Abstract

Current annotation technologies suffer from poor coverage over the development process phases, limited support for the broad scope of the security requirement types and inadequate interconnection between different process phases. In this paper, we introduce the concept of multi-level security annotation for software artifacts, which addresses these problems. Multi-level security annotations are traceable through the development phases to provide a light-weight, user-friendly alternative to full-fledged assurance methodologies and simplify the system evolution over time with respect to security.

1. Introduction

Annotations allow developers to enrich software artifacts, like class design and code, with auxiliary information. A trivial example is a class method annotated with the ‘deprecated’ tag. We have surveyed existing annotations technologies, with particular focus on security. In this paper we highlight the major existing shortcomings and outline our research roadmap in the field.

Coverage over process phases. Annotations do not cover the entire spectrum of the Software Development Life-Cycle (SDLC). Most of the existing technologies work at code level. For example, the Security Annotation Framework (SAF) [2] defines a set of Java 5 annotations that can be used to decorate application code with access control information. These annotations are then used by the SAF runtime as interception points where the security policy must be enforced. There are also some annotation technologies applicable at other levels of the SDLC. For instance, UMLsec [1] is a front-stage representative at design level. Because of space limitations, an extended discussion of existing technologies is not addressed in this paper. However, it can be observed from the state-of-the-art that existing support is still inadequate.

Coverage over requirements types. Annotation frameworks also differ in terms of the expressiveness they provide. Often, they specialize in a limited set of security features, while other are poorly represented. For example, SAF supports only access control requirements. Hence, properties like integrity, which can be represented in UMLsec, cannot be mapped to SAF annotations.

Integration. Even if different technologies were available for a given security property and applicable at different phases of the SDLC, still challenges would likely be confronted when trying to integrate and translate annotations from one level to another. For example, the technologies may assume different access control models (e.g., role-based vs. attribute-based), have different granularity (e.g., resources vs. operations), and so on. In this aspect, more integration across phases is needed. Also, integration is necessary across different security properties at the same level. Indeed, often two properties cannot be viewed separately; they are closely connected like, for example, access control and authentication. These dependencies should also be taken care of by the annotation framework.

We can conclude from the above observations that annotation technologies, as of now, do not cover the entire SDLC, are not representing the entire spectrum of security properties and are disconnected from one another. Our research agenda aims at filling these gaps in a step-wise fashion, that is, we will select an increasingly larger set of security requirements types, and, per each type, we will address the above-mentioned issues throughout the SDLC. The starting point is represented by the confidentiality requirement type and the supporting authorization and authentication mechanisms. This is, in a sense, a low-hanging fruit compared to more complex security requirements. However, it is a very popular (and fundamental) one
and hence it has the highest pay-off potential. For these reasons, in the rest of this paper we often refer to confidentiality requirements, although our research horizon goes clearly beyond.

Currently available security annotation technologies are difficult to use in the context of software assurance because of the lack of traceability support. For example, once the security constraints are elicited and documented in the requirements phase, e.g., as annotations to the use-cases, it is nearly impossible to verify whether all the constraints have properly propagated all the way down through the cycle and are still present at code level. The way around, it is also hard to trace code annotations back to the requirements. Thus, even if one can enforce all the constraints as specified by the annotations at the code level, the overall satisfaction of the security requirements cannot be “verified” by looking at the different levels of annotations throughout the various artifacts. This is a missed opportunity for annotations, which could represent a light-weight, user-friendly alternative to full-fledged assurance methodologies (e.g., formal V&V). Furthermore, traceability would simplify system evolution over time with respect to security.

2. Multi-level security annotations

Our research in the domain of multi-level security annotations is still in an initial state. This section provides a description of the methodology that, when followed, can lead to a definition of the annotation sets for various security requirements at different phases of the development processes. This section additionally describes how the traceability can be achieved and the practical advantages of it.

2.1. Method

To conduct our research on multi-level security annotations, we have defined a step-wise, iterative approach. As we aim at having an annotation methodology that is general and can be applied in different application domains, the framework resulting from our research need to be generic enough to cover different cases and able to be refined for the specific needs of different domains (e.g., health care, telecom). Therefore we have decided for an iterative approach where we can refine our results while performing iterative steps over different case-studies in different domains.

The step-wise character of the proposed approach results from the complexity of the problem and the large space of the security requirements. Therefore we will start from a smaller set of security requirements types and per each type we will address the problems with the current annotation technologies like those we have mentioned earlier. This exercise will clarify the existing technical gaps and will drive the definition of a novel annotation framework. At each step we will select a larger set of requirement types and repeat the exercise in order to extend the coverage provided by our framework.

2.2. Traceability and assurance

As we have stated in Section 1, multi-level security annotations can be used to achieve better assurance for security properties of software. The key feature that allows this is the traceability property. Using the traceability property it is possible to verify if the security requirements, as defined in the requirements phase, are still present in the final product, on the code level (and the deployment – configuration). Thus, once we enforce all of the requirements, as specified by the annotations on the code level, we achieve a degree of assurance that the the security requirements, as specified at the requirements phase, are fulfilled with the present security mechanisms. This way annotations could represent a light-weight, user-friendly alternative to full-fledged assurance methodologies (e.g., formal V&V).

An other important aspect of the traceability property is the two dimensional character of the dependencies between the annotations. Besides the vertical dependencies between the annotations at different phases of the process, the security requirements can depend on each other, on the horizontal axis. Security requirements can have different relations to each other, for instance, as observed in [3] identification requirements are dependent on authentication requirements, as identification requirements are seldom of much value alone and in most cases they must be accompanied by authentication requirements. An example of how the dependencies between the security requirements can be modeled (for building a requirements repository purposes) can be found in [4].

The importance of the dependencies between the security requirements for the traceability property becomes more clear in the example provided in Section 3. In that example, the confidentiality requirement is refined by the finer-grained authentication and authorization requirements. This dependency needs to be documented as an annotation as it could become unclear whether the confidentiality requirement is still met at later stages, where this requirement is represented by its substitutes.
The traceability property can be achieved by linking annotations to each other, either by defining a suited syntax for that purpose in the annotations themselves, or by writing down the dependencies between the annotations in a separate document (i.e., by means of meta-annotations).

The traceability property of the multi-level security annotations, aside from providing a possible alternative for assurance methodologies, can also facilitate security-related development and maintenance tasks. For example, traceability would simplify system evolution over time with respect to security. While iterating over the process, the changes that result from different iteration steps can propagate through all phases of the process together with the security requirements documented as annotations. Also, when a security mechanisms needs to be replaced (e.g., because a new vulnerability is found), this task is facilitated due to the better visibility of security sensitive software entities (e.g., functions, data), as they are properly annotated.

Altogether, the multi-level security annotation have the potential of increasing the overall security of the developed applications.

3. An illustrative example

This section provides an example of a confidentiality requirement propagating throughout SDLC phases as multi-level annotation. The provided examples of suggested annotations are purely illustrative in this context.

Requirements phase. Additionally to documenting the basic functionality requirements, a use case can be extended with security constraints. For instance, a use case stating that “an information asset (e.g., a patient healthcare record) must be accessible to the users” can be constrained by a requirement: “the information asset must be kept confidential to authorized users”. This constraint can be documented by annotating the use case:

(A) The asset itself is annotated as confidential in a given use context.
(B) The asset annotation is expanded with information about the type of users that are allowed to access the asset.

The two parts of the annotation can be viewed as a confidentiality constraint decomposed into authentication and authorization requirements.

Architectural design phase. When, for example, a Service Oriented Architecture (SOA) is built, the access to the above-mentioned asset is provided via an operation documented in a WSDL. The access control requirement (A) requires a finer-grained view on the application in order to be enforced. Therefore this requirement remains unhandled at the architectural design phase, i.e., it is deferred to a later stage in terms of annotations. Given the web-based environment, the authentication requirement (B) is further detailed at this level by annotating the operation with the Single Sign-On (SSO) constraint. The link between the SSO annotation and the authentication requirement (B) can be made explicit by documenting it separately or by extending the annotations with the linking information.

UML design phase. The operation specified at the architectural phase can be found in the UML diagram as a method of a class providing an implementation for WSDL interface. That method is then annotated with the access control constraint and the annotation is linked to the authorization requirement (A), which propagated through the architectural design phase.

Implementation and deployment phases. The types of users that can access the system can be found in the code or configuration files. For example, user types can be implemented as separated classes or documented as set of possible roles that users can activate. These artifacts are then annotated with user types as specified at the requirements phase in the authentication requirement (B). The user type annotations are also linked to the SSO and the access control annotations from previous phases. These links are further used to configure the users in the SSO mechanisms and to write the policies that regulate the access control mechanisms.

4. Enforcement of annotation-based security constraints

Once that a multi-level annotation framework is in place, a way to enforce the constraints defined by means of this framework is still needed. This should take place at different levels of abstractions as well. At the composition level, for instance, once a security requirement for an architectural entity is defined (e.g., a secure channel constraint for a connector), the requirement could eventually be enforced by the middleware the architecture relies on. Unfortunately, most of the enforcement runtime environments operates only at the level of implementation and deployment phases. A common way to “enforce” annotations at the code level, as done in SAF, is to use Aspect Oriented Programming (AOP) techniques. AOP allows good separation of concerns similarly, in a way, to what annotations do at the conceptual level. However, with respect to security, further problems are to be
addressed. For example, when the execution flow of an application is intercepted and the woven AOP-based access control functionality denies an access request to some resource, the application faces an unexpected exception. In some cases the application might be unable to recover from such event and this has a clear impact on the overall reliability of the system. To this aim, the application should provide extra information that allows the enforcement runtime to preserve the application integrity, albeit preserving the transparent weaving of security controls. For instance, an annotation could mark the compensation procedure to be executed in case an access denial is encountered.

5. Discussion

As it was mentioned in Section 2, we believe that multi-level security annotations have the potential to become a viable alternative (or companion) to validation and verification methodologies for assurance, to improve the quality of security solutions, and to make the development and maintenance of secure applications easier. However, it is still early to draw strong conclusions since a thorough validation must be performed first, according to the following guidelines.

The multi-level security annotations framework aims at being general. However, in order to be applicable in practice it must provide annotations that are expressive enough in order to fit real-world cases. To this aim, we must assess the power of the proposed framework by comparing it to existing approaches. The newly defined framework must match and possibly exceed the expressiveness of the compared technologies. This validation is meant to provide an initial form of quality control.

However, in order to take the evaluation to the next level, the expressiveness of the framework must be tested over a set of realistic (i.e., non-trivial and adequately sized) case studies. Further, the framework must be compared to alternative approaches using domain specific languages (DSL) for security. Ponder [7] and xADL [8] are examples in the area of policy and architectural description languages, respectively.

Finally, we must validate the framework in order to assess to what extent it increases assurance. In this respect, we can compare the outcome of two case studies in which the same security-sensitive application is developed according to (1) the approach we suggest and (2) a well known secure development process like CLASP [5] or SDL [6]. We could assess and compare the needed effort, the overhead and the overall security of the resulting applications. This validation method is hard to perform, as it takes considerable effort to develop the two versions of an applications and perform security tests. However, the results of this approach would be convincing.

These and similar issues are the focus of our ongoing research.

References