Security Hardening UML Profile (SHP): A New Approach to Specify Security Hardening Solutions in UML

D. Mouheb, C. Talhi, A. Mourad, V. Lima, M. Debbabi, L. Wang
Computer Security Laboratory, Concordia University, Montreal, Canada

M. Pourzandi
Software Research, Ericsson Canada Inc., Montreal, Canada

Abstract

Security plays a predominant role in software engineering. Enforcing security policies should be considered during the early stages of the software development lifecycle to prevent security breaches in the final products. Because of the pervasive nature of security, integrating security solutions at the software design level may result in the scattering and tangling of security concerns throughout the entire design. To tackle this problem, we propose in this paper a new approach of representing and enforcing the hardening solutions based on aspect-orientation. This new approach provides software designers with UML-based capabilities to perform security hardening in a systematic and organized way, at the UML design level, without the need to have extensive security expertise. We also present SHP profile, a UML-based security hardening language to describe and specify security hardening solutions at the UML design level. Finally, we illustrate the applicability and the relevance of our approach through a real case study.

Keywords: Security Hardening, UML Design, Aspect-Oriented Programming, Aspect-Oriented Modeling.

1. Introduction

Software security is increasingly facing more challenges than ever before. In fact, the scale of security breaches have been increasing with no complete victory against attacks. The challenge is even greater when it comes to develop software systems that need to be deployed into high risk environments. Software security enforcement is generally conducted as an afterthought phase of the software development life cycle. However, this practice is no longer acceptable for such an important aspect, especially with the increasing complexity and pervasiveness of today’s software systems. As a result, there is a clear need for practical methodologies and tools that improve secure software development. This must include specifying security requirements and enforcing them systematically on software.

During the last years, several proposals emerged to integrate security during the early phases of the software development process using the Unified Modeling Language (UML) [12]. For instance, the security engineering proposals provide security design patterns as guide to improve and integrate security during the design phase. They approach the problem by encapsulating experts knowledge in the form of well-defined UML solutions to common security problems. However, so far, these approaches miss the methodologies for enforcing such requirements and integrating them systematically into software. In most cases, applying their solutions requires high security expertise, which contradicts somehow with their claimed goal. This also leads to another problem consisting in the difficulty of finding developers who are experts in both the software functionalities and security domains. In addition, because of the pervasive nature of security, if designers add security solutions manually into a UML design, security components may become tangled and scattered throughout the whole UML design. Consequently, the resulting UML model will most likely become difficult to understand and maintain. Moreover, adding security manually is tedious and generally may lead to other security vulnerabilities.

To address this issue, Aspect-Oriented Modeling (AOM) [2] is a potential solution to enforce systematically security requirements at the early stages of software development. In fact, aspect-oriented techniques allow separating cross-cutting concerns, such as security, from the application functionalities, and then rely on underlying infrastructure to weave them together. Using AOM, security solutions can be precisely defined and injected at the matched places. However, to date, there is no standard language to support AOM. The existing AOM proposals are mainly programming-language dependent and lack many features needed for systematic security hardening specification.
In this paper, we propose an aspect-oriented approach for systematic security hardening of software at the design level. The main components of our approach are the security hardening plans and patterns that together constitute concrete security hardening solutions. The patterns provide high-level aspect-oriented solutions to known security problems. The plans capture the needed security requirements and use the appropriate patterns to harden the security of an application. Both plans and patterns are specified at the design level using the Security Hardening UML Profile (SHP) that is provided by our approach. Once the security solutions are specified for a given UML design, developers can systematically translate them into security aspects code without the need of having an extensive security knowledge.

The proposed approach is well suited for job separation. Security experts are responsible for providing high-level security solutions including the details on how and where to apply them in the application. At the same time, developers who are responsible for designing the software functionalities, can use those solutions to enforce systematically security requirements during the early stages of the software development. Using our approach, developers do not need to modify the security solutions provided by security experts. Even developers with limited security knowledge can use those solutions to enforce the needed security requirements in a systematic way.

The rest of this paper is structured as follows. Section 2 summarizes our approach for software security hardening. Then, we present the SHP profile in Section 3. Section 4 illustrates the approach through a case study. An overview of the related work is given in Section 5. Finally, we conclude the paper and present our future work in Section 6.

2. Approach Overview

This section presents an overview of our approach for security hardening of UML design. By security hardening of UML design, we mean leveraging the UML design models to add security functionalities and/or fix vulnerabilities in the software design. The approach architecture is depicted in Figure 1. The main components of our proposed approach are the following:

- **UML-Based Security Hardening Patterns**: They provide high-level aspect-oriented security solutions including the detailed information on how and where to inject each solution into the application. Patterns are developed by security experts with the help of security APIs and provided in a catalog. AOM [2] is adopted for the specification of patterns to keep the security solutions separated from the software functionalities.

- **UML-Based Security Hardening Plans**: They capture the needed security requirements for an application.

Figure 1. Security Hardening Approach.

From the patterns library developed by security experts, the developer selects the appropriate pattern(s) using the patterns parameters that provide information about the security solution, such as, the security protocol and the API used in the solution. Then, the developer specifies a plan by using the selected patterns. The developer does not need to modify the security solutions provided in the patterns. An example of using plans and patterns will be illustrated in Section 4.

- **Security Hardening UML Profile (SHP)**: It is a UML-based security hardening language used to specify plans and patterns at the design level. It is high-level, aspect-oriented, and independent of any programming language. The SHP profile provides an abstraction over the existing aspect-oriented languages to allow security hardening specification at the UML design level. The SHP profile is detailed in Section 3.

- **Security Hardening Aspects**: The developer systematically refines the high-level solutions provided by plans and patterns into security hardening aspects. The security hardening aspects are implemented using AOP languages. Afterwards, AOP weavers (e.g., AspectJ, AspectC++) can be executed to integrate the aspects into the software source code previously generated from the software design models.

The advantages of our approach are as follows:

- Any developer can specify a plan without having to understand the inner working of the security solutions. All that he/she needs to know is the existence of patterns that enforce the needed security requirements.

- Plans are totally separated from the application design models. Developers do not need to modify their design to specify security requirements.
• Plans can be added dynamically to encapsulate other security requirements.

• Patterns provide aspect-oriented solutions. They give the precise steps to be performed and the location where they should be applied. This maintains the separation of security from the design functionalities.

• Patterns provide high-level solutions. Their syntax is independent of any programming language. Thus, patterns can be refined and reused for various applications that are developed under different environments.

• Patterns can be easily refined into security aspects. Indeed, patterns provide solutions with enough details so that developers can translate them into the target language without the need to have any security expertise.

3. Security Hardening Profile (SHP)

This section presents our elaborated SHP profile that allows the description and specification of security hardening solutions for UML design. To harden security at UML design, three main approaches can be followed:

1. Creating a UML profile using the extension mechanisms provided by standard UML (stereotypes and tagged values) [12].

2. Extending the UML meta-language by new language constructs that allow the specification of security requirements.

3. Defining a new meta-language to specify security requirements.

Among these approaches, UML profiles seem to be the most usable for security specification since they are the extension mechanism provided by the standard UML. They allow the specification of almost all the security requirements that are usually specified and enforced on software. In addition, they are easy to learn and use and benefit from excellent tool support and high portability.

The proposed SHP profile provides the necessary elements needed for specifying plans and patterns at the UML design level. It is based on AOM to separate security concerns from the software functionalities. In the following, we present the meta-model specification of the SHP profile. We first present the meta-elements needed for specifying plans. Then, we give the meta-model for specifying patterns. The meta-elements are translated into UML stereotypes that extend the existing UML meta-elements [12]. The attributes of the meta-elements will become tags of stereotypes when the profile is applied [12]. The values of the attributes will be considered as tagged values of the stereotypes [12].

3.1. Specification of Security Hardening Plans

The meta-model for specifying security hardening plans is depicted in Figure 2. It provides developers with the necessary elements to specify security requirements for UML design, specifically, it shows how to represent a plan, the patterns that are used by the plan, and how to apply a plan to the base model. It also shows the relationship between the new meta-elements and the UML meta-elements (represented with a stereotype ≪metaclasse≫). In the following, we explain the semantics of each meta-element.

Plan: The meta-element Plan represents a security hardening plan. It extends the UML meta-element Class. As stated before, a plan encapsulates a set of security requirements that need to be enforced in a software. In our profile specification, plans can be applied to UML packages through an association presented by the meta-element Plan_Application. We also defined an inheritance relation between plans. This allows a plan to inherit the security solutions from another plan.

Plan_Application: The meta-element Plan_Application extends the UML meta-element Dependency. It denotes a relationship between a plan and a UML package. Since a plan can specify many security requirements that may involve various UML classes, it is useful to apply a plan to the UML package(s) containing those classes rather than applying it to each class. This way of applying plans will compact as possible the applied security solutions and therefore, reduce the complexity of the plans specification.

Pattern_Instatiation: The meta-element Pattern_Instatiation specifies the patterns used in a plan. It extends the UML meta-element StructuralFeature. In the UML meta-model, structural features are owned by classes. Thus, there is no need to associate Pattern_Instatiation with Plan. For a pattern to be instantiated, we defined some attributes that represent the pattern’s parameters (e.g., Language, API), in addition to the location where the pattern should be
applied (the attribute Where). We added the following OCL: [11] constraint to ensure that the stereotype «Pattern_Instatiation» can be applied only to features of classes that are stereotyped «Plan»:

context Pattern_Instatiation inv:
  allInstances.featuredClassifier.oclIsKindOf(Plan)

**Pattern_Instance:** The meta-element Pattern_Instance represents pre-defined patterns that give the security solutions for well-known application-independent vulnerabilities such as buffer overflow, SQL injection, etc. The stereotypes representing those patterns can be applied directly either on: (1) a UML class if the security problems are present in only one or a small number of classes, or (2) a UML package if the security problems are present in many classes of the same package. In this case, an attribute Where is used to specify to which classes the pattern should be applied. This solution helps keeping the design as simple as possible by reducing the number of security elements involved in the design.

### 3.2. Specification of Security Hardening Patterns

Figure 3 presents the meta-model proposed for the specification of security hardening patterns. This meta-model is based on aspect-orientation to allow the separation of security concerns from the software functionalities. The elements of this meta-model are used by security experts to specify security hardening solutions for well-known security problems. Those elements mainly represent patterns, the security components they embody, and the location where they should be injected. The relationship between the new meta-elements and the UML meta-features of classes that are stereotyped «metaclass» is also shown in the figure. In the following, we explain the semantics of each meta-element.

![Figure 3. The Meta-Model for Specifying Security Hardening Patterns.](image)

**Pattern:** The meta-element Pattern represents a security hardening pattern. It extends the UML meta-element Class. As stated earlier, a pattern specifies the precise steps of the hardening and the locations where the hardening should be performed. The meta-element Pattern contains some attributes (e.g., Protocol) that represent the pattern's parameters that could help in distinguishing the patterns with similar names and allow pattern instantiation. A pattern is composed of one or many Advice elements that indicate the security methods and the insertion points where they should be injected.

**Advice:** The meta-element Advice represents the security behavior. It extends the UML meta-element BehavioralFeature. The advice signature is represented as a stereotyped UML operation, whereas the advice behavior is specified in behavioral diagrams. The type of the advice is represented by the attribute type whose values are provided in the enumeration Advice_Type. We added two other attributes to the meta-element Advice that are exportParameter and importParameter. These two primitives are introduced in [10] to specify the parameters that should be passed between two Pointcut constructs. We added the following OCL constraint to ensure that the stereotype «Advice» can be applied only to features of classes that are stereotyped «Pattern»:

context Advice inv:
  allInstances.featuredClassifier.oclIsKindOf(Pattern)

**Pointcut:** The meta-element Pointcut specifies particular points in the base model where the security behavior should be applied. It extends the UML meta-element StructuralFeature since it does not specify a dynamic behavior. We defined the following OCL constraint to ensure that «Pointcut» can be applied only to features of classes that are stereotyped «Pattern»:

context Pointcut inv:
  allInstances.featuredClassifier.oclIsKindOf(Pattern)

We specialized the meta-element Pointcut to represent different kinds of pointcuts, mainly those that are commonly used by different AOP languages:

- **Operation_Pointcut:** describes pointcuts that select operation-based join points. It has a multi-valued attribute (operation) that contains the captured operations. The type of the join point is provided by the subclasses of this meta-element (Call_Pointcut and Execution_Pointcut).
- **CFlow_Pointcut:** captures join points occurring in the dynamic execution context of the join points specified in the attribute pointcut.
- **Context_Pointcut:** captures join points exposing context information such as the argument values of a function call.
4. Case Study

To demonstrate the utility of our approach, we present in this section a case study related to securing the HTTP connections of an open-source software called APT (Advanced Package Tool). We first present the security hardening solutions specified at the UML design level, then we provide the aspect code implemented using AspectC++ together with the experimental results. APT is an automated package downloader and manager for the Debian Linux distribution. It is written in C++ and is composed of more than 23,000 source lines of code (based on version 0.5.28, generated using David A. Wheeler’s ‘SLOCCount’). In the sequel, we present the plan, the pattern, and the aspect elaborated to secure the HTTP connections of APT.

4.1. UML-Based Security Hardening Plan

The UML diagram presented in Figure 4 depicts the APT package that contains APT acquire methods, such as HTTP, FTP, etc. In the following, we explain how the developer can add HTTPS support to APT using the plan Secure_Connection_Plan. The plan Secure_Connection_Plan is represented as a UML class stereotyped <<Plan>>. The plan is applied to the APT package using a stereotyped association <<Plan_Application>>. The developer specifies the plan using the Secure_Connection_Pattern. The Secure_Connection_Pattern is represented by an attribute stereotyped <<Pattern_Instantiation>>. Tags are attached to the stereotype <<Pattern_Instantiation>> to allow the identification of the selected pattern. The location where the pattern should be applied is also specified by a tag Where. In this example, the pattern should be applied to the classes Connection, ServerState, HTTPMethod and CircleBuf that are affected when establishing HTTP connections.

4.2. UML-Based Security Hardening Pattern

Figure 5 presents the solution part of the Secure_Connection_Pattern for securing the HTTP connections of the aforementioned application using GnUTLS/SSL. The security expert specifies the pattern as an aspect, therefore at the code generation, the security methods provided by the pattern will be injected into the source code. The pattern is represented as a UML class stereotyped <<Pattern>>. The tags attached to this stereotype specify the pattern’s parameters. The pattern contains a set of security methods and the locations where they should be applied. The security methods are represented as operations stereotyped <<Execution_Pointcut>>.

4.3. Security Hardening Aspect

We realized the UML-based security hardening solution by implementing its corresponding aspect using AspectC++. Due to space limitation, Listing 1 shows only an excerpt of the aspect, specifically the handshake code around the function connect().

Figure 4. The APT Package Diagram with Secure Connection Plan.

Figure 5. The Secure Connection Pattern.
inserted after the function `connect`. The reader will notice the appearance of `hardening_sockinfo_t`. This is the data structure that we developed to distinguish between secure and non-secure channels and export the parameters between the application’s components at runtime (since the primitives `ImportParameter` and `ExportParameter` are not yet deployed into the weavers).

![Figure 6. Behaviors of the Advices InitializeTLS and AddTLSHandshake.](image)

Listing 1. Excerpt of Aspect for Securing Connections

```java
aspect SecureConnection {

    advice call("%connect(...)") : around () {
        hardening_sockinfo_t socketInfo;
        ...
        tjp->proceed(); //original connect
        ...
        //TLS handshake
        gnutls_transport_set_ptr (socketInfo.session, (gnutls_transport_ptr) (*((int*)tjp->arg(0))));
        int result = gnutls_handshake (socketInfo.session);
    }
}
```

4.4. Experimental Results

In order to validate the hardened applications, we used the Debian apache-ssl package, an HTTP server that accepts only SSL-enabled connections. We populated the server with a software repository compliant with APT’s requirements, so that APT can connect automatically to the server and download the needed metadata in the repository. Then, we weaved (using AspectC++ weaver) the elaborated aspect with the different variants of APT. The resulting hardened software was capable of performing both HTTP and HTTPS package acquisition, based on the parameters in the configuration file. After building and deploying the modified APT package, we tested successfully its functionality by refreshing APT’s package database, which forced the software to connect to both our local web server (Apache-ssl) using HTTPS and remote servers using HTTP to update its list of packages. The experimental results in Figures 7 and 8 show that the new hardened software is able to connect using both HTTP and HTTPS connections, exploring the correctness of the security hardening process. In the sequel, we provide brief explanations of our results. Figure 7 shows the packet capture, obtained using WireShark software, of the unencrypted HTTP traffic between our version of APT and its remote package repositories. The highlighted line shows an HTTP connection to the `www.getautomatix.com` APT package repository. On the other hand, Figure 8 shows the connections between our version of APT and the remote package repositories on the local web server. The highlighted lines show TLSv1 application data exchanged in encrypted form through HTTPS connections.

![Figure 7. Packet Capture of Unencrypted APT Traffic](image)

![Figure 8. Packet Capture of SSL-protected APT Traffic](image)

5. Related Work

Few approaches aiming to enforce security requirements at the UML design level have been published recently. [13] proposes a new UML artifact called Role Slice to capture access control policies at the design level. Then, access control policies are mapped to AOP security enforcement.
code implemented in AspectJ. [7] defines a general metamodel for generating security modeling languages. SecureUML [7] is one instance of these languages defined for modeling access control requirements. However, these approaches enforce only access control policies.

Other approaches [1, 6, 8, 14] have been proposed in the last years to integrate security during the early phases of software development using UML. The majority of them propose extensions of the UML language using standard UML extension mechanisms to specify security requirements. The evaluation of UML models against the specified requirements is based on automatic verification tools such as model checkers and theorem provers [5]. More details about these contributions can be found in [9]. Although these approaches are useful attempts for specifying and verifying security requirements on UML design, only few of them generate code for new software. Security hardening is not their main concern as our approach.

Another field that is related to our work is security design patterns. Romanosky presents a set of design patterns in [15]. The discussion however has focused on architectural and procedural guidelines more than on security patterns. [4] introduced a design pattern which describes a general mechanism for providing authentication and authorization. However, the pattern is limited to distributed object systems. The Open Group [3] has possibly introduced the most mature design pattern catalog of 13 patterns. Although security design patterns provide reusable solutions to integrate security early during the development process, they generally lack the structure and the methodologies needed for their application. Most of them are described as directions written in English. In addition, they are applied manually and require high security expertise.

6. Conclusion and Future Work

In this paper, we presented a new approach for software security hardening. Through the case study, we showed how SHP enhances knowledge transfer and reuse by capturing and packaging security hardening solutions and providing those solutions to designers with limited security knowledge. Security expertise is then provided as high-level solutions including the details on how and where to apply them in the application. The approach provides the UML-based capabilities required for both, capturing security solutions and using them. These capabilities are provided by the SHP profile, which extends the UML language with aspect-oriented specification capabilities. A security expert can use the SHP profile to design security solutions; as patterns. Then, the developers can refine the security solutions to tune the security hardening by specifying the needed security requirements as plans. Moreover, the specified security solutions are translated into security aspects code and injected into the application. As a result of our contribution, security solutions can be integrated into the software from the early phases of the development life cycle. This in turn helps accelerating the development of secure applications and reducing errors. Currently, we are investigating the weaving mechanisms at the design level to generate secure UML models and secure code. The scalability and maintainability of the approach with respect to the complexity of the software will be addressed in the future.

References