Enriching Traceability with Context for Adaptive Information Security in the Cloud

Armstrong Nhlabatsi and Thein Tun and Niamul Khan and Yijun Yu and Arosha Bandara and Khaled Khan and Bashar Nuseibeh

29 April, 2014

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Enriching Traceability with Context for Adaptive Information Security in the Cloud

Armstrong Nhlabatsi1, Thein Tun2, Niamul Khan1, Yijun Yu2, Arosha Bandara2, Khaled Khan1, Bashar Nuseibeh2,3

1Department of Computer Science and Engineering, Qatar University, Doha, Qatar
2Department of Computing and Communications, The Open University, Milton Keynes, United Kingdom
3Department of Computing, Queen Mary University of London

Abstract – Cloud applications enjoy a diverse community of users to store and process a variety of data in different conditions in their execution environment. We refer to the attributes that determine these conditions as context. Therefore these applications have a variety of security requirements, the satisfaction of which depends on the application adapting on the users’ context. We call such adaptation capability Adaptive Information Security. The paper argues that one of the key prerequisites for adaptive information security in the cloud is the use of traceability as a means to reasoning the relationship between security requirements and the policies that satisfy those requirements. However, current approaches to traceability do not provide support for taking into account contextual attributes. This makes it challenging to reason about satisfaction of the security requirement at runtime. We propose an approach to traceability that addresses this challenge by making context explicit. Our approach uses entailment relationships to capture and enrich traceability links with context. We use these links to diagnose the violation of security requirements. We applied our approach to an open-source cloud application (ownCloud) which we re-engineered for adaptive access control.

Keywords – Traceability, Context, Adaptive Information Security;

I. ADAPTIVE INFORMATION SECURITY IN THE CLOUD

Information security is about protecting valuable information assets from intentional harm [1]. With the popularity of mobile and ubiquitous uses of computing infrastructures, such as the cloud [2][3], both technical and social contexts in which software applications operate are increasingly dynamic. By context we mean the properties of the environment in which a cloud application operates that have an effect in its behavior. For example we may want a cloud application to change its security behavior depending on where (location) it is used, who (subject) is using it, or when (time) it is being used. As a result of dynamic context, the assets, their values, and attack scenarios can change easily from one situation to another, increasing the challenge of finding out what the information assets are, who their owners are, where in the system vulnerabilities lie, and the extent to which the security requirements are satisfied.

A security mechanism that works in one usage context may not work in another. Cloud applications often have a diverse set of consumers with different security requirements that need to be enforced across different contexts. For cloud applications to comply with the various security requirements despite changes in contexts their security mechanisms need to be flexible. We called this Adaptive Information Security (AIS).

We propose policy-based access control as a means to enabling AIS. Our approach is rooted on the software engineering concepts of traceability and requirement satisfaction arguments. We argue that traceability is one of the key prerequisites for AIS in the cloud as it enables understanding of the relationship between security requirements and the policies that satisfy those requirements. Adaptive applications change their behavior based on different conditions determined by attributes of their contexts. Therefore, a good understanding of context is a key ingredient for engineering adaptive applications.

For example an adaptive application may need to satisfy a security requirement to keep a document confidential when accessed across a range of situations such as different locations (office, home, restaurant) or different devices (such as phone, tablet, laptop). Each of these situations are examples of different contexts and for each context a different policy may be required to satisfy the security requirement. For instance, as the office is a secure environment we could allow the user to access the document without any authentication. However, in a restaurant we may apply a policy that require authentication before access is granted. For each of the different contexts in our example a particular policy is applied. Without a good understanding of the context may not be possible to understand which policy need to applied under which situation. However, current approaches to traceability do not provide support for taking into account contextual attributes. In our proposed approach we use entailment relationships to capture and enrich traceability links with context. We then use the links enriched with context to reason about and diagnose violation of security requirements.

This paper makes contributions at two levels. The first contribution is the use of traceability as a way to understand the relationship between requirements and the policies that satisfy security requirement in adaptive applications. The second contribution is the enrichment of traceability links with contextual information in order to support the adaptation of cloud applications to context that was not clearly understood, unknown, or incomplete at design-time.
The paper is organized as follows: §2 presents an example scenario that is used to motivate and explain our work. §3 motivates the needs for traceability as one of the key requirements to achieving adaptive information security. §4 presents our approach to enriching traceability with contextual information and §5 shows how we applied our approach in re-engineering an open source cloud application for adaptive security. §6 discusses related work and §7 concludes the paper and discusses pointers for further work.

II. EXAMPLE SCENARIO

As a motivating example consider a cloud provider, CloudX, that offers an image storage and processing service through an application, ShutterClick, as illustrated in Figure 1. MediScan and NewsMedia are cloud consumers with subscriptions to ShutterClick. The cloud consumers have common information security requirements (which they all share) and specialized information security requirements which are specific to each consumer. For example, MediScan has specific information security requirements for its administrators, doctors, and patients; whilst NewsMedia also has its specific set of information security requirements for journalists, photographers, and editors. Adam is the CTO at CloudX. Bob is an employee of MediScan responsible for auditing a Service Level Agreement between MediScan and CloudX. Sara is a developer at CloudX.

Furthermore, the different employee roles (e.g., doctors, journalists, etc.) of the cloud consumers, may use ShutterClick in different contexts. For example a doctor may want to retrieve a medical scan using his office Ethernet, home Wi-Fi, or the public 3G network. In each case, ShutterClick needs to adapt its access control mechanism depending on the doctor’s location when attempting to access the information. When accessing through the hospital Ethernet network no authentication is required but if the doctor is accessing from outside the hospital (through a 3G network), authentication is required to preserve the confidentiality of medical records.

III. TRACEABILITY FOR ADAPTIVE INFORMATION SECURITY

Current approaches to developing adaptive security applications are based on enumerating the different contexts in which an application operates and engineering behaviors that would be applied exclusively in each context [4][5]. These approaches only work when the application always encounters known contexts whose attributes were engineered into the application at design-time. They do not handle how the application should behave at run-time when it encounters contexts whose attributes were unknown, ambiguous, or incomplete at design-time. The diversity of cloud consumers and their contexts means that it is inevitable that a cloud application will have to operate securely in a context that was not known at design-time. To address this problem, cloud applications need the ability to reason about unknown or incomplete contextual attributes and assess the impact of changes to such attributes on the satisfaction of security requirements. Such impact analysis can be achieved only if the relationship between requirements and the enforced policies is well understood and maintained.

Therefore, when engineering cloud applications to support adaptive information security it is crucial for a cloud provider to understand the relationship between requirements and policies. Context is crucial in engineering adaptive applications; this relationship should be understood in terms of the context. In software engineering, such relationships are called traceability information, which can help the cloud provider in many ways. For example, monitoring the violation of information security requirements at run-time; identifying the policies responsible for the satisfaction of those requirements; assessing completeness and satisfaction of requirements; and certifying the assurance of security requirements.

Traceability links between requirements and policies need to be captured at design-time. In cloud applications, the traceability effort entails both forward and reverse

![Figure 1. A Cloud Application Usage Scenario Diagram](image-url)
traceability. In forward traceability, requirements are traced to the components that implement the behavior satisfying the requirements. Upon violation of security requirements at runtime, the reverse traceability should make it possible to diagnose the components that contributed to the violation.

Continuing with our example in Figure 1. Sara uses requirements from MediScan and NewsMedia to derive the design of the access control functionality of ShutterClick and the policies that govern the run-time adaptation of functionality required to handle changes in user’s context. When the security requirement is violated it is often difficult for Sara to ascertain which parts of the existing ShutterClick application contributed to the violation. To address this problem, at design-time, Sara needs to ensure that each feature required by MediScan and NewsMedia is traced to the design of ShutterClick. She needs a systematic technique of maintaining traceability between the security policies that control the adaptive behavior of ShutterClick and the requirements from which these policies are derived. Sara also needs the traceability to be invariant – i.e. it should not be accidentally altered as ShutterClick adapts its behavior to satisfy the security requirements of its end users in the different contexts at runtime. The traceability techniques need to incorporate context information as part of the traceability links in order to understand the effect of changes to contextual attributes on the traceability links.

IV. ENRICHING TRACEABILITY WITH CONTEXT

Adaptive security in cloud applications requires a clear understanding between the security requirements and the solutions that implement the requirements. This traceability needs to address two key questions: (1) how do we know the set of components that interact to satisfy a particular requirement?; and (2) in the event that a requirement is violated (not satisfied) how do we trace the components whose failure has contributed to the violation of the requirement? In adaptive applications the understanding of the characteristics of context by system designers is important as contextual attributes are what drives adaptation. In our previous example of adaption on securing confidentiality of a document, location is an example of a contextual attribute. In this section we first describe an existing notation we use to make context explicit and describe the details of our approach to enriching traceability with context to support the understanding of the relationship with security requirements and policies by both cloud providers and end-users.

A. The Entailment Relation

The basic premise of our approach is the use of Zave and Jackson’s entailment relation [6][7] to capture and reason about traceability in cloud applications. The entailment relation relates three sets of descriptions: requirements, domain assumptions, and specifications. A requirement (R) describes a condition or capability that must be met or possessed by a system, that is, its purpose. Requirements are optative descriptions in that they describe how the world would be once the envisioned system is in place. For a medical record confidentiality (MRC) feature of ShutterClick this could be: ‘Only the assigned doctor of a patient can read, write, update, delete, or share a patient’s record. The assigned doctor is allowed to perform these actions only when they are on duty’.

Domain assumptions (W) describe facts about the behaviour of the environment where a system will be deployed. In this paper we use the term context to refer to the environment described in domain assumptions. In contrast to requirements, domain descriptions are indicative in that they describe objective truth about the context. In the MRC example this could be: ‘a GPS device gives precise location in terms of coordinates expressed as latitude and longitude’, and ‘WiFi provides connectivity allowing a device to be connected to a network to communicate with other devices’.

Specifications (S) then describe how the system should behave in order to satisfy the conditions described in R, given that the assumptions described in W hold. The specification for the MRC function could be: ‘To maintain confidentiality of the medical record access it should be controlled through an access control mechanism. Only users of the ShutterClick who are designated as doctors of the patient can have access to the record’. In general the relationship between the three sets of descriptions explained is given by (3.1) below.

\[ W, S \vdash R \quad (3.1) \]

WHERE $\vdash$ is the entailment operator.

The entailment relation as expressed in (3.1) does not prescribe languages for expressing the three artefacts. This absence of prescription has the advantage of giving the requirements analyst freedom to choose a language of their choice for representing details of the three descriptions. In order to support the argument that (3.1) holds we need to know the details about the specific behaviour of S and W. One way to describe the three artefacts (S, W, and R) is in terms of events and state changes.

B. Context as a Traceability Link

The entailment relation relates requirements (R), context (W), and specifications(S). Our approach leverages this representation to relate requirements to policies. We assume that policies are in the solution space as their behavior satisfies the requirements - we regard them as specifications. We can think about context as the traceability link between requirement and policy as described by the following relations.

\[ \text{Policy} \leftrightarrow T \leftrightarrow \text{Requirement} \quad (3.2a) \]

\[ \text{Policy} \leftrightarrow \text{Context} \leftrightarrow \text{Requirement} \quad (3.2b) \]

Relation 3.2a represents a general traceability T between a requirement and a policy. In 3.3b we propose that this traceability can be represented through the context. Figure 2
presents the general framework of our approach. The figure shows how we propose to relate entities in the problem space to those in the solution space through context. Requirements and policies are in the problem and solution spaces, respectively.

**Figure 2: Relating Requirements and Policies through Context**

Using our running example, the entities in Figure 2 can be instantiated as follows:

**Requirements:**
- R1: Ensure doctors can access medical records only on-site.
- R2: Prevent access by non-doctors.

**Policies:**
- P1: if subject is a doctor and is on-site -> do not authenticate (NAUTH);
- P2: if subject is a doctor and is not-on-site -> authenticate (AUTH);
- P3: if not doctor -> Deny

**Context:**
- W1: if the doctor is within certain GPS coordinates -> the doctor is on-site.
- W2: if the doctor is accessing through the hospital WiFi -> the doctor is on-site.

In this example without context, it is not obvious how to relate the requirement and the policy. For instance, the domain knowledge provided by context is necessary in order to explain what ‘on-site’ means in both the requirement and policy. Therefore we need to make the context explicit as a way of linking requirements to their policies.

**C. Rationale for Using Context for Traceability**

The context makes explicit the properties of the world (attributes) whose values are important in determining the behaviour of the application. The values of these attributes need to be monitored so that the applications is able to adapt itself by selecting the correct behavior (policies) suitable for the context at hand. In essence the context specifies three things: (1) What world properties are important for adaptive behavior?; (2) What values of these attributes are of interest in determining adaptive behavior; and (3) what behavior (actions) should the application adopt/perform when those properties assume the values of interest.

In our example the world properties (contexts) are the attributes of the subject trying to access the medical record and their location. The subject can assume the values doctor or non-doctor. The location of the subject may be on-site or off-site. The location value ‘on-site’ means the subject is physically on the premises of the hospital when requesting access to the medical record. Meanwhile, ‘off-site’ implies that the subject is not on the hospital premises.

Policy P2 states that if we can establish that the subject is a doctor and he is currently outside the premises of the hospital then we allow them access to the medical record after authentication to validate that they are who they claim to be. On the other hand, P1 states that if the subject is a doctor and he is currently on-site then ShatterClick should allow them access to the medical record without any authentication. P3 states that if the subject is not a doctor then we should deny them access to the medical record (P3), regardless of their location.

We can think of the traceability link between requirements and policies in terms of two links: the first link is between the problem and context domains. The second link is between the context and solution domain. An interesting question is what specific attributes to consider in the requirement, context, and solution domains in order to establish these links. The next two sections address this question.

**D. Traceability Link Between Requirements and Context**

According to the entailment relation a requirement is defined as some property that must be exhibited by an application in order to solve some problem in the real world. For this reason we express a requirement in terms of the conditions we would like to be true in the context once the system is in place. The expression of a requirement references some attributes of the context. For example according to R1 and R2 only doctors should have access to the medical images stored in the cloud - subjects who are not doctors should be denied access.

**Figure 3: Traceability Link Between Requirements and Context**

In our example the requirement R1 says doctors are allowed access to medical records only when they are within hospital premises. But what does being within the premises of the hospital mean? W1 and W2 tell us that if the doctor is within certain GPS coordinates or he is using the hospital WiFi for connection then he is within the premises of the hospital, respectively.

The example illustrates that the requirement references the context. It is through such references that we relate the requirement to the context. The relationship between the
requirement and context is captured by traceability link L1 in Figure 2. In Figure 3, x represents the attribute in the context that is referenced by the requirement. In our example, x has the value ‘on-site’. Using domain knowledge, the context qualifies what ‘on-site’ means by stating how it is determined. Such reference provides a concrete relationship between the requirement and context thus forming a traceability link.

E. Traceability Link Between Context and Policies

Link L2, in Figure 4, represents traceability link between the context and policies. Similar to entities the problem space, solution space entities also make assumptions about the behavior of the context. The solution uses these assumptions to implement behavior that enforces the satisfaction of the requirements. For instance the policies in our example implement access control mechanisms that determine who can or cannot access a medical image. Hence policies also references some attributes of the context. Again, we take advantage of these references to establish the traceability link between policies and context.

In our example policy P2 states that if a doctor is not on-site then they should be authenticated before being granted access to a medical record. P1 says that if the doctor is on-site then no authentication should be applied. For these policies to determine if the doctor is on-site they assume that some machinery is available for sensing the location information. In our example the location sensing is performed by the GPS device, which we assume the doctor carries with him. We use the reliance of the policy on properties of the context to establish a traceability relationship between the two. Worth noting is that the policies are formulated to enforce the satisfaction of the requirement in different contextual conditions. For example, we say that we have established through the context that the doctor is on-site then we do not require any authentication.

The relationship between x and y is defined by the behavior of the context – called a domain description [7][6]. For example a domain description could say: A GPS device gives its location in terms of latitude and longitude coordinates. In this statement latitude and longitude are numbers that, on their own, do not have any meaning. The description gives these numbers a meaning by stating they are a location. We can go further and enrich this description by stating that these coordinates are the location of the hospital. Similarly, ‘location of the hospital’ would be meaningless unless we can say exactly what geometric reference system we are using to locate it. The fact that location of the hospital is assigned to certain coordinates enables us to derive further facts such as whether the doctor is on-duty or not. Therefore, x and y are dependent on each other. We exploit this dependency to define traceability within the context through domain descriptions.

In summary by using traceability links enriched with contextual information we are able to relate entities in the problem space to entities in the solution space. We achieve this in two stages: (1) relating requirements to context; and (2) relating the context to policies.

F. Requirements Violation Diagnosis

While the analysis and refinement of policies can ensure that policies correctly implement behavior that satisfies the users security requirements, the highly dynamic contexts in which cloud services are used mean that the policies might not capture all possible security threats. To ensure that the adaptive application can detect when its security requirements are no longer being satisfied at runtime our approach includes requirements satisfaction information as part of the traceability links. Using the entailment relation, we can express the relationship between the requirements, context, and policies as Ws, Ps | Rs. This states that the behavior of the policy Ps satisfies the security requirement R given that some assumptions about the context Ws holds.

In our running example the security requirement is satisfied if the doctor supplies his credentials, a GPS device (carried by the doctor) supplies the correct location of the doctor, and the policy Ps behaves according to its specification. Provided the doctor has supplied valid credentials, his location according to the GPS device is such that he is within the premises of the hospital, i.e. he is on-site, then he should be allowed to read, update, write, and delete the patients medical record. Otherwise he must be denied. The argument that the security requirement will be satisfied is the satisfaction argument. For our medical record confidentiality problem, the satisfaction argument can informally stated as follows:

“If the doctor supplies valid credentials, he is the patient’s assigned doctor, the doctor is carrying a GPS device, the coordinates from the GPS device are valid and say that the his location is within the premises of the hospital, allow him to access the medical record. Otherwise deny.”

If the above argument is correct we can say that the security requirement is satisfied. The correctness or validity of the argument depends on a number of assumptions we make about the context and MRC machine. Some of these assumptions are: A1: The GPS device is well calibrated to give a correct and valid reading about the location of the doctor. A2: The doctors always carry the GPS device. A3: The hospital WiFi has a unique identifier, which makes it possible to uniquely identify it among other WiFi networks. A4: The device that reads the doctor’s authentication credentials reads them correctly. A5: The file system where the medical record is kept has features for read, write, delete, and update operations.
The satisfaction argument encompasses the requirements, specification, and context as well as the relationship between these three descriptions and the assumption/conditions under which the arguments can be assumed valid. We use the satisfaction argument to represent traceability between the requirements and the policies that implement the requirements.

Our representation of traceability with the entailment relationship makes explicit the context in a cloud application - which we consider essential component in reasoning about traceability between requirements and design for adaptive security. We also capture satisfaction argument as part of the traceability link at design-time. At run-time we re-evaluate the satisfaction arguments to check certain properties about the security requirement. For example, we can check if the requirement is being satisfied. If not, we can diagnose the failure by identifying the components whose failure has contributed to the failure of the requirement.

In our example, if the coordinates read from the GPS device indicate that the doctor is off-site, and the doctor attempts to access the medical record, access will be denied. A re-evaluation of the satisfaction argument indicates that the access request has been denied because he is currently off-site. The traceability link from the requirement to the context indicates that the component responsible for monitoring location is the GPS device. Sara, the developer at CloudX, can easily gather this information because the traceability links on both the requirement and policy sides of the context would contain references to some attributes of the GPS device as a component in the context of the application. Worth noting is that, in this case, the failure of the requirement is not because the GPS device has necessarily malfunctioned. It may have given the correct reading to the ShutterClick application only that this reading indicates that the doctor is outside the hospital and hence according to the satisfaction argument he should not be allowed access to the medical record.

V. DISCUSSION

Our approach to traceability is beneficial to both the end-users and providers of cloud applications. We have identified three benefits which we explain in this section: (1) change propagation minimization, (2) exposure of security requirements violation, and (3) ability to reason about requirements satisfaction. We make qualitative arguments to support our choice of these benefits.

A. Change Propagation Minimization

Change propagation is one of the key tasks in software maintenance [8]. It entails the propagation of a change made to one artifact to other artifact that needs to be updated as a result of that change. We show that with our proposed traceability approach it is possible to determine what kind of change is allowed, not allowed, important, or not important. This is useful because if a developer knows how a change will affect the satisfaction of a security requirement he will be able to decide whether to effect that change or not, thus minimizing the need to propagate unnecessary changes.

To evaluate this we changed the values of the GPS coordinates that determine the hospital space. After this change we re-evaluated the requirement satisfaction argument of the traceability. After this change accessing the medical record at the hospital failed - thus the satisfaction argument was not satisfied. This indicated that the GPS coordinates play an important role in the satisfaction of the of the requirement hence a change of their values should not be permitted unless intended and dictated by a change in the requirement. If such change is intended then it should be propagated to update the policy.

Encryption was initially turned-off in the WiFi router. We enabled it and re-evaluated the satisfaction of the access control requirements. The requirement is satisfied and that meant that the change in the encryption is not critical to the satisfaction of the requirement. Thus such change is not important and hence does not need to be propagated to the policy. The above scenario demonstrates that our approach minimizes the need for propagating unnecessary changes. This insulates the end-user or developer from looking at less important changes in order to focus on only essential changes.

B. Ability to Expose Security Requirements Violations

We show that without enriching traceability with context/domain knowledge security requirements are violated. Assuming a refined version of the access control requirement in we want medical records for patients admitted in the intensive care unit (ICU) to be only accessible when the doctor in the ICU. The ICU is located in the middle of a huge ward. During the festive season, due to increase in the number of patients requiring intensive care, the hospital administration decided to use the surrounding ward as part of the ICU.

If the traceability links do not incorporate context information, when a patient is moved out of the original ICU room their medical records become inaccessible. This restriction on access violates the security requirement as the boundary of the ICU has now increased. With our approach, when the requirement is updated with the new boundary of the ICU, we are able to evaluate such a change against the context traceability links and propagate the change to context and policy - thus preventing violation of the security requirement.

This scenario illustrates that if there is a change in the requirement that requires updating the policy, that change should be effected on the policy. Otherwise the requirement is not satisfied. Our approach provides live traceability between requirements, context, and policy. If context changes, the policy needs is notified so that it makes the most up-to-date knowledge about the assumptions it is making about the context. Changes in context are
intertwined with changes in policy otherwise the security requirement will be violated because the change.

C. Ability to Reason About Requirements Satisfaction

We show if the traceability information captured using context enriched links makes it possible to reason about the impact of changes arising from both the developer’s side or from the end-user’s side. In particular, we evaluate if having the knowledge of W from the end-user’s side, the developer can judge W, S |- R is satisfied. We also evaluate if the end-user can determine W, S |- R is satisfied by knowing W from the developer’s side.

In our example, the part of the context that is on the end-user’s side is the Boolean variable attribute ‘on-site’. Whether this variable evaluates to true or false is determined by GPS and WiFi entities on the developer’s side. The developer’s and user’s sides of are related by the context. Assume we have the GPS device to determine that the doctor is in the hospital and we used any other connection (such as a public GSM network) to link to the CloudX server instead of restricting to the hospital WiFi. The GPS device will determine that the doctor is in the hospital and the public connection will provide the connection to access the medical record from ShutterClick.

Although, this seems to satisfy the access control requirement, we can argue that it actually does not. This is because an attacker tempering with the public network may still violate the confidentiality of the medical record. The motivation for using the hospital WiFi instead of any other connection is because this provides a more secure link than a public connection whose security properties we may not be able to ascertain. This scenario demonstrates the need for the completeness of W. Such completeness is necessary in order to correctly argue that the requirement is satisfied.

From both the developer and end user’s side we can show that the entailment relation W, S |- R holds because the domain properties tell us that GPS provides us with a precise method of measuring location, the WiFi connection provides a secure transmission medium for transferring files. Based on these facts, we can be certain that the satisfaction argument that the policy satisfies the requirement holds regardless of the side (whether end-user’s or developer’s side) from which it is evaluated. This is possible because we have explicitly captured the context.

By explicitly considering the context, the developer can (through a satisfaction argument) tell that behavior provided by the GPS device and hospital WiFi is sufficient to satisfy the access control requirement. Without that knowledge he cannot judge that the security policy satisfies the access control requirement. Similarly, without knowledge of what GPS coordinates are for the hospital then end-user can not determine if WSR is satisfied. Therefore, having the complete traceability through the context enables validation of satisfaction of the requirement from both the end-user’s and developer’s ends.

VI. CASE STUDY

We applied our approach to the case study of an open source file-sharing cloud application called ownCloud (http://owncloud.org). ownCloud is a Dropbox-like cloud service for storing, syncing and sharing various types of files, such as documents, music and calendar. It is open source software that runs on multiple platforms. Although ownCloud has implemented important security mechanisms, such as the use of https for client-server communication and a database-driven access control for file sharing, its security mechanisms are not currently adaptive to user context. We re-engineered ownCloud for adaptive access control by integrating it with XACML and simulated how it responds to variations in context to demonstrate the utility of our approach.

A. Access Control in the Existing ownCloud Application

The access control architecture is captured in figure 5. Access control decisions are based on permissions stored in database tables. Permissions are granted per user per resource. The available permissions are create, read, update, delete, and share (CRUDS).

A user requesting access to a resource is authenticated through their credentials. If the credentials are correct a decision is made based on the permissions stored in the permission database. A sample permissions database table is as shown in Figure 6.

![Figure 5: Architecture of ownCloud Application Access Control](image)
the access control mechanism in ownCloud does not adapt as the context in which the cloud application is used changes.

Figure 6: ownCloud Permissions Table

B. Adaptive Access Control in ownCloud

To address the challenges stated above we re-engineered ownCloud to allow for the specification of more generic policies which allow for more flexible and adaptive access control. Figure 7 shows the improved architecture for adaptive access control that we implemented. For our example we wanted ownCloud to adapt its behavior as his location changes to ensure that the confidentiality requirement of a medical record continue to be satisfied.

Figure 7: Architecture of ownCloud after Re-Engineering with Adaptive Access Control

To achieve such adaptive behavior, we introduced a number of components. On the user device side we a location monitor, which determines the location of the user using a GPS device. When a user makes a request to a medical record their credentials and their GPS coordinates are sent to the ownCloud server (1). If the credentials are valid ownCloud consults the location database to determine if the coordinates are for on-site or off-site (2). For purposes of simulation we hard-coded the GPS coordinates in the server. A request is then sent to the XACML policy engine to determine if given the user can be allowed access (3). The XACML engine consults its policy database to determine the policy that matches the subject, resource, environmental attributes, and action of the request (4). A sample policy is show in Figure 8. We had three policies for on-site, off-site, and non-doctor.

Figure 8: Sample XACML Policy

A deny or permit response is then sent to ownCloud (5). In our example a request is denied if either the doctor is not on-site or the user requesting is not a doctor. Depending on the response, ownCloud either sends the contents of the requested file to the user or send a message informing the user that their request has been denied (6).

C. Evaluation

We set up an experiment to evaluate the effect of enriching traceability with context. Figure 9 shows the geometry of the ICU Room. For this experiment we assumed that the security requirement is to 'ensure that the medical record is accessible only inside the ICU'. We then tried accessing the medical record with three variations of the context. We varied the context in terms of the technology used for determining the location of the ICU room. Z1, Z2, and Z3 are the different locations from which access was tested. Positions Z1, and Z3 are located inside the ICU while Z2 is outside the ICU. The scenarios are described below.

Figure 9: Geometry of the ICU Room

Scenario 1: In this scenario we used the hospital WiFi as a way of determining the current location of the doctor. We described our context such that if he is using the hospital WiFi then we assume that he is on-site. Otherwise, we assume he is off-site. In all locations Z1, Z2, and Z3 access to medical record was granted. Accessing the medical record from Z2 violated the access control requirements as Z2 is outside the ICU.
Scenario 2: We then recorded one GPS coordinate at position Z1 as an attribute for determining the location of the ICU. Z1 is at the center of the room. We changed the context description to say that if the doctor’s GPS coordinates match position Z1 then he should be considered on-site. Otherwise he should be considered off-site. This restricted access from position Z2. However, it also restricted access to the medical record when the doctor was in Z3. This partially satisfied the access control requirement as access was granted in only some parts of the ICU room.

Scenario 3: We further enriched the context on how we determine location by recording four GPS coordinates at positions A, B, C, and D. We then revised the description of the context on how location is determined to say that if the doctor’s GPS coordinates say that he is inside the boundary marked by the four GPS coordinates then he is on-site, otherwise off-site. With this description of context the location of doctor could be determined more accurately and access was granted in all GPS coordinates inside the room. Access was denied from outside the ICU room. Table 1 shows a summary of the experimental results.

<table>
<thead>
<tr>
<th>Scenario #</th>
<th>Contextual Attributes Used</th>
<th>Access Status Different Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WiFi</td>
<td>Z1: Granted</td>
</tr>
<tr>
<td></td>
<td>Single GPS Coordinate</td>
<td>Z2: Denied</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Z3: Denied</td>
</tr>
<tr>
<td>2</td>
<td>Multiple GPS Coordinates</td>
<td></td>
</tr>
</tbody>
</table>

This experimental simulation demonstrated that the richer the context captured in the traceability link the better the accuracy of satisfying the security requirement. If the geometry of the ICU room is not taken into account in the context we either allow more access than we should or over-restrict access. This also showed that if the context is not rich enough the security requirement is not always satisfied. On the contrary, with a richer context we were able to satisfy the requirement more accurately.

VII. RELATED WORK

Researchers in the software engineering and security domains realized the significance of the problem of ensuring that security requirements continue to be satisfied when an application operates in dynamic environments. Several approaches have been proposed on how software systems should be engineered in order to address this problem [9][6]. In this section we review the most relevant approaches to adaptive security and traceability.

Adaptive Mechanisms for Security Enforcement: The extensible security infrastructure [10], the strata security [11], monitoring and switching problems specification [4], the adaptive trust negotiation framework [12]. A key limitation of these approaches to adaptive security is that they are design to deal with changes that are known at design-time. Moreover, they do not provide mechanisms of relating requirements to the implements that satisfy those requirements. In our approach we support adaptation by capturing traceability links with contextual information at design-time, which we use for reasoning about the satisfaction of requirements at run-time. We use the traceability links to better understand the relationship between requirements and design. Such understanding help in formulation better strategies for the application to adapt to its context.

Relating Requirements to Design: Traceability from requirements to design [13] is key for understanding an evolving system and making necessary changes in response to user requests. Bauer et al.’s [14] approach for model-based security assurance allows for formally verifying design models against high-level security requirements such as secrecy and authentication on the specification level and helps to ensure that their implementation adheres to these properties.

While this approach may be effective when the environment in which the application is used is static, in dynamic environment it may fail to detect vulnerabilities, as it does not explicitly model the context. Our approach explicitly model the behavior of the context as part of the traceability links and this makes our traceability links to be more informative in terms of the reasoning about violation of security requirements from both the end-user and developer’s sides of an adaptive application. One of the most popular traceability techniques is the traceability matrix. This technique has been enhanced in several ways including state diagrams [15] and topic modeling [16]. The fundamental limitation of traceability matrixes is their inability to show the role of context in relating requirements. This renders them inadequate for capturing traceability in adaptive applications.

The notion of invariant traceability, where the traceability between the design and requirements holds even when the system undergoes changes have been shown to be effective [17]. However, this concept has so far only been applied to software maintenance activities. In our approach we are using the idea as a mechanism of ensuring that adaptive applications satisfy their requirements in dynamic environments.

For software maintenance bi-directional transformations [18] have been proposed of maintaining invariant traceability [17]. With these transformations separate traceability links are required for forward and reverse traces. With our approach a single traceability link can be used for both forward and reverse trace tasks. This is made possible by the fact that our traceability links contain references to the requirements, context, policies, and satisfaction arguments. The satisfaction arguments explicitly relate the requirements to the policies through context. In this way regardless of what changes occur in the context, the relationship between the requirements and policies is
maintained. Our approach also allows for a ‘live’ validation of this relationship by providing the capability to re-evaluating this relationship at run-time.

Satisfaction arguments: Our use of satisfaction arguments as a way of capturing and reasoning about traceability is similar to the idea of dependability arguments with trusted base proposed by Kang an Jackson [19]. While they use the arguments to determine if a set of components in a trusted base to establish a requirement, we use the satisfaction arguments to detect if a new change in context or policy has violated a security requirement.

VIII. CONCLUSION AND FURTHER WORK

This paper argues that security in cloud applications need to be adaptive in order to cater for the different variations resulting from requirements of diverse cloud consumers and the different contexts in which they use cloud applications. We have proposed that traceability is a key prerequisite for adaptive information security as a way of understanding the relationship between security requirements and the policies that enforce those requirements. For effective maintenance of this traceability, the paper has shown the need to embed contextual information as part of traceability links. We have proposed an approach to traceability that is rooted on representing traceability links as entailment relations and requirements satisfaction arguments.

Our evaluation shows that by using our approach it is possible to maintain traceability in an application despite changes in context. Through our traceability approach we are also able to reason about the behavior of the application at run-time to detect violation in security requirements and identify the components whose failure has contributed to the violation. We plan to extend our approach so that it is possible use the traceability links as a means to generating mitigation strategies to help a cloud provider prevent the further requirements violation. We will also extend our approach so that it augments the policy-based adaptive information security system with runtime monitoring and feedback mechanisms that will give users assurance that their requirements are being met.

ACKNOWLEDGMENT

This work was made possible by the support of a grant (NPRP 05-079-1-018) from the Qatar National Research Fund (QNRF). The statements made herein are solely the responsibility of the authors.

REFERENCES


