) are representations of inference relations, the logical consequence of a set of logical formulas represent a set of inferred facts. Hence there is a unique set of inferred facts when all inference relations contain a single inferred fact or a conjunction of inferred facts. On the other hand there could be situations where inference relations could contain a disjunction of inferred facts. The associated logical implications would then contain a disjunction of atoms in their consequent as given below:

known (C) V known (D) ← known (A) & known (B) … (8.4)

A logic program that contains logic formulae with disjunction of atoms in their consequent is called a Disjunctive Logic Program (DLP). Let us look at a disjunctive logic program P' that contains the following logical implications:

P' = *{ known (C) V known (D) ← known (A) & known (B), known (E) ← known (C) }*

Let the initial known state or interpretation for the above logic program be I = *{known (A), known (B)}*

Using a closure operator Tp (similar to the Tc operator we used for definite logic program P) we obtain the following fixed point. Ifp = *{ known (C) V known (D), known(E) V known(D), known(A), known(B) }*

In the case of the definite logic program P, the logical consequences consists of just ground atoms and hence gave rise to a unique minimal model. In the case of the disjunctive logic program since the logical consequences contain disjunctions, they give rise to the following set of minimal models as follows:

MM(1) = *{ known(A), known(B), known(D) }*
MM(2) = *{ known(A), known(B), known(C), known(E) }*

We thus have two minimal models. The first minimal model shows D as the inferred fact (since facts A and B are known – as per the initial state). The second minimal model shows that facts C and E are inferred facts. This is intuitively obvious from our logical implication (8.4) that shows that fact C or fact D can be inferred from facts A and B. If fact D is inferred, then no further inferences are possible. On the other hand if fact C is inferred, fact E can be inferred using the second logical implication in the DLP P'.

The question to be addressed in this situation is the following: Now that we have a set of minimal models (and hence multiple logical consequences), which set should be taken as the one containing the most likely set of inferred facts? In other words, we need an uncertainty reduction policy here. There are two types of uncertainty reduction policies that are possible:

• **Risky Uncertainty Reduction Policy:** By this policy we can assume that the set of facts that are most likely to be inferred are the ones that are found in both minimal models. When we follow this policy we obtain the set of inferred facts by taking an intersection of the minimal models MM(1) and MM(2). In the above example this means that only facts A and B will be inferred. But then A and B are already known facts and in effect this shows that no new fact will be inferred. This is contrary to the underlying semantics in formulating the logical implication 8.4. There we specified that either C or D will be inferred only because in some instances C will be inferred and in other instances D will be inferred. Choosing a policy that runs counter to that known inference semantics is not an acceptable solution. Hence this policy has a WEAK NOTION OF PRIVACY.

• **Conservative Uncertainty Reduction Policy:** By this policy we assume that the set of inferred facts is the union of the atoms (facts) in both minimal models. By taking the union of atoms in MM(1) and MM(2) we find that facts D, C and E will be inferred. If we examine the logical implications in DLP P' (inference relations), it means that we are implicitly assuming the fact that both facts C and D will be inferred although the original intent in formulating logical implication 8.4 is that C will be inferred in some instances and D will be inferred in some other instances. This policy of taking the union of facts in the minimal models MM(1) and MM(2) is thus a conservative policy. However, since the focus here is on information disclosure this policy has a STRONG NOTION OF PRIVACY.

8.4. Detecting Violations of Privacy Labeling Semantics

As shown by the theory in the previous section, the first step for performing inference analysis is to formulate inference relations. The privacy policy context we have chosen is a large hospital enterprise. The inference relations therefore involve information types handled and disclosed to outside entities by hospital IT systems. To formulate inference relations correctly for specialized domains such as a hospital requires a great deal of domain knowledge. The domain knowledge in our current context covers the following:

• All the business processes of the hospital – patient registration, ward allocation, staff scheduling (both healthcare staff as well as support staff), sending lab test requests to affiliated clinical labs, sending referral requests for expert opinions of specialists, providing sanitized information to public health officials and social workers, etc.

• All the information handled in the various business processes – these include the patient demographic information, patient registration information, patient clinical information (lab tests, drug prescriptions, treatment regimens, diagnosis, etc.), insurance billing information, patient co-payment information, lab request information, lab results information, etc. Inference relations show the information types that could be inferred in a given enterprise domain without being explicitly provided using the already

available/known information. The amount of information that can be inferred by a user of that information is dependent upon the domain knowledge gained over time as well as the semantic relationship among the data items stored in the enterprise's information repositories. For example, consider the data item John Smith's salary. The value of this data item should strictly be known only to John's supervisor, his department manager and the payroll officer processing the payroll for John's department. As far as rest of the organization is concerned, John's salary must be confidential. However it is known that John is the project manager for a $5M project and that only officers above the rank of GS-15 are assigned to those projects. Since the salary ranges for the rank GS-15 is known, John's salary could be inferred within a given range. In this case although the exact salary is not revealed or disclosed, there is considerable reduction in uncertainty with regard to what its possible value can be. This type of uncertainty reduction is also included under the term inference. Now the question is: given the privacy label assignment (hereafter called the current label) for a particular information type (hereafter called current information type) and the set of inference relations involving data items within that information type as well as data items within information types assigned to the same label or to labels lower in the hierarchy, does the assignment satisfy the strong notion of privacy? The general strategy we adopt to answer the above question is the following:

• Analysis-Step 1: Formulation of Logic Program: The first step is to enumerate all the inference relations in which the data items belonging to the current information type participate either fully or partially. A full participation of an information type within an inference relation means that all antecedents in logical implications come from the current information type. In addition there could be other inference relations where some of the antecedents are data items from the current information type and the others are publicly known (e.g., pay scale structure). All these inference relations (associated logical implications) together with all the individual data items in the current information type form the Logic Program.

• Analysis-Step 2: Compute the Logical Consequence: Compute the logical consequence of the formulated logic program in the previous step using the strong notion of privacy based on the theory outlined in Section.

• Analysis-Step 3: Analyze the Logical Consequence to Detect Any Violations to Privacy Labeling Semantics: Analyze whether the data elements in the logical consequence either individually or collectively belong to any information type with a privacy label that dominates the current label. If this occurs, it denotes that the data items in the current information type can be used to infer an information type that has a privacy label that dominates the current privacy label. In other words, the current information type does not satisfy the underlying semantics for the current label—the current information type violates the labeling semantics.

• Analysis-Step 4: Correction Process for Restoring Privacy Labeling Semantics for the Current Information Type: Examine the data items within the current information type that leads to the inference of data items in the information type belonging to the dominated privacy label. Remove one or more of those items thus changing the composition of the current information type. Repeat Steps 1, 2 & 3 until the current information type does not violate the strong notion of privacy.

Let us now consider the information type {Lab Test Request Information}. This is the information type that contains the details of the request for conducting a lab test on a patient. This information is generated and sent by the hospital to an affiliated test lab. The business process involved here is that the lab conducts the test and sends the test results back to the hospital. The lab is paid by the hospital for its services. The lab schedules the patient visit to the lab site through the hospital or in many cases the hospital itself collects the specimen required and sends it to the lab. Since the lab does not have to deal with the patient who is undergoing a clinical test, the lab need not know who the patient person is and the demographic information about that patient. Hence the lab test request information sent by the hospital has the privacy label “Pseudo Anonymous (Patient Name)”. For the purpose of maintaining communication with the lab as well as to associate the lab test results with the correct patient, the lab test request is made out using the patient ID as is the identifier. If multiple tests are ordered for a patient, a combination of patient ID + test request ID is used as an identifier to uniquely identify an instance of lab test request. Now our goal of inference analysis is as follows:

Using the lab test request information type that carries the privacy label “Pseudo Anonymous (Patient Name)” and using a set of inference relations involving data items from this information type as well as from other information types assigned to the same label or any dominated label, is it possible to infer an information type that is labeled as “Privacy-Intrusive”. Let us assume that the hospital maintains an information type called {substance-abuse-persons}. This category contains the data items name, address and abuse history. Since individually identifiable information of sensitive nature whose disclosure must be highly restricted is carried in this information type, this information type is given the “Privacy-Intrusive” label.

8.4.1 Formulation of Logic Program

Our first task now is to formulate the set of inference relations (in the form of logical implications) involving data items from lab test request information type. Now the lab test request information type is itself a composition of other information types and some atomic data items. This composition can be expressed as an inference relation. For example a hospital’s lab test request consists of patient ID + InsuranceNo + {Test_Details Information} (the first two are data items while the third is an information type). Now the reason that the hospital is sending the InsuranceNo of the patient for whom the test is ordered is that the hospital may know the exact type of tests to be conducted but may not know the medical code for the test and whether that test is covered under the patient insurance. Sending the InsuranceNo to the test lab will enable it to determine the exact medical code for the test and then verify from the insurer whether that test is covered or not under the medical coverage plan identified by the InsuranceNo. The inference relation showing the composition of lab test request information is formulated as:

known ({Lab_Test_Request }) ← known(patientID) & known(InsuranceNo) & known({ Test_Details}).. (8.5)

Now the lab looks at the test details and determines the medical code for the test. When the lab contacts the insurer to verify that the person’s insurance (identified by

InsuranceNo) has coverage for the particular test code, the insurance company often provides the name of the primary insured and his/her address. This is due to the federation model and the trust relationship between the lab and the insurer as well as due to the fact that the lab needs to confirm the name and address of the primary insured in some cases. These cases are the test requests that the lab receives from group medical practices who unlike the large hospitals place the onus on collecting the insurance payment on the test directly on the lab itself. However it may be the case that the primary insured may not be the patient in many cases but may be a family member covered under his/her insurance policy. Taking into account this uncertainty, the inference relation that deals with information exposure due to the InsuranceNo is formulated as follows:

known(PatientName) V ({known(Primary_Insured_Name) & known(Address)})← known(InsuranceNo) ----- (8.6)

Since in the majority of the cases the person who is covered under the insurance policy of another person is a dependent living under the same roof, there is no distinction made between the address of the patient and the address of the primary insured. Now knowing the details of the lab tests contained in the lab test request information together with the patient name and his/her address, the lab essentially can infer the persons who are listed in the information category {substance-abuse-person}. This inference relation is shown as:

known({Substance-Abuse-Person})← known({Lab_Test_Request})& known(Primary_Insured_Name) & known(Address) ----- (8.7)

Now the logical implications 8.5 through 8.7 together with the known facts PatientID, InsuranceNo, {Test_Details} represented by the following logical assertions (8.8 through 8.10) constitutes our Disjunctive Logic Program (DLP).

known(PatientID) --- (8.8)
known(InsuranceNo) --- (8.9)
known({ Test_Details}) --- (8.10)

8.4.2 Computing the Logical Consequences

The logical consequence I” of the DLP represented by logic formulae (8.5 through 8.10) is computed by repeated applications of the fixed point for the DLP as follows:

I”={*known(PatientName)V*
({known(Primary_Insured_Name)& known(Address)),
known(PatientName) V {Substance_Abuse_Person } ,
Known(Lab_Test_Request),
known(PatientID),
known(InsuranceNo),
known({ Test_Details}) }

Now the minimal models for the logical consequence are the following:

MM(1)=*{known(PatientName),*
known(Lab_Test_Request),
known(PatientID),
known(InsuranceNo),
known({ Test_Details}) }
MM(2)=*{known(Primary_Insured_Name),*
known(Address),
known{Substance_Abuse_Person } ,
known(Lab_Test_Request),
known(PatientID),
known(InsuranceNo),
known({ Test_Details}) }

8.4.3 Analyzing the Logical Consequences

The facts revealed in the logical consequences (and hence in the minimal models) is consistent with the logic that the substance-abuse-person information type is only revealed if the primary insured person is himself/herself the patient. Now our strong notion of privacy stipulates that the union of facts in the minimal models should be treated as the total set of inferred facts and if we take this union then the substance-abuse-person information type becomes an inferred fact. On the other hand if we take the intersection of atoms in the minimal models, no more new information other than the original known facts like lab test request, patientID, InsuranceNo and Test_Details will be in the set of inferred facts. By our privacy label classification the information type {substance-abuse-person} is clearly having the label “Privacy-Intrusive”. Hence this is a case where an information category {substance-abuse-person} with a privacy label (i.e., Privacy-Intrusive} which dominates the current privacy label (i.e., Pseudo Anonymous (Patient Name)) is revealed using one of the information type (i.e., Lab Test Request) belonging to the current privacy label and a set of inference relations.

8.4.4 Correction Process for Restoring Privacy Labeling Semantics for the Current Information Type

Having identified that there is inference of information with a privacy label that dominates the current label (or the current information type violates the labeling semantics), the set of all inference relations (logical implications) is examined to see as to which one is responsible for the violation. The programmatic way to do this is to take each of the set of ground atoms in the minimal models and backtrack to examine as to which ones of the known initial facts leads to this violating information disclosure. For our example we could visually see that the InsuranceNo data item is the one that leads to this privacy violating information disclosure. Hence to take care of this privacy violating information disclosure, the hospital should remove the InsuranceNo data item from the lab test request information type. Since the purpose of the hospital sending this InsuranceNo is to verify coverage, the hospital should take the responsibility for determining the correct medical code for the test and for verifying the coverage for this test code for the insured person directly with the insurance company (insurer) before sending of the test request to the lab. This definitely requires a change in the business process associated with

generating lab test request. Thus we see that ***privacy policy compliance measures include re-design of business processes in many enterprises***.

8.5 Assigning Information Types to Privacy Labels obtained from User Preferences

The inference analysis described in the previous section is based on the fact that we do have an existing assignment of information types to various privacy labels in the taxonomy. In this section we illustrate the logic of arriving at these assignments. Understanding the logic of associating information types with privacy labels improves the effectiveness of the inference analysis framework in two interrelated ways. First, it helps to understand the data semantics better since information types associated with a privacy label need not have 1:1 correspondence with logical data structures in the enterprise information systems. Second, a better understanding of data semantics helps to formulate inference relations that are to a great degree consistent and complete. The case study here involves a home audio/video manufacturing enterprise wanting to determine the subset of customer information records it can disclose to three outside entities: authorized sales agencies, authorized service providers and value-added resellers. It has come up with privacy labels: Mass-Mailers, Prudent-Buyers and Discount-Lovers for customer information sent to these external entities respectively (here we are not considering any taxonomy for privacy labels). In order to determine records for which each of these labels apply, the enterprise collects the following values for the “Disclosure Consent” privacy attribute (user preference):

• Consent Option 1: His billing/email address will be used for mass mailing for new product promotions.
• Consent Option 2: His billing/email address will be used for sending deals about extended warranty packages but for a period not exceeding six months from the date of purchase.
• Consent Option 3: His billing/email address will be used for sending discount coupons.

The enterprise’s customer information database contains of the following logical data structures (i.e., relational tables):

• Targeted Customer Information Table: This table contains information about targeted customers and has been obtained from information service companies on a subscription basis. Since the information service companies collected this information from the target population after obtaining their consent for disclosure, the records in this table are deemed to have the value “Consent Option 1” (for the “Disclosure Consent” attribute) implicitly.

• Historical Customer Information Table: These are customers who have placed orders with the enterprise in the past but have not placed any orders within the last six months.

• Active Customer Information Table: These are customers whose orders are either pending or have placed orders within the last six months. Now using the attribute values for “Disclosure Consent” attribute, the information type associated with each of the privacy labels are determined. This information type could involve all the records from a particular customer information table category or a subset of records from one or more

of the above categories. The logic for determining these information types for each of the three privacy labels is described below and the output is shown in Table 2

• Mass-Mailers: This is the list of customers who have consented explicitly or implicitly to receive email/postal mail notifications of new product promotions (Consent Option 1). This information type can thus include all records from the Targeted Customer Information Table and the subset of customers from the Historical Customer Information Table and the Active Customer Information Table who have consented to the mass mailing option (Consent Option 1).

• Discount- Lovers: This is the list of customers to whom special discount coupons will be sent. This will only include customer records from the Historical Customer Information Table and the Active Customer Information Table whose "Disclosure Consent" value equals "Consent Option 3".

• Prudent-Buyers: This is the list of customers who have chosen "Consent Option 2". This list can only include subset of customers from the Active Customer Information Table, since the Historical Customer Information Table contains records for customers who have not placed any orders for the last six months and warranty package offers cannot be sent to them as per the conditions of "Consent Option 2".

Readers, please note that in the above example, since privacy labels are directly based on user preferences and do not form any partial order, there is no question of any of the information types violating the privacy labeling semantics. Additionally, user preferences become an intrinsic attribute of the information type and override the privacy implication caused due to data dependencies.

Table 8.2: Privacy Label Assignment for Customer Information Types

Privacy Label	**Associated Information Type**	**Consent Option**
Mass Mailers	(a) All Targeted Customer Information Records (b) (Subset of) Historical Customer Information Records (c) (Subset of) Active Customer Information Records	- Option 1 (implicit) (for (a)) - Option 1 (for (b)) - Option 1 (for (c))
Discount Lovers	(a) Historical Customer Information Record (b) Active Customer Information Record	- Option 3 - Option 3
Prudent Buyers	Active Customer Information Record	- Option 2

8.6 Summary

To obtain assurance that their privacy policies are compliant with regulatory requirements and effectively meet corporate goals, enterprises have to perform these early life cycle tasks as part of the overall privacy policy formulation and enforcement:

- Proper identification and instantiation of privacy label taxonomy that is consistent with their line of business
- Assignment of information types to privacy labels in a way that data dependencies do not violate the labeling semantics

Once these tasks are successfully accomplished, the correct operational enforcement of these labeling semantics depends upon proper choice of security mechanisms and secure configuration of the hardware/software/firmware implementing these mechanisms.

References

[8.1] Health Insurance Portability and Accountability Act (HIPAA), http://www.hep-c-alert.org/links/hippa.html

[8.2] Gramm-Leach-Bliley Act: Financial Privacy and Pretexting, Federal Trade Commission, http://www.ftc.gov/privacy/glbact/index.html

[8.3] Lobo Minker and Rajasekar. Foundations of Disjunctive Logic Programming, MIT Press, Cambridge, 1992.

[8.4] Chandramouli, R., “Privacy Protection of Enterprise Information through Inference Analysis” – Proceedings of the 6th IEEE International Workshop on Policies for Distributed Systems and Networks to be held in Stockholm, Sweden, June 2005.

[8.5] C.L. Chang and R.C.T. Lee. Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York 1973.

13. Assurance for Identity Enabled Authorization policies

13.1 Introduction

The PCCP (policy checking/certification and compliance point) is an important component of a distributed policy architecture. As its name implies, the PCCP performs the functions of policy verification and validation and hence ensures the trustworthiness of policy enforcement point (PEP) and the legitimacy of the decisions that are returned from a PDP. The PCCP is closely integrated with the policy management point (PMP) and in many implementations is a subcomponent of the PMP. In order to effectively perform its function, the PCCP uses a policy validation framework.

Authorization (or access control) policies, just like device policies and privacy policies, are an important class of policies for safeguarding enterprise resources. Specifically, authorization policies provide confidentiality and integrity of enterprise IT resources by placing restrictions on reading and modification of these resources. Hence it is imperative that there should be a policy validation framework in PCCP for validation of authorization policies.

Enterprise authorization specifications specify the access rights of various users or roles to enterprise resources and are used by a module of IT systems to enforce access restrictions during the operation of these systems. This module is called the access control mechanism. Hence the first point of trust in the overall access control mechanism is the underlying data it uses (i.e., enterprise authorization specification). The enterprise authorization specification in turn should reflect the intent of authorization policies. Hence it is necessary to validate the enterprise authorization specification for conformance to authorization policies. We will call a methodology or approach to accomplish this as the authorization policy validation framework.

In this chapter, we describe an authorization policy validation framework. This framework was developed by the co-author of this book and outlined in [13.1]. The description of policy validation framework is organized as follows. In section 13.2 we describe the background information and the overall approach for the authorization policy validation framework. The entire framework – its building blocks, sample encodings of enterprise access specifications and policy constraints and the output it generates is covered in sections 13.3 through 13.5. In section 13.6 we provide a summary of the benefits and limitations of our authorization policy validation framework.

Before we lead the reader into the details of the authorization policy validation framework, we would like to point out its two salient features. They are:

(1) It is an out-of-band policy validation framework since the access control mechanism (the PEP in the authorization policy context) is not one of its components.
(2) The authorization specification, the basic artifact used by the validation framework is expressed using the platform-neutral XML language and is based on a high level role-based access control (RBAC) model [13.2].

The choice of the high level RBAC model is due to two reasons:

(1) RBAC lends itself to expression of many different types of authorization policies (such as separation of duty, least privilege, etc.)
(2) Authorization specifications expressed using an RBAC model can be easily mapped to native access control model in many platforms, be they user-centric (e.g., capability lists) or resource-centric (e.g., ACLs in Windows, permission bits in Unix).

13.2 Authorization Policy Validation Framework – Background & Overall Approach

An access control mechanism provided by or within any software (e.g., operating system, DBMS) is the executable module for controlling access to resources under the control of the software. Every access control mechanism is built on a structural framework called the access control model that provides the means for specifying authorizations (or access rights) for resources. An access control model is based on certain concepts involved in interaction with resources. These concepts can be broadly described as being made up of entities (e.g., subject, object, operations, permission or right, user, role, label, group etc) and relations (e.g., the combination of an object and operation defines a permission) that describe the nature of association between entities. The deployment of an access control model for an enterprise environment is called an access configuration. An access configuration for a given enterprise contains instances of model entities for that enterprise (e.g.,Teller, Loan Officer, etc. are instances of the “role” entity for a commercial bank) and is expressed as the enterprise authorization specification. The safety of an access configuration is defined as the state where the configuration does not violate enterprise access control/authorization policies. Any approach that seeks to verify the safety of an access configuration based on a particular access control model should have the capability to capture (express) the relevant authorization policies using the same entity and relation instances in the access configuration.

A common approach adopted to implement and validate authorization policies is to augment the access control model with expressions called constraints (also called policy constraints). There are however many practical limitations in ensuring that the enterprise authorization specification is safe (does not violate policy constraints). The first one is the limitation of the underlying access control model. Since policies are specified using model entities and relations, it should be obvious to many practitioners that some access control models are more amenable for expression of complex enterprise policies than others. In general, higher the level of abstraction of model entities, more is the policy definition capabilities of the model. Secondly, even if the underlying access control model does provide policy definition capabilities, the access control mechanism may not provide features for specification of all the different types of constraints needed to capture those policy requirements. The above two limitations point the need for an out-of-band approach (independent of access control mechanism and the underlying software platform) to represent enterprise authorization specification and validate it for satisfaction of enterprise authorization policy constraints.

In this chapter we provide one such approach. We have represented the enterprise authorization specification for a commercial bank enterprise in XML. The authorization specification is based on the role-based access control (RBAC) model. The RBAC model itself is specified using XML Schema [13.3]. The RBAC XML Schema specification is then augmented with policy constraints using the Schematron constraint specification language [13.4]. The XML document containing the bank-enterprise authorization specification is then validated using the Schematron Validation Tool [13.5[].

With the above background, we now provide the more detailed description of the contents of sections 13.3 through 13.5. In section 13.3, we provide an overview of the various components in our authorization policy validation framework as well as the rationale for their choice. Choosing a version of RBAC model called Bank-RBAC model, which is applicable for our reference enterprise (i.e., a commercial bank), we describe the specification of the Bank-RBAC model using XML-Schema language in section 13.4. We will refer to the specification of Bank-RBAC model in XML Schema as Bank-RBAC XML Schema and the XML encoding of the bank-enterprise authorization specification based on the RBAC XML Schema as Bank-Authorization XML Data. In the same subsection we provide a sample encoding in XML of bank-enterprise authorization specification. In section 13.5, we illustrate the encoding of bank-enterprise authorization policies as constraints using the Schematron language. We also provide the policy violation messages that result from applying these constraints on the bank-enterprise authorization specification using the Schematron validator tool.

13.3. Authorization Policy Validation Framework Components

A framework for programmatic or tool-based validation of enterprise authorization specification should have the following components:

(a) Choice of the underlying access control model and a language for its specification
(b) A language for encoding enterprise authorization specification
(c) A language for specifying policy requirements as constraints using access control model entity and relation instances
(d) A tool (with a well-defined API) for programmatic validation of the enterprise authorization specification for conformance to access control model structure and policy constraints.

13.3.1 Choice of Access Control Model and Its Specification

Our motivation for choosing RBAC as the underlying access control model for the bank-enterprise authorization specification is that it is a sufficiently abstract model with configurations capable of expressing varied types of policies such as least privilege and separation of duties. RBAC has been widely implemented for different types of products such as database management systems, workflow systems and enterprise security management systems [13.6]. A brief description of RBAC models is as follows. The role-based access control model (RBAC) provides a generalized approach for representation of many types of access control policies (each describable only using a specific access control model) through the abstraction concept of roles. Many RBAC models have been proposed in the research literature [13.7] and the NIST RBAC

standard provides a taxonomy of RBAC models [13.8]. The RBAC reference model in the standard has four main entities – users, roles, privileges and sessions. Roles generally represent organizational functions (e.g., teller in a bank). Users are assigned to roles and privileges are assigned to roles as well. Users derive all their privileges by virtue of their role memberships. Users interact with the system through sessions and roles are assigned to a particular sessions as well. The user rights or permissions in a session are determined by the set of roles that are active in that session. Now the interactions among these four entities of the RBAC model results in the following relations:

(a) Role-Inheritance relation (RH)
(b) User-Role relation (UA)
(c) Privilege-Role relation (PA)
(d) User-Session relation (US)
(e) Role-Session relation (RS).

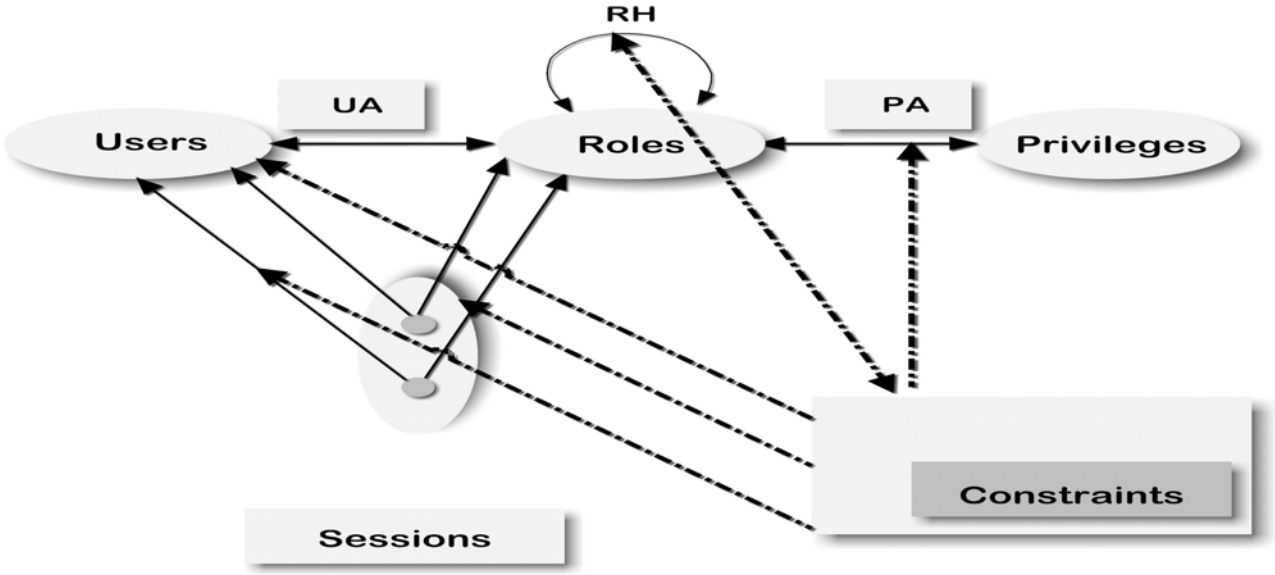

Figure 13.1 A schematic diagram of our reference RBAC model is given.

The Bank-RBAC model that we have chosen for illustration in our policy validation framework is based on the RBAC reference model described above but without the session entity and its two associated relations, US and RS. We have excluded the session entity since session is a platform-dependent artifact. For example, a DBMS session has a different set of parameters than an OS login session like Telnet. For the same reason, we have also excluded session-related constraints. To summarize our Bank-RBAC model consists of users, roles and privileges as entities and the following relations – role inheritance (RH), user-role relation (UA) and privilege-role relation (PA).

Our language for Bank-RBAC model specification is XML Schema since it provides constructs for specifying binary relations and hierarchical structures (the basic structural relationships of the RBAC model). XML Schema is one of the languages under the XML standard that is used for describing the structure of information within an XML document. Our choice of XML Schema over the other meta-data language DTD is due to the fact that XML Schema supports specification of cardinality and participation restrictions as well as rich data types (like enumerated data types). Further we need a means to augment the specification of the Bank-RBAC model with policy constraints. The XML

Schema language enables this feature as well by allowing the embedding of constraints in other languages under a special “annotation” tag. We have made use of this feature by embedding our policy constraints specified using the Schematron language within the XML-Schema representation of our Bank-enterprise RBAC model.

13.3.2 A Language for Encoding Enterprise Authorization Specifications

Our choice of XML Schema for Bank-RBAC model automatically provides XML as the choice for encoding enterprise authorization information. An advantage of encoding a structured information (such as bank-enterprise authorization specification) in XML is that there are special types of software called XML Parsers that could be used to extract information from XML documents based on its associated structure (that is specified through XML Schema document). These XML Parsers are based on standard application programming interfaces such as Document Object Model (DOM) [13.9]. These parser libraries implemented in various procedural languages enable an application program written in the corresponding procedural language to create, maintain and retrieve XML encoded data. With an API for extracting information, a program could be written to properly interpret the contents of the validated enterprise authorization specification (encoded in XML), and map them to the native access control structures in the access control mechanisms present in heterogeneous application systems and platforms within the enterprise.

It is useful to point out at this stage that XML Parsers can also be used to validate an XML document for conformance to the structure specified in an associated XML Schema document. Hence in our case the Bank-Authorization XML Data document (containing bank-enterprise authorization specification) can be validated for conformance to Bank-RBAC model specified through a XML Schema document. However as we pointed out earlier, XML Schemas can only be used for specifying data typing and cardinality constraints. These constraints are useful for properly specifying model entities and their associated binary relations. Hence XML Schemas can specify model-based constraints and therefore can be used to validate whether the Bank-Authorization XML data does indeed conform to the particular adaptation of the RBAC model for our banking enterprise (i.e., Bank-RBAC model).

13.3.3 A Language for Specifying Policy Constraints

We already alluded to the fact that the XML Schema with its support for data types, cardinality and participation constraints can handle structural constraints and hence all model-based constraints (being structural in nature) can be expressed through XML Schema. However policy constraints pertain to the enterprise domain and hence involve the contents of enterprise authorization specification. More specifically they involve the model entity and relation instances found in the Bank-Authorization XML Data. Further, studies have shown [13.10,13.11] that the content-based policy constraints are much more complicated than model-based constraints since they may involve complex logical expressions or rules.

One approach that has been adopted to represent domain constraints is to annotate an XML Schema that has been used for representing a model for a domain, with ontological information regarding the domain using pattern based languages such as RDF [13.12] and Schematron. In this paper we have annotated the XML Schema for Bank-RBAC

Model with Schematron constraints that specify rules that that the access control data (in Bank-Authorization XML Data) pertaining to the bank enterprise domain has to satisfy.

13.3.4 A Tool for Validation of Enterprise Authorization Specification

We have used a tool called the Schematron Validator for validating the bank-enterprise authorization specification (in Bank-Authorization XML Data) for conformance to policy constraints specified through the Schematron language. Since the Schematron Validator tool also validates an XML document for conformance to the referenced structure, it also automatically checks the XML encoded bank-enterprise authorization specification for conformance to the Bank-RBAC model specified through XML Schema. Hence using this tool we can validate the Bank-Authorization XML Data for satisfaction of both model-based and policy constraints.

13.3.5. XML Schema Specification of Bank-RBAC Model

The basic artifact for modeling any concept in XML Schema is the element. A name, type and a set of attributes can be specified for an XML Schema element. The type can be a simple data type like a ‘string’, or a complex data type. A complex data type in turn may involve additional elements. A data type can be an enumerated type (can only assume a value from a given set) as well. In addition a special data type called ‘ID’ is supported. This is often used as the data type for an attribute if that attribute uniquely identifies an instance of that element.

It is possible to specify certain structural constraints associated with an element. We can specify the maximum and minimum of times that element instance can occur in the XML document based on the XML Schema specification. We can also specify whether the use of an element or attribute is mandatory or optional.

As far as our Bank-RBAC model is concerned, all the entities (User, Role, Privileges) as well as relations (User-Role relation (UA), Role-Inheritance relation (RH) and Privilege-Role relation (PA)) are modeled as elements. Since these entities either contain multiple attributes (as in the case of elements representing User, Role and Privileges) or sub elements (as in the case of UA, RH and PA) relations, the data type associated is always a complex data type.

The specification of the User entity is as follows:

```
<xs:element name="user"     type="userType"/>
<xs:complexType name="userType">
<xs:attribute name="userID" type="xs:ID" use="required"/>
<xs:attribute name="fullname" type="xs:string" use="optional"/>
</xs:complexType >
```

The above definition of the data type ‘userType’ means that a user is represented as having two attributes ‘userID’ and ‘fullname’ with the former declared as a mandatory attribute and the latter declared as an optional attribute. Please note that the data type for ‘userID’ attribute is designated as ‘xs:ID’ which implies that the value for ‘userID’ attribute must be unique and hence no duplicates are allowed. The entity ‘Role’ is specified as follows:

```
<xs:element name="role"   type="roleType"/>
<xs:complexType name="roleType">
<xs:attribute name="roleID" type="xs:ID" use="required"/>
<xs:attribute name="rolename" type="validRole" use="required"/>
<xs:attribute name="cardinality" type="roleLimit" use="optional"/>
</xs:complexType>
```

To complete our definition of role component, we need to define the data types "validRole" and "roleLimit". The data type definition of "validRole" lists the set of permissible role names in the bank enterprise while that for the "roleLimit" is used to specify a number that stands for the minimum and maximum number of users that can be assigned to that role.

```
<xs:simpleType name="validRole">
    <xs:restriction base="xs:string">
<xs:enumeration value="BranchManager"/>
<xs:enumeration value="Customer_Service_Rep"/>
<xs:enumeration value="SD_Vault_Officer"/>
<xs:enumeration value="Loan_Officer"/>
<xs:enumeration value="Accounting_Manager"/>
<xs:enumeration value="Internal_Auditor"/>
<xs:enumeration value="Teller"/>
<xs:enumeration value="Accountant"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="roleLimit">
    <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

The privilege is a combination of a resource and operation. The privilege entity of the Bank-RBAC model is specified as:

```
<xs:element name="privilege" type="privilegeType"/>
<xs:complexType name="privilegeType">
<xs:attribute name="privID" type="xs:ID" use="required"/>
<xs:attribute name="resource" type="xs:string" use="required"/>
<xs:attribute name="oper" type="operType" use="required"/>
</xs:complexType>

<xs:simpleType name="operType">
    <xs:restriction base="xs:string">
<xs:enumeration value="Open"/>
<xs:enumeration value="Close"/>
<xs:enumeration value="Debit"/>
<xs:enumeration value="Credit"/>
  </xs:restriction>
</xs:simpleType>
```

We now provide the XML Schema representation for the User-Role Assignment (UA) relation of the Bank-RBAC model.

```
<xs:element name="UserRoleAssignment"
                type="URAType"/>
<xs:complexType name="URAType">
    <xs:sequence>
        <xs:element name="user" type="xs:IDREF"
                        maxOccurs="10"/>
    </xs:sequence>
    <xs:attribute name="role" type="xs:IDREF"
                use="required"/>
</xs:complexType>
```

The XML Schema representation for the role-inheritance relation (RH) is as follows:

```
<xs:element name="role_inherit" type="InheritType"/>
<xs:complexType name="InheritType">
<xs:attribute name="Inherit_ID" type="xs:ID" use="required"/>
<xs:attribute name="FromRole" type="validRole" use="required"/>
<xs:attribute name="ToRole" type="validRole" use="required"/>
</xs:complexType>
```

The Privilege-Role relation (PA) is specified in XML Schema as:

```
<xs:element name="RolePrivilegeAssignment" type="RPAType"/>
<xs:complexType name="RPAType">
  <xs:sequence>
                <xs:element name="privilege" type="xs:IDREF"
maxOccurs="unbounded"/>
  </xs:sequence>
<xs:attribute name="role" type="xs:IDREF" use="required"/>
</xs:complexType>
```

The specification of a construct to specify conflicting roles for checking static separation of duty constraints is specified as:

```
<xs:element name="ssd_roles" type="SSDType"/>
<xs:complexType name="SSDType">
<xs:attribute name="SSD_ID" type="xs:ID" use="required"/>
<xs:attribute name="BaseRole" type="validRole" use="required"/>
<xs:attribute name="ConflictRole" type="validRole" use="required"/>
</xs:complexType>
```

Finally the fact that the entire Bank-RBAC model is made of entities User, Role, Privilege and UA, RH and RP relations is specified in the XML Schema by creating a root element

called ‘BANK_RBAC_Model’ with elements representing the entities and relations as sub-elements.

```
  <xs:element name="Bank_RBAC_Model" type="BankRBACModelType"/>
<xs: complexType name=”BankRBACModelType”
<xs:sequence>
     <xs:element ref="user" maxOccurs="unbounded"/>
        <xs:element ref="role" maxOccurs="unbounded"/>
        <xs:element ref="privilege"  maxOccurs="unbounded"/>
        <xs:element ref="role_inherit"  maxOccurs="unbounded"/>
     <xs:element ref="ssd_roles"  maxOccurs="unbounded"/>
        <xs:element ref="UserRoleAssignment" maxOccurs="unbounded"/>
        <xs:element ref="RolePrivilegeAssignment"
                                        maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

Observe that some of the elements specified above do not have a name (like other element definitions we have seen before) but refers to the already defined elements through the value specified in the ‘ref’ attribute. The above XML Schema definition was verified to be syntactically correct using the XML Schema Validator tool – XML Spy [13.13].

13.4. Encoding the Enterprise Authorization Specification in XML

Now that we have developed an XML Schema specification of the Bank-RBAC model, we now encode the enterprise authorization specification in an XML document whose tag structure should correspond to the element definitions in the XML Schema. We represent a sample set of users (by providing instances of the ‘user’ element in XML schema) as given below:

```
<user userID="DrayJ" fullname="Jim Dray"/>
<user userID="GranceT" fullname="Tim Grance"/>
<user userID="VincentH" fullname="Vincent Hu"/>
```

A sample set of encodings for role instances is:

```
<role roleID="BRM" rolename="BranchManager"
                        cardinality="1"/>
<role roleID="CSR" rolename="Customer_Service_Rep"
                       cardinality="3"/>
<role roleID="SDV" rolename="SD_Vault_Officer"
                       cardinality="2"/>
```

A sample set of privileges are given below:

```
<privilege privID="OPEN_ACCT" resource="DepAcct" oper="Open"/>
<privilege privID="DEBIT_ACCT" resource="DepAcct" oper="Debit"/>
<privilege privID="CREDIT_ACCT" resource="DepAcct" oper="Credit"/>
<privilege privID="CLOSE_ACCT" resource="DepAcct" oper="Close"/>
<privilege privID="APPROVE_LOAN" resource="LoanForm" oper="Update"/>
```

A sample set of User-Role relations is:

```
<UserRoleAssignment role='BRM'>
        <user>GranceT</user>
                <user>JansenW</user>
</UserRoleAssignment>
<UserRoleAssignment role='CSR'>
        <user>Sheila</user>
                <user>TomK</user>
</UserRoleAssignment>
```

A sample of set Role-Inheritance relations is:

```
<role_inherit  Inherit_ID="HY1" FromRole="Teller" ToRole="Customer_Service_Rep"/>
<role_inherit  Inherit_ID="HY2" FromRole="Accountant"
ToRole="Accounting_Manager"/>
<role_inherit  Inherit_ID="HY3" FromRole="Customer_Service_Rep"
ToRole="BranchManager"/>
```

A sample set of encodings for Role-Privilege relations is:

```
<RolePrivilegeAssignment role='TEL'>
          <privilege>DEBIT_ACCT</privilege>
      <privilege>CREDIT_ACCT</privilege>
</RolePrivilegeAssignment>
<RolePrivilegeAssignment role='LNO'>
          <privilege>CSR</privilege>
       <privilege>APPROVE_LOAN</privilege>
       <privilege>DEBIT_LOAN</privilege>
       <privilege>CREDIT_LOAN</privilege>
</RolePrivilegeAssignment>
```

A sample set of encoding for specifying conflicting roles is:

```
<ssd_roles  SSD_ID="SSD1" BaseRole="Internal_Auditor" ConflictRole="Accountant"/>
<ssd_roles  SSD_ID="SSD2" BaseRole="Internal_Auditor"
ConflictRole="Accounting_Manager"/>
<ssd_roles  SSD_ID="SSD3" BaseRole="Internal_Auditor"
ConflictRole="BranchManager"/>
```

13.5 Specification of Authorization Policy Constraints & Validation Outcomes

As already stated, authorization policy constraints are domain-specific conditions and involve use of instances of entities and relations of the access control model used by enterprise authorization specification. These constraints are encoded in Schematron language and embedded in Bank-RBAC XML Schema document. In this section we illustrate with several examples the specification of policy constraints using Schematron.

In a Schematron constraint definition, constraints are defined using the following tags:

(a) a ‘rule’ tag to define the context (in terms of the XML schema element) for the constraint and
(b) one or more ‘assert’ tags: Each ‘assert’ tag contains the Boolean expression for the property that each of the instances of the element (named in the context) has to satisfy. Any violation of the property will be flagged off as an error.
(c) one or more ‘report’ tags: Each ‘report’ tag contains the Boolean expression for the property that each of the instances of the element (named in the context) should not satisfy. Any instance where the property is satisfied will be flagged off as an error.
(d) A set of ‘diagnostic’ tags: Each of these provides information on the violating data.
(e) The above tags are also enclosed within a named ‘pattern’ tag.

With the above primer on Schematron, we now illustrate the specification of some important policy constraints that govern the access control requirements for the bank enterprise environment.

Constraint 1: (Role Cardinality Constraint): The cardinality limit (the maximum number of users that can be assigned) specified in the role definition for a role should not be violated in the actual user assignments for that role.

The role definition for the Branch Manager role(roleID = ‘BRM’) in our XML encoded access control data file is as follows:

<role roleID="BRM" rolename="BranchManager" cardinality="1"/>

The reference to the above data through the XML Schema components forms the context. The context therefore is a role instance definition whose roleID attribute is ‘BRM” (for Branch Manager). This context is expressed in Schematron as:

<sch:rule context="Bank_RBAC_Model/role[@roleID='BRM']">

The assertion to be made in this context is that in the corresponding User-Role relation (where the @role=’BRM’), the count of the number of users should not exceed the number specified through the cardinality attribute (@cardinality = 1). The assertion and the corresponding diagnostic messages expressed in Schematron through the assert and diagnostic tags respectively are given below:

<sch:assert test = "./@cardinality >= count(../UserRoleAssignment/user[../@role = 'BRM']) " diagnostics="Cardinality_Exceeded">Cardinality for the role exceeded
</sch:assert>

<sch:diagnostics>
<sch:diagnostic id="Cardinality_Exceeded">The actual number of users assigned is: <sch:value-of select="count(../UserRoleAssignment/user[../@role = 'BRM'])"/> while cardinality limit is: <sch:value-of select="./@cardinality"/>
</sch:diagnostic
</sch:diagnostics>

The actual data in our Bank-Authorization XML Data file is:

<UserRoleAssignment role='BRM'>

<user>GranceT</user>
<user>JansenW</user>
</UserRoleAssignment>

The Schematron validator therefore generated the following error message:
From pattern "Checking for Role Cardinality":
Assertion fails: "Cardinality for the role exceeded" at
/Bank_RBAC_Model[1]/role[1]
<role roleID="BRM" rolename="BranchManager" cardinality="1">...</> The actual number of users assigned is: 2 while cardinality limit is: 1

Constraint 2: (Inheritance Integrity Constraint): Two conflicting roles (specified in the Static Separation of Duty specification) cannot inherit each other. For example the constraint that the role that conflicts with the Internal Auditor role cannot inherit that role is specified as:

```
<sch:pattern name="Checking for Inheritance Integrity">
  <sch:rule context="Bank_RBAC_Model/role_inherit[@FromRole
='Internal_Auditor']">
   <sch:assert test= "not(@ToRole =
(../ssd_roles[@BaseRole='Internal_Auditor']/@ConflictRole))"
diagnostics="INT_INTEG">A conflicting role cannot be inherited.
    </sch:assert>
  <sch:diagnostics>
       <sch:diagnostic id="INT_INTEG">The violating inheritance assignment is made
for the role: <sch:value-of select="./@ToRole"/>
        </sch:diagnostic>
      </sch:diagnostics>
  </sch:rule>
</sch:pattern>
```

The role that violates this inheritance integrity constraint in our bank-enterprise authorization specification is identified and the following diagnostic message is generated by the Schematron Validator tool:

From pattern "Checking for Inheritance Integrity":
Assertion fails: "A conflicting role cannot be inherited." at
/Bank_RBAC_Model[1]/role_inherit[6]
<role_inherit Inherit_ID="HY6" FromRole="Internal_Auditor"
ToRole="BranchManager">...</> The violating inheritance assignment is made for the role: BranchManager

Constraint 3: (Static Separation of Duty Constraint): A user assigned to the Internal Auditor role (@role='AUD') should not be assigned to the Accountant role (@role='ACC') since Internal Auditor and Accountant are conflicting roles.

The context, the assertion and the diagnostic tags used to specify the above constraint is as follows:

```
<sch:rule context="Bank_RBAC_Model/UserRoleAssignment[@role ='AUD']/user">

 <sch:assert test= "not(text() = (../../UserRoleAssignment[@role='ACC']/user/text() ))"
diagnostics="SOD_AUD">There should not a common user in Audit and Accounting roles.
 </sch:assert>
 <sch:diagnostics>
       <sch:diagnostic id="SOD_AUD">The violating assignment  is made for user: <sch:value-
of select="text()"/>
       </sch:diagnostic>
</sch:diagnostics>
  </sch:rule>
```

For our Bank-Authorization XML Data, the Schematron validator generated the following message:

```
From pattern "Checking for Separation of Duty":
    Assertion fails: "There should not a common user in Audit and Accounting roles."  at
  /Bank_RBAC_Model[1]/UserRoleAssignment[6]/user[1]
      <user>...</> The violating assignment is made for user: VincentH
```

Constraint 4: (Constraint specifying Conflicting Users): Users John Wack (user/text() =’JohnW’) and Susan Wack (user/text() = ‘SusanW’) should not be assigned to the same role (whatever be the role) since they have spousal relationship.

The Schematron description of the above constraint is:

```
<sch:pattern name="Checking for Conflicting Users">
  <sch:rule
     context="Bank_RBAC_Model/UserRoleAssignment">
    <sch:assert test="2 > count (user [text () = 'JohnW'])
                          + count(user [text() = 'SusanW'])"
          diagnostics="Wack_Violate">John Wack and
      Susan Wack should not be assigned to the same role
     </sch:assert>
    <sch:diagnostics>
      <sch:diagnostic id="Wack_Violate">The violating
            assignment is for the role: <sch:value-of
            select="@role"/>
       </sch:diagnostic>
      </sch:diagnostics>
  </sch:rule>
</sch:pattern>
```

The diagnostic message prints out the role that JohnW and SusanW are assigned:

```
From pattern "Checking for Conflicting Users":
    Assertion fails: "John Wack and Susan Wack should not be assigned to the same role"  at
      /Bank_RBAC_Model[1]/UserRoleAssignment[4]
      <UserRoleAssignment role="LNO">...</> The violating assignment is for the role: LNO
```

Constraint 5: (Constraint specifying dependent role assignments): Every user assigned to Safe Deposit Vault role (@role=’SDV’) should already be assigned to Customer Service Representative role (@role=’CSD’).

The Schematron syntax for the above constraint is:

```
<sch:pattern name="Checking for Dependent Role
                    Assignments">
   <sch:rule
context="Bank_RBAC_Model/UserRoleAssignment[@role='SDV']/user">
     <sch:assert test="text() = (../../UserRoleAssignment[@role='CSR']/user/text())"
diagnostics="SDV_CSR_Depend">A user assigned to SDV must already be assigned to CSR role
    </sch:assert>
      <sch:diagnostics>
<sch:diagnostic id="SDV_CSR_Depend">The following user is assigned to SDV role but not to CSR role:
<sch:value-of select="text()"/>
     </sch:diagnostic>
    </sch:diagnostics>
   </sch:rule>
 </sch:pattern>
```

The diagnostic message due to our authorization specification not conforming to the above constraint is:

From pattern "Checking for Dependent Role Assignments":
Assertion fails: "A user assigned to SDV must already be assigned to CSR role" at
/Bank_RBAC_Model[1]/UserRoleAssignment[3]/user[2]
<user>...</> The following user is assigned to SDV role but not to CSR role: Gray

<u>Constraint 6:</u> (Limits on Role Assignment for a specific user): The specification of the constraint that a particular user Tom (user/text()= 'TomK') should not be assigned more than two roles is:

```
 <sch:pattern name="Checking for limit on Tom's Assignments">
  <sch:rule context="Bank_RBAC_Model">
    <sch:assert test="3 > count(UserRoleAssignment[user/text()='TomK'])"
diagnostics="Tom_Limit">Tom should be assigned a maximum of 2 roles
     </sch:assert>
   <sch:diagnostics>
      <sch:diagnostic id="Tom_Limit">The actual number of roles assigned to Tom is: <sch:value-of
select="count(UserRoleAssignment[user/text()='TomK'])"/>
       </sch:diagnostic>
     </sch:diagnostics>
  </sch:rule
 </sch:pattern>
```

The diagnostic message generated on our authorization specification is:

From pattern "Checking for limit on Tom's Assignments":
Assertion fails: "Tom should be assigned a maximum of 2 roles" at
/Bank_RBAC_Model[1]
<Bank_RBAC_Model
xsi:noNamespaceSchemaLocation="A:\BankRBAC.xsd">...</> The actual number of roles assigned to Tom is: 3

Constraint 7: (Least Privilege Constraint): The right to open an account as well as to close an account should not be assigned to the same role. The Schematron constraint can be specified not only to verify whether a role with both privileges exists in the Bank-Authorization XML Data but also to print out the violating Role. The constraint specification is:

```
<sch:pattern name="Excess Privilege for a Role">
<sch:rule context="Bank_RBAC_Model/RolePrivilegeAssignment">
    <sch:assert test="2 > (count(privilege[text()='OPEN_ACCT']) +
count(privilege[text()='CLOSE_ACCT']))" diagnostics="Excess_Priv">The Privilege to
Open and Close Accounts should not be assigned to same role
    </sch:assert>
  <sch:diagnostics>
    <sch:diagnostic id="Excess_Priv">The errant role is: <sch:value-of
select="@role"/>
      </sch:diagnostic>
    </sch:diagnostics>
  </sch:rule>
  </sch:pattern>
```

The diagnostic error message generated by Schematron Validator is:

```
From pattern "Excess Privilege for a Role":
  Assertion fails: "The Privilege to Open and Close Accounts should not be assigned to
same role"  at
    /Bank_RBAC_Model[1]/RolePrivilegeAssignment[3]
    <RolePrivilegeAssignment role="CSR">...</> The errant role is: CSR
```

Constraint 8: (Transaction Integrity Constraint): The right to perform certain operations should be assigned to more than role in order to maintain the integrity of the transaction that is facilitated by this operation.

The constraint that the right to perform the operation of Loan Approval should be given to more than one role is specified as:

```
<sch:pattern name="Minimal Roles Roles required for an operation">
  <sch:rule context="Bank_RBAC_Model">
    <sch:assert
test="count(RolePrivilegeAssignment[privilege[text()='APPROVE_LOAN']]) >=2"
diagnostics="Min_Role_Reqmt">A Minimum of two roles is required for loan approval
</sch:assert>
  <sch:diagnostics>
    <sch:diagnostic id="Min_Role_Reqmt">The only role now is: <sch:value-of
select="RolePrivilegeAssignment[privilege[text()='APPROVE_LOAN']]/@role"/>
      </sch:diagnostic>
    </sch:diagnostics>
  </sch:rule>
```

</sch:pattern>

The diagnostic error message generated due to a violation of the policy constraint in our bank-enterprise authorization specification is:

From pattern "Minimal Roles Roles required for an operation":
Assertion fails: "A Minimum of two roles is required for loan approval" at /Bank_RBAC_Model[1]
<Bank_RBAC_Model xsi:noNamespaceSchemaLocation="A:\BankRBAC.xsd">...</>
The only role now is: LNO

13.6 Summary, Benefits and Limitations

We described an authorization policy validation framework. The framework uses XML to encode the enterprise authorization specification and XML Schema to specify the underlying access control model which in our case is RBAC. The policy requirements are encoded in a constraint specification language—Schematron. The XML Schema of the RBAC model is then augmented with these constraint specifications using an annotation feature that is provided as part of the XML Schema language specification. The conformance of the XML-encoded enterprise authorization specification to authorization policies (specified through constraints in Schematron) is verified through a Schematron Validator tool.

The benefits of the authorization policy validation framework described in this chapter are: (a) Authorization specifications pertaining to any native control mechanism and hence any access control model can be captured and expressed since we are using a platform-neutral representation (XML) and (b) Use of high level RBAC model enables specification of several types of authorization policy constraints. The limitation is that, being an out of band approach, we are not verifying the behavior of an access control mechanism in real time, and hence dynamic authorization policies (e.g., a user cannot assume more than one role in a single user-session) cannot be validated.

References

[13.1] R.Chandramouli, “A Policy Validation Framework for Enterprise Authorization Specification” – Proceedings of 19th Annual Computer Security Applications Conference (ACSAC 2003), Las Vegas, NV, USA, Dec 2003.

[13.2] D.Ferraiolo, J.Cugini, and D.R.Kuhn. “Role Based Access Control (RBAC): Features and Motivations” Proc. 11th Annual Computer Security Applications Conference, December 1995.

[13.3] XML Schema Part 0: Primer W3C Recommendation, 2 May 2001 http://www.w3.org/TR/xmlschema-0/

[13.4] Schematron - Pattern-based schema language,

[13.5] http://www.topologi.com/

[13.6] Ferraiolo D.F, Kuhn D.R, Chandramouli R, “Role-Based Access Control”, Artech House, April 2003.

[13.7] R.S. Sandhu, E.J.Coyne, H.L.Feinstein and C.E.Youman. “Role Based Access Control Models” IEEE Computer, vol 29, Num 2, February 1996, p38-47.

[13.8] D.Ferraiolo, R.Sandhu, S.Gavrila, D.R.Kuhn and R.Chandramouli, “Proposed NIST Standard for Role-based Access Control”, ACM Trans. Inf.Syst.Security, Vol 4, Aug 2001, pp 224-274.

[13.9] Document Object Model Technical Reports, http://www/w3.org/DOM/DOMTR

[13.10] R.Chandramouli, “Application of XML Tools for Enterprise-wide RBAC Implementation Tasks”, Proc. Of 5th ACM workshop on Role-based Access Control, July 2000, Berlin, Germany.

[13.11] A.Schaad, “Role-based Access Control system of a European Bank: A Case Study and Discussion”, Proc. Of 6th ACM Symposium on Access Control Models and Technologies (SACMAT 2001), Chantilly, VA, USA.

[13.12] Resource Description Framework (RDF), http://www.w3.org/RDF/ http://www.ascc.net/xml/resource/schematron/schematron.html

[13.13] http://www.xmlspy.com/download.html