Integrating Access Control Policy Specification into the Software Development Process

Qingfeng He and Annie I. Antón
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8207 USA
{qhe2, aianton}@eos.ncsu.edu

Abstract

Access control policies (ACPs) express rules concerning who can access what information, and under what conditions. Traditionally, ACP specification is not an explicit part of the software development process and often isolated from requirements analysis, leaving systems vulnerable to security breaches because policies are specified without ensuring compliance with system requirements. In this paper, we present the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method for deriving ACPs from various sources. This method integrates policy specification into the software development process, ensures consistency across software artifacts, and provides prescriptive guidance for how to specify ACPs. To date, we have validated the method by applying it within the context of four operational systems. This paper reports the results of an empirical study in which we evaluated the usefulness and effectiveness of the method.

1. Introduction

Data security and privacy are important in every software system, but particularly vulnerable in critical infrastructure systems, such as medical, immigration, and financial information systems. Access control (AC) is often used in these kinds of systems to protect data confidentiality and integrity [35]. Two major challenges in an access control system are: (a) defining correct and complete policies to control users' access to the system and its resources, and (b) ensuring the resulting policies comply with the system requirements and high-level security and privacy policies. Proper access control analysis requires one to examine system requirements in tandem with organizational security and privacy policies to specify access control policies (ACPs). However, these analysis activities are often conducted in isolation, leaving sensitive data vulnerable to security breaches.

Researchers are recognizing the need to bridge the gap between requirements analysis and complex ACP specification [6]. Existing RE approaches (e.g., KAOS [11], *i** [36] and the analytical role modeling framework [6]) provide limited support.

Methodological support is needed to guide software engineers as they specify a system's ACPs. To this end, we have developed the Requirements-based Access Control Analysis and Policy Specification (ReCAPS) method [18] to integrate these analysis activities, improve software quality and develop policy- and requirements-compliant systems. In this paper, we present the ReCAPS method for deriving ACPs from various sources, including software requirements specifications (SRS), design documents, database designs, as well as high-level security and privacy policies. The method provides prescriptive guidance for specifying ACPs, improving the quality of software documentation, and ensuring compliance between access policies and system requirements.

The remainder of the paper is structured as follows. Section 2 discusses related work. Section 3 briefly summarizes the ReCAPS method. Section 4 presents the results of an empirical study conducted to evaluate the effectiveness and usefulness of the method. Finally, Section 5 summarizes our plans for future work.

2. Background and Related work

2.1. ACP specification in security

ACPs can be broadly grouped into three main policy categories: Discretionary Access Control (DAC) [23, 17], Mandatory Access Control (MAC) [10, 5], and Role-Based Access Control (RBAC) [31]. Instead of considering ACP specification from a holistic, real-systems perspective as we advocate in this paper, current ACP specification research has a much narrower focus (e.g., uniform or flexible ways to specify ACPs [22], specifying ACPs for XML documents [14], etc.). There are few reported methods and experiences [33] relating ACP specification in operational software systems. In the RBAC literature, researchers are investigating role engineering, the process of defining roles and privileges as well as assigning privileges to roles [7]. Several role-engineering approaches employ requirements engineering (RE) concepts. For example, Fernandez and Hawkins suggest deriving the needed rights for roles from use cases [12]. Neumann and Strembeck propose a scenario-

driven approach for engineering functional roles in RBAC [28]. Role engineering is specific to RBAC, whereas our method is a more general approach for specifying ACPs.

2.2. Elements of ACPs

An access control *policy* is comprised of a set of AC rules. A *rule* can have various modes (e.g., allow/deny/oblige/refrain). This paper focuses on allow and deny rules. *Allow* rules authorize a subject to access a particular object. *Deny* rules explicitly prohibit a subject from accessing a particular object. When a subject requests to perform an action on an object, the corresponding rules are evaluated by the enforcement engine for that request. A typical AC *rule* is expressed as a 3-tuple <*subject*, *object*, *action*>, such that a *subject* can perform some *action* on an *object* [9]. A *subject* is a user or a program agent, or any entity that may access objects. An *object* is a data field, a table, a procedure, an application or any entity to which access is restricted. An *action* is a simple operation (e.g. read or write) or an abstract operation (e.g. deposit or withdraw). In this paper, we extend the typical AC rule 3-tuple to include conditions and obligations.

An ACP may express *conditions* that must be satisfied before an access request can be granted. For example, in healthcare applications, the location from which the access request originates might affect the grant/deny decision [2]. If an access request is from the emergency room, then the request may be granted. We can specify *the location of the request is emergency room* as a condition for the AC rule. *Obligations* [3] are actions that must be fulfilled if a request to access an object is granted. For example, consider: *require affiliates to destroy customer data after service is completed*. Here, "destroy customer data" is an obligation that must be satisfied by affiliates.

In requirements specification, we are concerned with the *actions* for which each actor (*subject*) is responsible, and the conditions under which each action can occur (constraints and pre-*conditions*). Thus, AC elements may be mapped to requirements specification elements, helping to guide analysts as they derive ACPs from requirements.

2.3. Access control analysis in RE

Requirements engineering (RE) researchers are investigating access-related security requirements. Fontaine [15] employs KAOS, a goal-based requirements analysis method [11], to refine security requirements into specific authorization rules and ACPs expressed in Ponder—a language for specifying management and security policies for distributed systems [8]. Fontaine's work is an important step towards requirements-level access control analysis for security policy specification. ACPs come from requirements as well as high-level security and privacy policies. However, Fontaine's approach cannot derive ACPs from security and privacy policies whereas the ReCAPS method can. Additionally, Fontaine's approach does not focus on ensuring compliance between ACPs, requirements and design. In contrast, compliance among these artifacts is an important design principle in ReCAPS.

Liu et al. apply the *i** framework [36], a goal-based method, to support AC analysis by modeling the dependencies among actors, tasks and a system's resources [25]. However this approach assumes the roles and privileges have been previously derived. It lacks guidance on how to identify roles and privileges, from where they originate, or how privileges are assigned to these roles.

Crook et al. propose an analytical role-modeling framework to model ACPs [6]. This approach offers two contributions. First, the framework clarifies the need to model ACPs during requirements analysis. Second, the rationale for deriving roles based on organizational structures is very useful. Job positions in an organization can be mapped to *roles* in RBAC. Organizational and seniority hierarchies can be mapped to RBAC *role hierarchies*. Deriving roles from organizational structures facilitates the user assignment and authorization management processes in access control.

Moffett et al. [26] discuss the relationship between ACPs and requirements engineering, broadly classifying ACPs into three categories: global ACPs that are built into the a system, discretionary ACPs that are basically security requirements, and mandatory ACPs that are essentially mechanisms and should not concern requirements engineering. ReCAPs is consistent with this view; deriving ACPs from requirements and high-level global security and privacy policies.

Lamsweerde employs KAOS [11] to elaborate security requirements by constructing intentional anti-models [24]. This approach generates malicious obstacles set up by attackers to threaten security goals, and provides alternative resolutions to counteract these obstacles. The idea is similar to other approaches that capture security requirements through misuse cases [1]. Misuse cases express the viewpoint of an actor with hostile intent. Security requirements are specified to protect assets in a system from malicious attacks. Thinking from an attacker's standpoint helps elicit security requirements and provide countermeasure resolutions. In ReCAPS, misuse cases are also employed to specify implicit conditions for ACPs.

2.4. Policy specification and software development

Moffett and Sloman [27] discuss the importance of policy hierarchies and define several relationships (e.g., partitioned targets, goal refinement, delegation of responsibility) that may exist between policies in a hierarchy. Policies can be classified according to three expression modes: natural language, declarative or semi-structured languages, and formal languages (see Figure 1). Policies at the top level are stated in natural language; examples include website privacy policies, corporate security policies, security and privacy laws, etc. Policies at the middle level are specified in declarative or semi-structured languages, such as Ponder [8] and XACML [29]. These policies instantiate high-level policies into rules that describe who has permission to access which object in a specific system. Bottom level policies are specified in formal languages such as Authorization Specification Language (ASL) [21] and Alloy [19], which are often used for verification and analysis purpose.

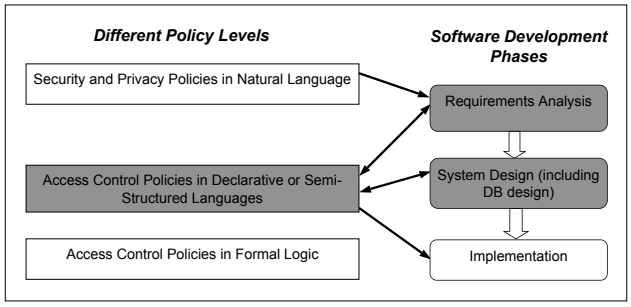

Figure 1. ReCAPS integrates policy specification with requirements analysis and software design

Policy specification is typically isolated from requirements analysis and software design, and often results in high-level policies that fail to comply with system requirements, or mid-level ACPs that fail to comply with system requirements. Brose et al. propose integrating AC design into the software development process by extending UML to specify ACPs for distributed object systems [4]. We concur that ACP specification should be an integral part of software development processes. However, Brose et al.'s approach does not emphasize compliance between different policy levels, requirements and system design, whereas ReCAPS maintains compliance as a key principle.

ReCAPS integrates policy specification with requirements analysis and software design (see Figure 1). The three shaded boxes are the focus of this paper. High-level security and privacy policies should be specified as system requirements. Mid-level policies are instances of high-level policies within a specific system's context. To specify these policies, one must examine system requirements to identify users and their interactions with the system, and system designs (e.g., database design) to identify the data to be protected. Focusing on mid-level policies offers two major advantages. First, they are machine-enforceable, whereas natural languages policies are not. Second, it is relatively simple for system security officers to specify security policies that meet organizational security goals using declarative mid-level languages than using formal languages. The policies discussed herein fall into the former category. ACP specification is an iterative process; although we derive policies from requirements and designs, we also improve the requirements and designs during analysis (see two arrows from mid-level policies to requirements analysis and software design in Figure 1). Finally, mid-level policies can be used by software engineers to implement access control.

3. The ReCAPS Method

The ReCAPS method was developed in two formative case studies involving operational software systems: the Security and Privacy Requirements Analysis Tool (SPRAT) [20] and the Transnational Digital Government (TDG) remote border control system [34]. Examples from these two case studies are employed to elucidate the ReCAPS heuristics in Section 3.2. This section overviews the ReCAPS method, clarifies its underlying assumptions and presents example heuristics.

3.1. Overview of the ReCAPS Method

The overall objective of the ReCAPS method is to produce a comprehensive set of ACP specifications. The inputs are any available source documents (e.g. SRS, design document, database design, E/R diagrams, security and privacy policies, etc.). The SRS and DB design are the minimally required sources. These two source documents are complementary—the SRS justifies the rationale for the ACPs, whereas the DB design details the objects to which any access should be controlled.

The ReCAPS method aids analysts in clarifying and resolving ambiguities, conflicts and redundancies during requirements analysis so that correct and consistent ACPs can be derived and specified for policy enforcement. The mechanisms include a detailed process description and a set of heuristics to guide analysts in deriving and specifying ACPs as well as a software tool to support analysis activities. The process relies on RE techniques, including goal [11] and scenario analysis techniques [30]. The outputs are a set of ACPs and the modified source documents that benefit from refinement, resulting in more complete, correct and less ambiguous documentation.

ReCAPS is based on three assumptions that narrow the scope of application to information systems:

Assumption #1: The system's DB design and SRS exist and are available.

Assumption #2: There are various *data* objects in the system to which access must be restricted (Resource control, such as granting employees access to certain printers but not others, is beyond the scope of this paper.).

Assumption #3: ACPs are specified for information systems, supported by a database containing sensitive data (We have not investigated ACP specifications for security kernels such as file access in operating systems.).

Figure 2 portrays the main activities an analyst undertakes to derive ACPs from source documents. These activities comprise four steps:

***Step #1*:** Develop understanding of problem domain;

***Step #2*:** Scan available source documents to identify AC elements and specify AC rules;

***Step #3*:** Refine AC rules; and

***Step #4*:** Group logically connected AC rules into ACPs.

These steps are accompanied by heuristics. Because the emphasis of this paper is the empirical study, we can only

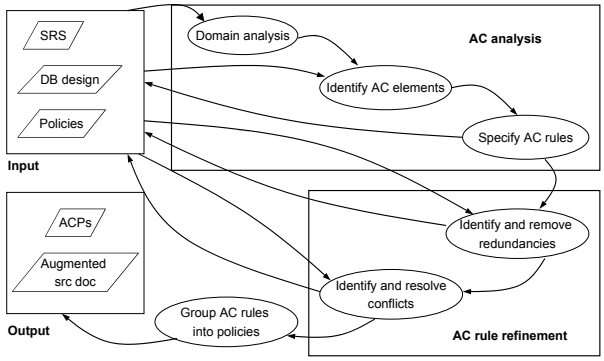

Figure 2. ReCAPS analysis activities

provide a very high level overview of the method and its associated heuristics. A full codification of the method and heuristics is available in [18].

3.2. The ReCAPS Analysis Heuristics

Four kinds of heuristics support the ReCAPS analysis activities: identification (IH), refinement (RH), specification (SH), and grouping heuristics (GH). Currently, ReCAPS provides a total of 32 heuristics (IH: 19, RH: 7, SH: 4, GH: 2).

The identification heuristics help identify the scope of access control and each element of an AC rule: <subject, action, object, condition, obligation>. For example, the following heuristic helps identify objects:

$IH_{object}3$: To identify the objects that must be included in the database, every object identified in the SRS should be mapped to an object (e.g., a table, a column, a row or a cell) in the database design.

Because requirements and design specifications are often treated as separate phases conducted by different persons in traditional software development processes, inconsistencies inevitably exist across these artifacts. Heuristic $IH_{object}3$ helps analysts clearly define what the objects are in the DB (e.g., a table, a column, a row or a cell in a table) early on and helps ensure the requirements and database design are consistent with one another.

Consider this border control system requirement:

TDG 2.3.1: The system shall allow border immigration agents to determine if the traveler is on the "watch list".

This requirement was annotated in the SRS with questions and answers from the requirements engineers and stakeholders. These annotations clarify exactly what is on the watch list as follows:

TDG 2.3.1 Annotation: ***Question C:*** What data is contained on the "watch list"?

Belize Answer: Name, Date of Birth, Nationality, Reason for being on the list, Action to be taken.

DR Answer: Basically name, gender, citizenship, watch list inclusion explanation and actions to be taken in case of positive identification.

Three items mentioned in these annotations were missing in the DB design: gender, the reasons for a person's name being on the watch list, and actions to be taken if a person whose name appears on the watch list is encountered at a border station. These items were added subsequently added to the database. Heuristic $IH_{object}3$ also helps analysts correct database designs early on, preventing possible costly changes that are typically not identified until later in the software lifecycle.

In ReCAPS, explicit conditions are clearly defined as constraints or pre-conditions in requirements specifications. ReCAPS defines six kinds of constraints that are useful for identifying conditions: authentication constraints, contextual constraints (temporal, location, relationship, affiliation, attribute, and state), usage constraints, database constraints, security constraints, and privacy constraints (purpose, recipient, and consent). For example, the following heuristic helps identify security constraints.

$IH_{cond}7$: (Security constraints) Use general security principles to construct misuse cases that a user may use to exploit the capabilities for hostile intent. Corresponding security constraints should thus be specified in the resulting AC rule.

Security constraints specify restrictions that are based on general security principles such as least privileges and separation of duties. These security principles are important to prevent fraud and errors. Consider this requirement:

SPRAT FR-PM-3: The system shall support multi-user analysis results comparison.

This requirement needs to be clarified. The requirement actually means *Analysts* can classify goals according to some predefined taxonomy and *Project Managers* can view these classification results. *Project Manager* and *Analysts* are not mutually exclusive roles, which means a user can assume both roles at the same time. To identify security constraints, we ask "can a *Project Manager* exploit the capability for his/her own good?" and construct the misuse case shown in Figure 3. Note that the misuse case diagram does not employ the standard use case diagram. In this misuse case, user A assumes both roles: *Project Manager* and *Analyst*, whereas user B assumes only the *Analyst* role. The attack pattern is shown as Step (1) – (3) in Figure 3.

Step 1: B classifies goals,

Step 2: A views B's classification results

Step 3: A classifies goals.

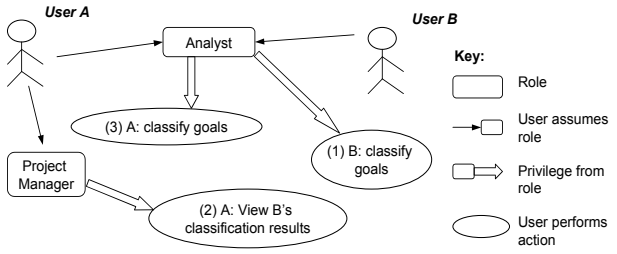

Figure 3. A misuse case for *Project Manager*

This is not desirable because A would be biased if he/she saw B's classification results before he/she has classified the goals. Misuse case analysis allowed us to specify a deny rule for *Project Manager* to view goal classification results as shown in Table 1.

Table 1. Example AC rule derived from Figure 3

Mode	Deny
Subject	Role (Project Manager)
Action	Read
Object	goals.taxonomy
Condition	Role (user, Analyst) = TRUE AND user.scheduledToClassify = TRUE AND user.classifyingFinished = FALSE
Obligation	NULL
Source	FR-PM-3

3.3. Validation Case Studies

We have applied the ReCAPS method within the context of four operational systems. The first two case studies were formative, offering early and preliminary

validation during to their central role in shaping the method [18]. The second two were summative case studies: the Surry Arts Council (SAC) Web enhancement project and the College of Management Event Registration System (ERS) at North Carolina State University (NCSU), both of which are Web-based applications. Results from these studies suggest that the ReCAPS process and heuristics are useful and efficient in specifying requirement-compliant ACPs for data-intensive information systems. These four case studies enabled us to empirically evaluate the effectiveness and usefulness of the ReCAPS method when employed by others.

4. Empirical Evaluation

To evaluate the effectiveness and usefulness of ReCAPS, we conducted an empirical study in a graduate-level software engineering class at NCSU. Because of the lack of prescriptive guidance for ACP specification, we were unable to compare ReCAPS with other existing methods. The closest is Fontain's mapping approach from KAOS specifications to Ponder policies [15]. However, it is difficult to compare ReCAPS with Fontain's approach in an educational environment. Fontain's approach starts from KAOS specifications expressed in a formal language. It requires significant training for a group of undergraduate and graduate students to understand and use KAOS specifications, making it unsuitable for a homework assignment. Moreover, the experimenter would have to manipulate the source documents and produce KAOS specifications from these source documents a-priori for the group that applies Fontain's approach. This comparison is unfair because the two groups would start from different inputs to derive ACPs. Thus, our empirical study required careful design as we now discuss. Our main hypothesis was that ReCAPS allows analysts to specify better quality ACPs than does the lack of a method (which is the current state of the art).

4.1. Experimental Method

This experiment compares the use of ReCAPS to specify ACPs for information systems with a control condition in which no method was stipulated. The subject population was a group of undergraduate and graduate students enrolled in the Spring 2005 NCSU graduate software engineering course (CSC 510). Students were invited to voluntarily participate in an experiment for which they would be compensated with extra credit.

Students participating in the experiment possessed varied backgrounds: 3 PhD students; 28 master's students, and 17 undergraduate seniors. Their knowledge in software engineering, databases, and security differed significantly. To minimize noise that would have occurred with unbalanced groups, a survey was conducted beforehand to collect information from each subject concerning their background. The course instructors blinded any identifying information (e.g. students' names) from us to ensure that we assigned students to groups in an unbiased manner. All 48 students were then assigned to the two groups in a balanced fashion. Students were first grouped according to their expertise and class level. Allocation was random within a group with similar expertise and the same class level. For example, when an undergraduate student with limited security knowledge was assigned to one group, another undergraduate student with similar level of security knowledge was assigned to the other group.

Of the 48 students enrolled in the course, 28 participated. Of the 28 participants, 26 submitted valid responses. Assignments were deemed valid if the participant completed the entire assignment, including the time/effort form and evaluation questions. Note that among those 26 students who participated and submitted valid results, the ReCAPS group was less confident about their security and software engineering skills than the control group; whereas the control group was less confident about their database background than the ReCAPS group.

The project used in the experiment was a simplified version of the Surry Arts Council (SAC) Web enhancement project [18]. Two documents served as the sources for this study: an SRS document and a database schema design. The simplified SRS contains ten requirements (in contrast to 15 requirements in the full SRS), and the DB schema design contains five tables (as in the original DB design).

Both groups were given identical source documents. What differed in the materials were the instructions. The ReCAPS group was given an assignment description that summarized the method, including the main activities and several heuristics. The control group was provided a different assignment description that only summarized necessary background information for how to complete the study (much of this was identical to that provided to the ReCAPS group, but all ReCAPS context was removed), and what correct results look like (e.g., what is access control, what an access control policy is comprised of).

Although no specific method for how to specify ACPs was provided to the control group, both groups were given the exact same criteria for a good set of ACPs (see Section 4.2) with example rules and policies. The instructions made it clear that these criteria would be used to evaluate their results. The ReCAPS instructions explained how to ensure these criteria are met, whereas the control group knew the criteria but only received guidance about how to achieve the criteria in the form of correct AC rule and policy examples.

An initial pilot study was run in the Fall 2004 NCSU graduate software engineering class (CSC 510). Pilot studies are necessary prerequisites for the design of a sound empirical study. This pilot study enabled us to revise the materials to ensure that students in the subsequent empirical study fully understood what constituted a "valid response." Also, the access control rules produced during this initial study were given to an independent security expert to evaluate the quality of the rules specified by the pilot study students. We included these evaluation criteria in the final empirical study to ensure that all participants in both groups had a common understanding how their results would be evaluated (see Section 4.2 for these criteria).

The main task for the subjects was to produce a set of ACPs derived from two source documents. Additionally,

subjects were required to improve both documents during ACP analysis and specification; for example, if they identified an inconsistency or missing requirement/data element, they were to document this in the respective document(s). All results were documented on the provided worksheets. Subjects were also required to document the amount of time they spent. However, subjects were assured that the amount of time spent on the assignment would in no way affect their grades. Upon completing the study, subjects had to answer several qualitative questions concerning their experience during the assignment.

4.2. Measurement

We employed the Mann-Whitney U Test [32] to conduct statistical analysis on the empirical study results. This test is a non-parametric method for testing the significant differences between two independent groups with non-normal data and small sample size.

Three main aspects of the experiment may be compared across the two groups of subjects: the quality of resulting ACPs, the improvements to the two source documents, and the time effort. The following eight criteria were used to evaluate the quality of resulting ACPs. These criteria were determined together with a security expert after the pilot study, as mentioned in Section 4.1.

(1) All possible access control rules are specified.
(2) Each rule is within the scope of access control.
(3) Each action is a database operation and each object is an object in the DB.
(4) The conditions for each rule are correctly specified and as completely as possible.
(5) Each rule is traceable to the sources from which it was derived.
(6) No two rules are redundant.
(7) No two rules conflict with one another.
(8) Logically connected rules are grouped together.

These eight criteria were used to evaluate the resulting ACPs. The improvements to both source documents were measured using the number of inconsistencies identified between the SRS and the database design, the number of ambiguities and inconsistencies identified within the requirements specifications, and the number of problems identified within the database design. All subjects were aware that their results would be evaluated using the same quality criteria. This ensured that all subjects had a common understanding about the study's objective and prevented the experimental results from being skewed.

We followed the following process to quantitatively evaluate the quality of AC rules derived by subjects. First, we counted all the subject-derived rules and compared these rules with a rule set that was derived from the same sources and specified by experts. For each rule in the experts' rule set, we noted whether there was a semantically equivalent rule in the subject's rule set. If yes, we counted the subject-identified rule, otherwise we counted it as incorrect. The form of rules specified by the subjects did not have to be identical to those specified by the experts. In this way, we were able to create two sets of rules for each subject: a set of rules *A* that are in the experts' rule set and a set of rules *B* that are false positive (either outside the scope of access control or cannot be derived from sources, see Figure 4).

For each rule in rule set *A*, we analyzed: whether each action is a database operation and each object is an object in the DB, whether the conditions are correct and as completely specified as possible, and whether the rule is traceable to the sources from which it was derived. For the entire rule set *A*, we analyzed whether there were any redundant or conflicting rules and whether logically connected rules were grouped together. A set of rules is logically connected if the rules have the same subject, action or object. For each rule in rule set *B*, we documented whether it was "outside the scope of access control" or "cannot be derived from sources".

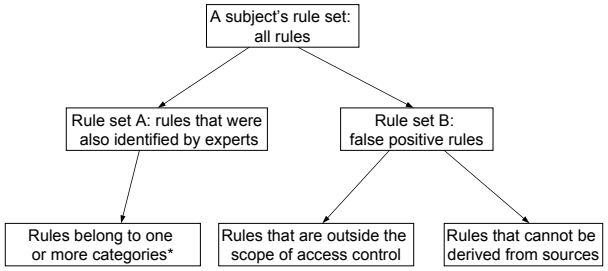

Figure 4. Evaluation of a subject's rule set

*: E.g., ambiguous actions/objects, incomplete conditions, etc.

We used the number of AC rules identified by each subject to measure criteria 1—4 and 6—7. However, traceability (Criterion 5) is evaluated as a binary measure: a subject is either able to trace all rules to the sources or is not. We did not find a case in which a subject was able to trace some rules to their sources but not others. In all cases, all rules were traced or none were. With respect to grouping (Criterion 8), the rules were graded on a scale of 0—10 based on the grouping in each subject's rule set. Criterion 8 was scored as 10 if all rules were grouped into policies, otherwise only partial credit was given, depending on based on how many rules were correctly grouped.

4.3. Results

4.3.1. Quality of Access Control Policies

Table 2 compares the performance of the ReCAPS and Control groups with respect to the eight evaluation criteria. The evaluation criteria are listed in the first column and the second column specifies whether the ReCAPS group outperformed the Control group, based upon whether the Mann-Whitney U test revealed the results as statistically significant. As shown in Table 2 and Figure 5, the ReCAPS group identified more AC rules that were also identified by the experts than the control group (Criterion 1). The average number of rules identified by the ReCAPS group that were also in the experts' rule set is 22.29, compared with 15.17 rules identified by the control group. The results of this comparison are statistically significant (Mann Whitney $U = 152.5$, $p < 0.001$, two-tailed test). This data shows the ReCAPS group provided better AC rule and policy coverage than the control group.

If we subtract the number of rules identified by the experts from the total number of rules identified by each subject, we obtain the number of false positive rules. The ReCAPS group identified fewer false positive rules than the control group (see Figure 6). The average number of false positive rules identified by the ReCAPS group is 5.79, compared with 8.25 false positive rules identified by the control group. The results of the comparison are statistically significant (Mann Whitney U = 122.5, $p < 0.05$, two-tailed test). This data further demonstrates that the ReCAPS not only ensures better coverage; it also reduces false positive access control rules.

Table 2. Summary of empirical study results

Evaluation Criteria	ReCAPS > Control
(1) All possible access control rules are specified.	Significant, $p < 0.001$
(2) Each rule is within the scope of access control.	Not significant
(3) Each action is a database operation and each object is an object in the DB.	Significant, $p < 0.001$
(4) The conditions for each rule are correctly specified and as completely as possible.	Not significant
(5) Each rule is traceable to the sources from which it was derived.	Significant, $p < 0.001$
(6) No two rules are redundant.	Not significant
(7) No two rules conflict with one another.	Not significant
(8) Logically connected rules are grouped together.	Significant, $p < 0.001$

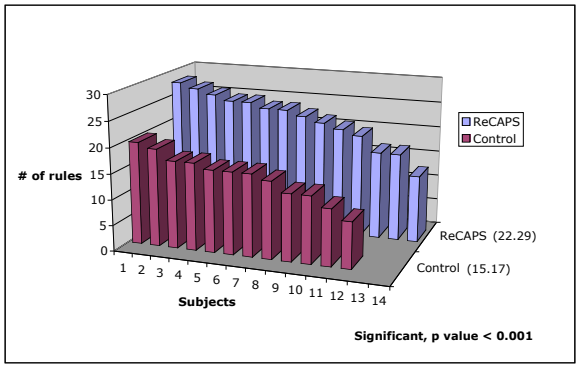

Figure 5. Number of rules identified by each subject that were also identified by the experts: the ReCAPS group outperformed the control group by identifying more rules that were also identified by the experts.

Subjects were asked to specify database-level ACPs, in which every action is a DB operation and every object is an object in the DB. This helps minimize the number of ambiguous actions and objects identified. The ReCAPS group outperformed the control group by identifying fewer rules with ambiguous actions/objects (see Figure 7). The average number of rules with ambiguous actions/objects identified by the ReCAPS group is 2.71, compared with 14.58 rules identified by the control group. The results of the comparison are statistically significant (Mann Whitney U = 160.0, $p < 0.001$, two-tailed test).

The ReCAPS group yielded more rules with correctly specified elements than the control group. Here, *correctly* means that each action is a DB action, each object is an object in the DB, and any conditions are fully and correctly

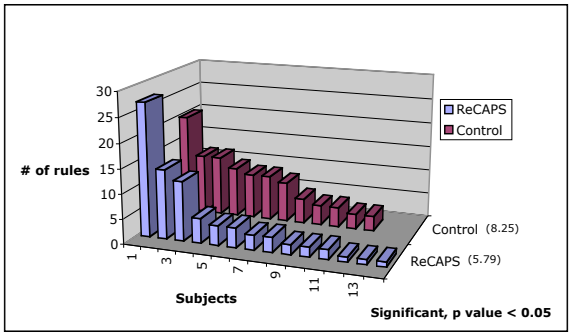

Figure 6. Number of false positive rules identified by each subject: the ReCAPS group outperformed the control group by identifying fewer number of false positive rules.

specified. The average number of rules with every element correctly specified by individuals in the ReCAPS group is 9.43, compared with 1.25 rules specified by individuals in the control group. The results of this comparison are statistically significant (Mann Whitney U = 153.5, $p < 0.001$, two-tailed test).

As previously mentioned, traceability was evaluated in a binary fashion in this study. The individuals in the ReCAPS group consistently out performed the individuals in the control group with regard to maintaining traceability. The results of the comparison are statistically significant (Mann Whitney U = 154.0, $p < 0.001$, two-tailed test).

In short, the ReCAPS group outperformed the control group in all eight evaluation criteria. However, the Mann-Whitney U test results are statistically significant in four of the eight evaluation criteria (1, 3, 5, 8). In the remaining criteria (2, 4, 6, 7), the results are positive, but anecdotal at best because they are not statistically significant.

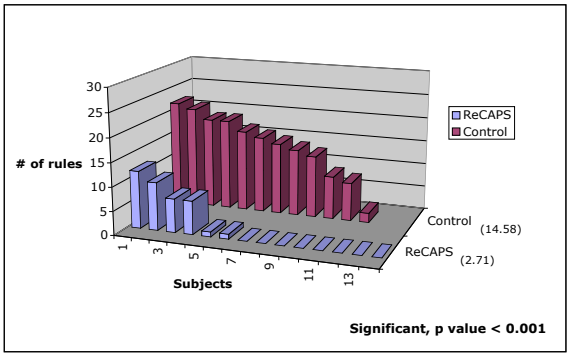

Figure 7. Number of rules with ambiguous actions/objects identified by each subject: the ReCAPS group outperformed the control group by identifying fewer rules with ambiguous actions/objects.

4.3.2. Improvements to Source Documents

The empirical study provided only anecdotal evidence that the ReCAPS group outperformed the control group by identifying more inconsistencies between the SRS and the database design, more ambiguities/inconsistencies within the requirements specifications, and more problems in the database design. Although not statistically significant, the data suggests that improvement to source documents is a side-benefit of the approach, not a main contribution.

4.3.3. Time Effort

The average amount of time spent on the assignment by the ReCAPS subjects was 4.53 hours, compared with 3.79 hours by individuals in the control group. The results of this comparison are statistically significant (Mann Whitney $U = 124.0$, $p < 0.05$, two-tailed test).

One might challenge the validity of the above data by suggesting that the ReCAPS group outperformed the control group simply because they spent more time than the control group. To address this, we performed a correlation analysis between the number of rules that were in the experts' rule set identified by the subjects and the amount of time spent on the assignment by the subjects. The result shows there is no strong correlation between these two factors (see Figure 8). The variance R^2 (0.0326) would lead to a correlation coefficient *r* close to 0.18. The correlation result is not statistically significant either, with *p* value greater than 0.05.

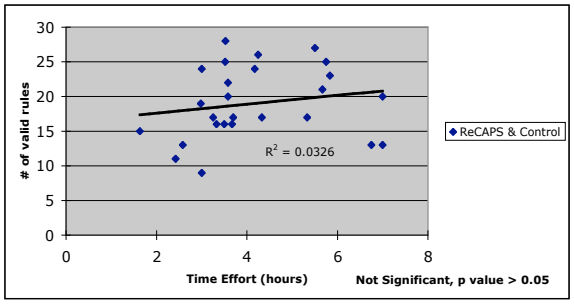

Figure 8. Correlation analysis shows there is no correlation between the time effort and the results in all subjects

We conducted the same correlation analysis within the ReCAPS group and the control group. The variance R^2 for the ReCAPS group is 0.0939 (correlation coefficient $r = 0.31$) and the variance R^2 for the control group is 0.1221 (correlation coefficient $r = 0.35$). Neither of the correlation analysis results is statistically significant, with *p* value greater than 0.05. Thus, it is unlikely that time/effort alone can account for the results in this study.

4.4. Summary

As previously stated, this empirical study sought to evaluate the effectiveness and usefulness of the method in comparison to other approaches (none of which exists) and whether analysts who are unfamiliar with the method can effectively employ it with reasonable training. The study demonstrated the feasibility of employing the ReCAPS method as a reasonable policy specification method. The results are supported by the similar specification of ACPs for the SAC Web enhancement project across the ReCAPS and control groups. Four of the evaluation criteria for high-quality access control rules and policies were shown to yield statistically significant results in which the ReCAPS group outperformed the Control group.

Because we were limited in the size and complexity of the assignment that we could design for an optional extra credit assignment, only a subset of the ReCAPS heuristics were evaluated in this empirical study. The ReCAPS group was provided with 18 heuristics ($IH_{scope}1\text{-}2$, $IH_{object}1\text{-}3$, $IH_{subject/action}1$, $IH_{subject/action}3$, $SH_{DLP}1\text{-}2$, $SH_{DLP}4$, $IH_{cond}1$, $IH_{cond}3$, $IH_{cond}6$, $IH_{cond}8$, $RH_{redundancy}1\text{-}2$, $RH_{conflict}1\text{-}2$) out of total 32 heuristics. These 18 heuristics were selected for this empirical study by design. First, because the study was being conducted within the confines of a homework assignment, the selected heuristics could not require excessive prerequisite knowledge nor could so many heuristics be included without risking lack of participation due to the assignment's length and complexity. Second, the set of heuristics needed to stand on their own and be representative of the ReCAPS' mission. Third, the selected heuristics needed to be broadly applicable in the assignment. These factors helped ensure that the set of heuristics used in this empirical study supported the identification, specification and refinement of AC rules.

The empirical study results and the feedback received from the ReCAPS group suggests that this subset of heuristics is helpful in guiding analysts as they derive AC rules and specify ACPs. One student made the following statement in addressing whether the heuristics helped him/her perform the analysis efficiently:

"I believe it was very efficient. Several assumptions I would have overlooked were analyzed more meticulously because I was forced to state them."

Another student commented on the heuristics:

"The heuristics, just as the breaking of the access control rule into four parts, helped me to wrap my head around each section. This made the assignment manageable. Rather than trying to do everything at once in a non-logical order, the heuristics force me to do each step in order and sometimes in parallel."

These anecdotal statements provide further insights into how ReCAPS helps analysts. The remaining heuristics will need to be evaluated in follow up studies due to limitations on the scale of this study. Thus, it was not possible to evaluate the heuristics as a complete set (e.g., whether they are complete and sufficiently comprehensive).

4.5. Discussion

These results are very encouraging, but require careful interpretation. Because there are no existing documented ACP specification methods that provide prescriptive guidance comparable to ReCAPS, any evaluation necessarily involves comparing the performance of subjects applying ReCAPS to those using a different type of method or using no method at all. Examples of non-comparable methods include best industry practices. This evaluation strategy was rejected because we were unable to obtain a written version of best industry practices.

The alternative used in the current empirical study—comparing individuals' performance in the ReCAPS group to individuals' performance in a control group—is open to the objection that *any* method may help people, at least initially, because of the so-called Hawthorne Effect [13]. By virtue of being the focus of an investigation and by manipulating their behavior in an obvious way (here, training in ReCAPS), subjects are more attentive to the task at hand and perform better than they would have otherwise. Such a difference stems not from the treatment (ReCAPS) but from the fact there is a treatment.

There are two responses to this concern. First, it is still important to confirm that the ReCAPS group significantly outperformed the control group. For the Hawthorne Effect to be a potential threat to validity, there has to be a potentially valid effect to threaten. The statistical analysis reveals that ReCAPS improved performance at the $p<0.001$ level. Research in process interventions is notoriously vulnerable to individual differences, task variables, experimental demand characteristics and seemingly random properties of the experimental materials, e.g., different worksheets used for each group. Plausibly justified software engineering techniques and methods often fail to show any advantage when subjected to experimental test. In many cases, this may be because the techniques do not help at all; but in view of the factors just mentioned, it is important to replicate such studies before accepting such results at face value. Thus, any significant result, such as the one obtained in the current study, is clear prima facie evidence that the treatment effect yields a genuine benefit.

The second response concerns the mechanism through which the Hawthorne Effect is supposed to work. It could be that being subject to a visible treatment caused subjects to pay more attention to the task or take it more seriously, presumably because the attention they were receiving by being subjects in an experiment was reinforcing and they wanted to do well. However, since there was no significant difference between the time spent on the task by the subjects in the two groups, we can safely conclude that the results are not merely due to one group focusing on the problem for longer. It is of course possible that the ReCAPS subjects paid more careful attention and used the time they had more effectively and would have done so with any instructions at all. However, this residual effect is unlikely for the following reason. The Hawthorne Effect is most likely to arise in a repeated measures design where the subjects are aware of the nature of changing treatments. (The original interventions at the Hawthorne Plant, from which the effect gets its name, are a classic example of subjects being aware that they were being subjected to one treatment after another.) In such a case, a control condition is manifestly less a treatment than an experimental condition. However, in an independent group design, such as the one used in the current study, the subjects in the experimental condition had no experience of the control condition with which to compare it. Subjectively, therefore, they were as likely to improve their performance as the ReCAPS subjects. We conclude that the difference between the ReCAPS and control groups can be attributed to the ReCAPS process and heuristics and not to any methodological artifact. Such a conclusion requires future replication before it can be accepted as definitive, but it is better supported currently than alternative explanations.

5. Discussion and Plans for Future Work

In this paper, we briefly summarized the ReCAPS method and reported our experience in evaluating its effectiveness and usefulness through an empirical study. ReCAPS supports AC analysis and ACP specification by providing methodological support and a rich set of heuristics to help software and security engineers specify requirements-, design- and policy-compliant ACPs. ReCAPS offers three main advantages that are not currently available: it integrates ACP specification in the software development process, supports traceability to help ensure compliance between policies and requirements (especially with the aid of our newly developed software tool), and offers prescriptive guidance for ACP specification.

Although ReCAPS is essentially an analysis method supported by a set of heuristics and a tool (that was completed after the empirical study reported herein), incorporating ACP specification as an explicit part of the software development process is a significant contribution of this work. By integrating ACP specification with requirements analysis and software design, ReCAPS provides a basic framework for ensuring compliance between different levels of policies, system requirements and software design. The impact of this compliance is important. Companies and organizations are plagued by the degree of confidence they must have when claiming that their information systems are enforcing security/privacy laws and global policies. This challenge also plagues law enforcement agencies; currently they lack technology to help them measure an organization's ability to enforce laws. This is critical given that organizations are being held accountable for their practices. The ReCAPS approach is a promising step in the right direction. By deriving ACPs from system requirements and high-level security / privacy policies, we are able to focus on what is required to achieve and enforce compliance early on while improving the overall correctness and quality of a project's software artifacts. The ReCAPS method helps ensure that a software system actually enforces high-level security / privacy laws and policies. Additionally, by establishing traceability links between high-level policies, system requirements, and ACPs, ReCAPS (especially with the aid of the SPRAT) helps ensure that any changes in the high-level policies can be easily traced to the corresponding software artifacts (e.g., requirements specifications, ACPs), where appropriate changes can be made to properly support evolution.

The ReCAPS method does have its limitations: to date, it has only been applied within the context of systems for which a requirements specification document and a database design were readily available—it has yet to be validated within the context of a newly envisioned system; formal analysis of the resulting ACPs for completeness and consistency is not yet available in the tool (SPRAT); and the method does not yet provide enough support for defining roles for RBAC systems. These limitations motivate our plans for future work. Specifically, we plan to extend ReCAPS so that it can be applied during initial requirements analysis and to drive database design in newly envisioned systems. We are also investigating role engineering [16] within ReCAPS. These extensions will broaden the scope of the method and tool, making it more useful and accessible. Finally, we plan to integrate the SPRAT with other software tools and further validate the ReCAPS method and the tool in industrial settings.

Acknowledgements

This work was supported by NSF ITR Grant #0325269. We thank Dr. Thomas Honeycutt and David Wright for allowing us to conduct the empirical study in their class; the NCSU students who participated in the pilot and empirical studies; as well as Colin Potts, Calvin Powers, Jonathan Moffett, Ting Yu and the NCSU ThePrivacyPlace.Org reading group for their helpful comments to an early version of the paper.

References

[1] I. Alexander. Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, Vol. 20 (1), pp. 58-66, 2003.

[2] K. Beznosov. Requirements for Access Control: US Healthcare Domain, *3rd ACM Workshop on Role-Based Access Control*, pp. 43, 1998.

[3] C. Bettini, S. Jajodia, S. Wang, D. Wijesekera, Provisions and obligations in policy rule management and security applications, *28th Int'l Conf. on Very Large Data Bases (VLDB'02)*, pp. 502-513, 2002.

[4] G. Brose, M. Koch, and K.-P. Löhr. Integrating Access Control Design into the Software Development Process. *6th Int'l Conf. on Integrated Design and Process Technology (IDPT)*, 2002.

[5] D.E. Bell and L.J. LaPadula. Secure computer systems: Mathematical foundations, *Technical Report MTR-2547*, Vol. 1, MITRE Corporation, 1973.

[6] R. Crook, D. Ince, and B. Nuseibeh. Modelling Access Policies Using Roles in Requirements Engineering, *Information and Software Technology*, 45(14), pp. 979-991, Elsevier, 2003.

[7] E.J. Coyne. Role Engineering, *1st ACM Workshop on Role-Based Access Control (RBAC'96)*, pp. 15-16, 1996.

[8] N.C. Damianou. A Policy Framework for Management of Distributed Systems, *PhD Thesis*, Imperial College, London, 2002.

[9] D. E. Denning and P. J. Denning, *Cryptography and Data Security*, Addison-Wesley, 1982.

[10] D.E. Denning. A Lattice Model of Secure Information Flow, *Comm. of the ACM*, 19 (5), pp. 236-243, 1976.

[11] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20: 3-50, 1993.

[12] E.B. Fernandez and J.C. Hawkins. Determining Role Rights from Use Cases, *2nd ACM Workshop on Role-Based Access Control*, pp. 121-125, 1997.

[13] R.H. Franke and J.D. Kaul. The Hawthorne experiments: First statistical interpretation. *American Sociological Review*, 43, pp. 623-643, 1978.

[14] I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath, *9th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pp. 61-69, 2004.

[15] P.-J. Fontaine. Goal-Oriented Elaboration of Security Requirements, *Project Dissertation*, Université Catholique de Louvain, Belgium, 2001.

[16] Q. He and A.I. Antón. A Framework for Modeling Privacy Requirements in Role Engineering, *9th Int'l Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03)*, pp. 137-146, 2003.

[17] M.H. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems, *Communications of the ACM*, 19 (8), pp. 461-471, 1976.

[18] Q. He. Requirements-Based Access Control Analysis and Policy Specification. *PhD Dissertation*, North Carolina State University, Raleigh, NC, 2005.

[19] D. Jackson. Alloy: A Lightweight Object Modelling Notation. *ACM Transactions on Software engineering and Methodology*, Vol. 11 (2), pp. 256-290, 2002.

[20] N. Jain, A.I. Antón, W.H. Stufflebeam, and Q. He. Security and Privacy Requirements Analysis Tool (SPRAT) Software Requirements Specification Version 2.00, *NCSU CS Technical Report TR-2004-7*, April 9, 2004.

[21] S. Jajodia, P. Samarati, and V.S. Subrahmanian. A Logical Language for Expressing Authorizations. *1997 IEEE Symposium on Security and Privacy*, pp. 31-42, 1997.

[22] S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian. Flexible support for multiple access control policies, *ACM Transactions on Database Systems (TODS)*, 26(2), pp. 214-260, 2001.

[23] B.W. Lampson. Protection, *5th Princeton Symposium on Information Science and Systems*, pp. 437-443, 1971.

[24] A. van Lamsweerde. Elaborating Security Requirements by Construction of Intentional Anti-Models. *26th Int'l Conf. on Software Engineering (ICSE'04)*, pp. 148-157, 2004.

[25] L. Liu, E. Yu and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting, *11th Int'l Requirements Eng. Conf. (RE'03)*, pp. 151-161, 2003.

[26] J.D. Moffett, C.B. Haley, and B. Nuseibeh. Core Security Requirements Artefacts. *Technical Report Number 2004/23*, Department of Computing, Open University, UK, 2004.

[27] J.D. Moffett and M.S. Sloman. Policy Hierarchies for Distributed Systems Management. *IEEE Journal on Selected Areas in Communications*, Vol. 11 (9), pp. 1404-1414, 1993.

[28] G. Neumann and M. Strembeck. A Scenario-driven Role Engineering Process for Functional RBAC Roles, *7th ACM Symp. on Access Control Models and Technologies (SACMAT'02)*, pp. 33-42, 2002.

[29] *OASIS eXtensible Access Control Markup Language (XACML)*. Version 2.0, February 1, 2005. http://www.oasis-open.org/committees/xacml

[30] C. Potts, K. Takahashi and A.I. Antón. Inquiry-Based Requirements Analysis, *IEEE Software*, 11(2), pp. 21-32, 1994.

[31] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman. Role-Based Access Control Models, *IEEE Computer*, 29(2), pp. 38–47, 1996.

[32] S. Siegel. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill Kogakusha Ltd., 1973.

[33] A. Schaad, J. Moffett, J. Jacob. The Role-Based Access Control System of a European Bank: A Case Study and Discussion, *6th ACM Symp. on Access Control Models & Technologies (SACMAT'01)*, pp. 3-9, 2001.

[34] S. Su, et al. Transnational Information Sharing, Event Notification, Rule Enforcement and Process Coordination. *Int'l Journal of Electronic Government Research (IJEGR)*, Vol. 1 (2), pp. 1-26, 2005.

[35] P. Samarati, S. De Capitani di Vimercati. Access Control: Policies, Models, and Mechanisms, *IFIP WG 1.7 Int'l School on Foundations of Security Analysis and Design (FOSAD 2000)*, LNCS 2171, pp. 137-196, 2001.

[36] E. Yu. Modeling Organizations for Information Systems Requirements Engineering, *1st IEEE Int'l Symp. on Requirements Engineering*, pp. 34-41, 1993.