An Aspect-Oriented Approach for Software Security Hardening: from Design to Implementation[1]

Djedjiga MOUHEB [a], Chamseddine TALHI [a], Azzam MOURAD [a], Vitor LIMA [a], Mourad DEBBABI [a], Lingyu WANG [a] and Makan POURZANDI [b]

[a] *Computer Security Laboratory, Concordia University, Montreal, Canada*
[b] *Software Research, Ericsson Canada Inc., Montreal, Canada*

Abstract. Security is a very challenging task in software engineering. Enforcing security policies should be taken care of during the early phases of the software development life cycle to prevent security breaches in the final product. Since security is a crosscutting concern that pervades the entire software, integrating security solutions at the software design level may result in scattering and tangling security features throughout the entire design. To address this issue, we propose in this paper an aspect-oriented approach for specifying and enforcing security hardening solutions. This approach provides software designers with UML-based capabilities to perform security hardening in a clear and organized way, at the UML design level, without the need to be security experts. We also present the SHP profile, a UML-based security hardening language to describe and specify security hardening solutions at the UML design level. Finally, we explore the efficiency and the relevance of our approach by applying it to a real world case study and present the experimental results.

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

Introduction

Security is a major challenge in the software development process. In fact, the scale of security breaches have been increasing with no complete victory against attacks. The challenge is even greater with the increasing complexity and pervasiveness of modern 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 on software.

During the last years, some new methodologies have been proposed to incorporate security requirements during the early phases of software development using the Unified Modeling Language (UML) [33] since it is the de facto language for software specification and design. However, most of these approaches concentrate their efforts on speci-

[1]The research leading to this work was possible due to funding and scientific collaboration with Software Research, Ericsson Canada Inc.

fying security requirements and sometimes analyzing UML models against the specified requirements. So far, complete, practical, and efficient methods and tools are still largely absent for enforcing such requirements on software. In most cases, this task, that requires high security expertise, is conducted as an afterthought by developers who are not, in general, security experts. As a result, security flaws remain abundant and hard to fix in the software products.

In addition, because of the pervasive nature of security, if the 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 design model will most likely become difficult to understand and maintain. Additionally, adding security manually is tedious and generally may lead to other security vulnerabilities.

To tackle this problem, Aspect-Oriented Modeling (AOM) [4] is a natural and appealing solution to enforce 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 by introducing new modules called `aspects` that capture only one concern. The integration of aspects with the rest of the application is performed by a mechanism called `aspect weaving`. Using AOM, security solutions can be precisely defined and injected in the right places without altering the original functionalities of the software. However, to date, there is no standard language to support AOM. Even the existing AOM languages are not suitable for security hardening specification. They are mainly programming-language dependent and lack many features needed for systematic security hardening.

In this context, the objective of this paper is to elaborate an aspect-oriented approach for software security hardening at the UML 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. They are developed by security experts and provided in a catalog. The plans capture the needed security requirements and use the appropriate patterns to harden the security of an application. In other words, plans specify the needed security requirements and patterns specify how those requirements can be enforced. Both plans and patterns are specified at the design level using the Security Hardening UML Profile (SHP) that is provided by our approach. Once security hardening solutions are specified for a given UML design, developers can systematically translate those solutions into security aspects code.

Our approach has significant benefits. It allows security experts to provide high-level security solutions including all the details on how and where to apply them. At the same time, developers can use those solutions to enforce security requirements during the early stages of the software development without the need to have any extensive security expertise. Moreover, the use of aspect-oriented techniques allows the specification of security solutions in a clear and organized way, without altering the original software functionalities. The experimental results that we obtained after applying our solutions illustrate the relevance and efficiency of our approach.

The rest of this paper is structured as follows. Section 1 gives an overview of the current state of the art related to security specification and enforcement using UML. Afterwards, in Section 2, we summarize our approach for software security hardening. Then, we present the SHP profile in Section 3. Section 4 illustrates the usability of the approach through a case study. Finally, we conclude the paper and present our future work in Section 5.

1. Related Work

Few approaches aiming to enforce security requirements at the UML design level have been published recently [25, 36]. Pavlich-Mariscal *et al.* [36] propose a new UML artifact called `Role Slice` to capture RBAC (Role-Based Access Control) [16] policies within UML class models. A role slice diagram contains information on a role's permissions that cut across all classes in an application. RBAC constraints are represented within a role slice diagram using UML stereotypes [33]. Moreover, this approach proposes algorithms that map access control policies provided in role slice diagrams to AOP [23] security enforcement code implemented in AspectJ [22]. In another effort [37], the authors propose an aspect-oriented approach to model access control policies. They augment the UML meta-model with new diagrams that are separated from the main UML design to represent RBAC (Role-Based Access Control) [16], MAC (Mandatory Access Control) [7] and DAC (Discretionary Access Control) [30] models. The separated security features are then composed with the main UML design. In another effort [37], the authors propose an aspect-oriented approach where they augment the UML meta-model with new diagrams to represent access control policies. In contrast to our approach that uses a UML profile to represent security requirements, Pavlich-Mariscal *et al.* extend the UML language by defining new artifacts to capture only access control policies.

Another contribution that is related to security enforcement is given in [25] which is called Model-Driven Security. This approach proposes a general schema in which systems modeling languages such as UML are combined with security modeling languages by defining `dialects`. The latter identify the protected resources from elements of the system modeling language. This approach defines a general meta-model for generating security modeling languages. SecureUML [24, 25] is one instance of these languages defined for modeling RBAC requirements [16]. SecureUML has an abstract syntax that is independent of any modeling language, and a concrete syntax that is defined as a UML extension using stereotypes and tagged values [33]. From models in the combined languages, access control infrastructures are automatically generated using MDA-based transformation mechanisms [31].

Other approaches [1, 3, 9, 11, 12, 14, 20, 26, 35, 39, 48] 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 most covered security requirements are access control policies [7, 16, 30]. The evaluation of UML models against the specified security requirements is based on automatic verification tools such as model checkers and theorem provers [18]. More details about these contributions as well as a

comparative study on them can be found in [28]. 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. A security pattern describes a particular recurring security problem that arises in a specific context and presents a well-proven generic scheme for a security solution [43]. A number of security design patterns are proposed in order to guide software engineers in designing their security models during the architecture and design phases. Romanosky presents a set of design patterns in [41]. The discussion however has focused on architectural and procedural guidelines more than on security patterns. Fernandez and Warrier [15] introduced a design pattern which describes a general mechanism for providing authentication and authorization. The Open Group [8] has possibly introduced the most mature design pattern catalog that contains 13 patterns. Although security design patterns provide reusable solutions to integrate security early during the development process, they have some shortcomings. They generally lack the structure and the methodologies needed for their application. In addition, they are applied manually and require high security expertise.

Regarding aspect-oriented modeling using UML, considerable work has been done in the literature during the last years. An overview of some of the proposed contributions is presented in [40, 42]. Prior work in this area [2, 5, 6, 17, 21, 27, 38, 45] proposes extensions of the UML language using standard UML extension mechanisms. However, the majority of these approaches are programming language dependent and specify only few concepts of AOP. Some other proposals [10, 47] extend the UML meta-language by defining new meta-classes for AOP instead of extending the existing UML meta-classes. However, extending the UML meta-language with new meta-classes is hard to implement and may require modifying the existing UML CASE tools to support the new meta-classes. The most recent work in this domain is presented by Evermann in [13]. This approach proposes a specification of AOP concepts based on the existing UML meta-model using standard UML extension mechanisms. In contrast to previous work, this is possibly the most complete specification so far. The proposed profile covers most of AOP concepts. However, it is based on AspectJ. Thus, it cannot be used to specify our hardening solutions that are independent of any programming language.

2. Approach Overview

This section illustrates our approach for software security hardening. By security hardening, we mean adding security functionalities and/or fixing vulnerabilities in the software. The approach architecture is depicted in Figure 1. The main steps of our proposed approach are the following:

- Specification of Security Hardening Solutions: Security experts, with the help of security APIs, design UML-based solutions to known security problems. Those solutions are provided as Security Hardening Patterns. AOM [4] is adopted for the specification of security hardening patterns to keep security solutions separated from the software functionalities. Security hardening patterns are specified using

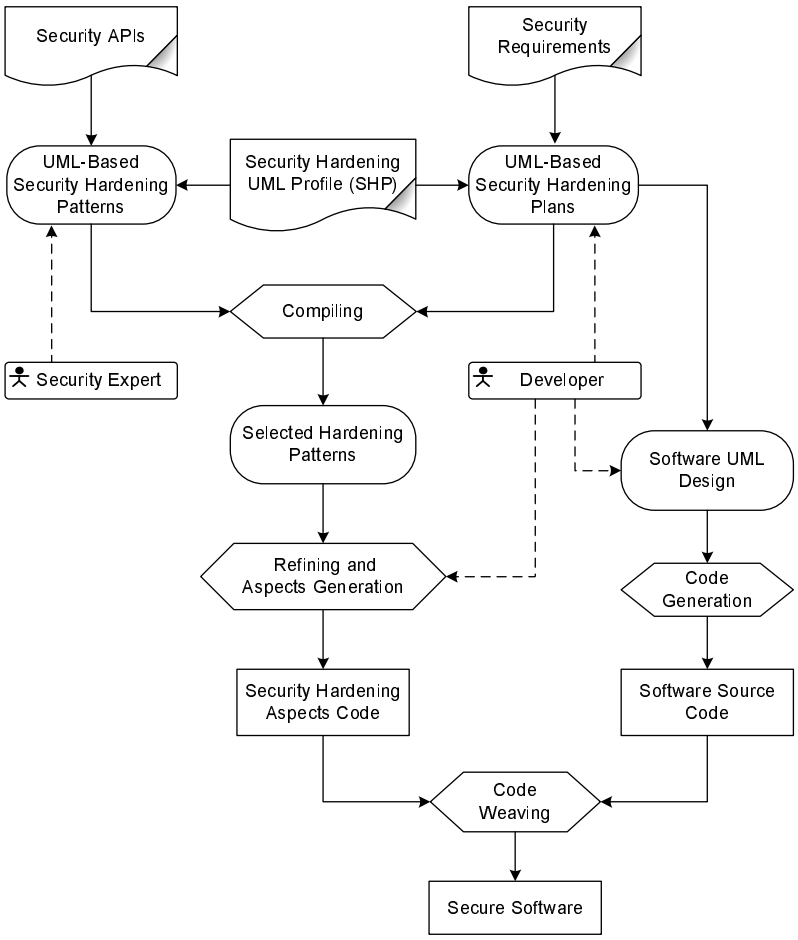

Figure 1. Security Hardening Approach.

the SHP profile provided by our approach (See Section 3.2). Once designed, patterns are packaged in a catalog and presented to designers with limited security knowledge.

- Specification of Security Requirements: Developers, when designing the application, specify the needed security requirements by writing a Security Hardening Plan. From the catalog of patterns developed by security experts, developers select 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, developers specify a plan by using the selected patterns. Developers do not need to modify the security solutions provided by the patterns. The SHP profile is also used for the specification of the plans (See Section 3.1).
- Selection of Actual Hardening Patterns: From the security hardening plans specified by developers and the catalog of patterns provided by security experts, the actual patterns needed to enforce the specified security requirements are selected.

The selection of the appropriate patterns is based on the patterns parameters specified in the hardening plans.

- Refinement and Aspects Generation: Once the needed patterns are selected, developers systematically refine the solutions provided by the patterns and implement the corresponding security hardening aspects. The security hardening aspects are implemented using AOP languages. Using our approach, developers do not need to learn the security APIs to write security aspects since developers are guided by the patterns.
- Weaving of Security Aspects into Software Code: The actual security hardening is performed by weaving the security aspects into the software source code previously generated from the software design models. AOP weavers (e.g., AspectJ, AspectC++) are used to inject the involved advices into the matched places.

The advantages of our approach are as follows:

- Any developer can specify a plan without having the need 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 base 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 in the base models 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 platforms and environments.
- Patterns can be easily refined into security hardening 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) [33].
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 on UML models.

Our choice of the first approach is based on our previous usability study of the mentioned approaches [46]. In fact, 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 spec-

ified 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 presented in Section 2 at the UML design level. It is based on AOM to separate security concerns from the software functionalities. The main features provided by the SHP profile are the following:

- Description and specification of security behavior. This is done by the addition of new behavior before or after the application behaviors, and substitution of an existing behavior by a new one.
- Specification of particular locations in the base model (join points) where the security behavior will be injected.
- Specification of patterns parameters to allow their instantiation by the plans.
- Specification of security solutions in an organized way and without altering the original software functionalities.
- Description and specification of reusable solutions.
- Programming-language independency.
- Easiness of understanding and use by non security experts.

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 [33]. The attributes of the meta-elements will become tags of stereotypes when the profile is applied [33]. The values of the attributes will be considered as tagged values of the stereotypes [33].

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 and instantiated 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 ≪`metaclass`≫). 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. The security solutions enforcing these requirements are implemented in patterns used by the plans. 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 the developer to reuse existing plans instead of creating a new plan for each security breach. Thus, a plan can inherit the security solutions from another plan. For example, as shown in Figure 3, if a plan B extends a plan A, then the plan B will provide all the security solutions provided by the plan A, i.e., `Pattern_A1` and `Pattern_A2`, in addition to those solutions that are specific to the plan B, i.e., `Pattern_B`.

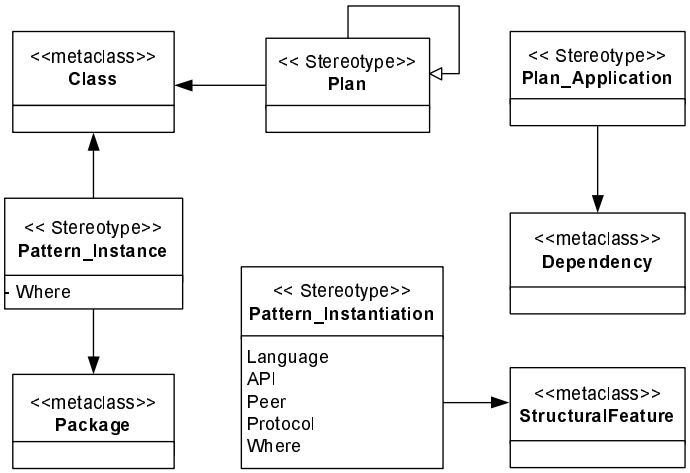

Figure 2. The Meta-Model for Specifying Security Hardening Plans.

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_Instantiation: The meta-element `Pattern_Instantiation` 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_Instantiation` with `Plan`. For a pattern to be instantiated, we defined for the meta-element `Pattern_Instantiation` 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 constraint [32] to ensure that the stereotype ≪`Pattern_Instantiation`≫ can be applied only to features of classes that are stereotyped ≪`Plan`≫:

```
context Pattern_Instantiation 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.

To better illustrate the usability of the SHP profile for specifying plans, an example is given in Figure 3. It shows the different possibilities of applying hardening solutions. For instance, `Plan_A` uses two patterns that are `Pattern_A1` and `Pattern_A2`. `Plan_B` uses `Pattern_B`. In addition, since `Plan_B` extends `Plan_A`, `Plan_B` inherits the patterns instantiated by `Plan_A` (`Pattern_A1` and `Pattern_A2`). The figure also shows how patterns can be applied directly as stereotypes either on UML packages (e.g., `Pattern_1`), or UML classes (e.g., `Pattern_2`). The patterns' parameters are not shown in this figure. Please refer to Figure 5 for an example of their use.

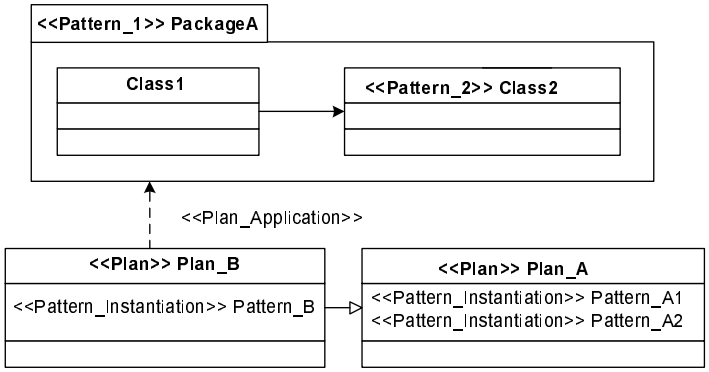

Figure 3. Different Possibilities of Applying Security Hardening Solutions.

3.2. Specification of Security Hardening Patterns

Figure 4 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. They mainly represent patterns, the security components they embody, and the location where each component should be injected in the base models. The relationship between the new meta-elements and the UML meta-elements (represented with a stereotype ≪`metaclass`≫) is also shown in the figure. In the following, we explain the semantics of each meta-element.

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 solutions should be injected. The meta-element `Pattern` contains some attributes (e.g., Language, API) that represent the pattern's parameters that could help in distinguishing the patterns with similar names and allow the pattern instantiation. A pattern is composed of one or many `Advice` elements that indicate the security behavior and the insertion point where it should be injected.

Advice: The meta-element `Advice` represents the security behavior to be integrated into the base model. It extends the UML meta-element `BehavioralFeature` since it

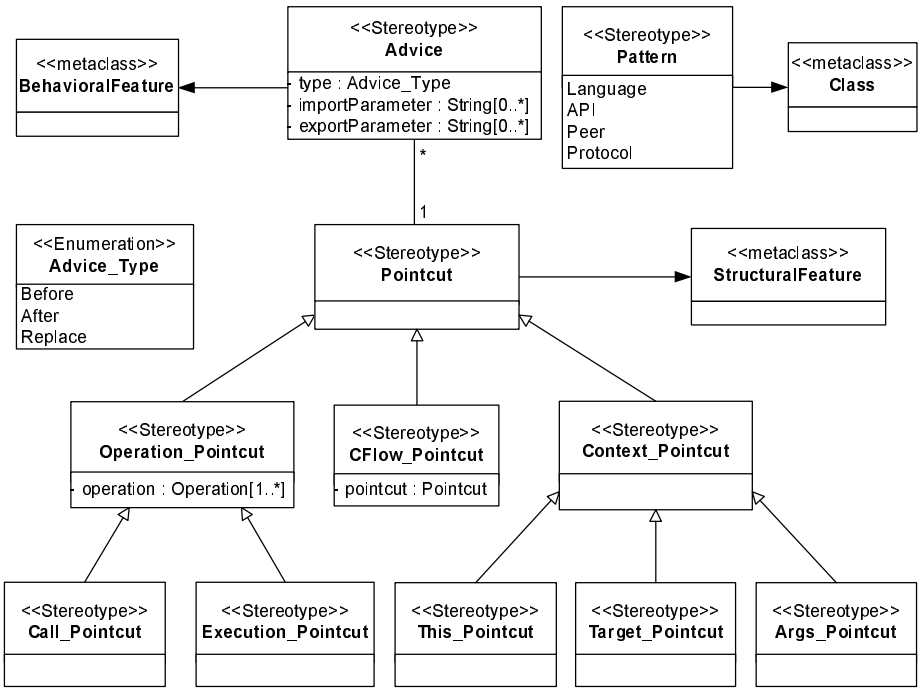

Figure 4. The Meta-Model for Specifying Security Hardening Patterns.

specifies a behavior. The advice signature will be represented as an operation in a UML class diagram. The advice behavior will be specified in behavioral diagrams, such as activity, sequence and state machines. The meta-element `Advice` is an abstract representation of an advice in the advice-pointcut model. Since behavioral features are owned by classes in the UML meta-model, there is no need to associate the meta-element `Advice` with the meta-element `Pattern`.

The location where the security behavior should be injected is specified by the attribute `type` and the meta-element `Pointcut`. The values of the attribute `type` are provided in the enumeration `Advice_Type`. We added two other attributes to the meta-element `Advice` that are `exportParameter` and `importParameter`. These two concepts are introduced in [29] to specify the parameters that should be passed between two `Pointcut` constructs. The attribute `exportParameter` is defined at the origin `Pointcut`. It allows to specify a set of variables and make them available to be exported [29]. The attribute `importParameter` is defined at the destination `Pointcut`. It allows to specify a set of variables and import them from the origin `Pointcut` where the `exportParameter` has been defined [29].

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 specified in the `Advice` should be applied. It extends the UML meta-element `StructuralFeature` since it does not specify a dynamic behavior. The meta-element `Pointcut` is an abstract representation of a pointcut in the advice-pointcut model. As for `Advice`, 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 (See [22, 23] for more details about the different kinds of pointcuts):

- `Operation_Pointcut`: describes pointcuts that select operation-based join points such as an operation call or execution. It has a multi-valued attribute (`operation`) that contains the captured operations. The type of the join point (e.g., call, execution) is provided by the meta-elements `Call_Pointcut` and `Execution_pointcut` that are sub-classes of the meta-element `Operation_Pointcut`.
- `CFlow_Pointcut`: describes pointcuts that select join points occurring in the dynamic execution context of the join points specified in the attribute `pointcut`.
- `Context_Pointcut`: specifies pointcuts that select join points exposing context information such as the argument values of a function call. It has three subclasses: `This_Pointcut`, `Target_Pointcut` and `Args_Pointcut` that specify the type of the context-based pointcut.

4. Case Study: Securing Connection of Client Applications from Design to Code

To demonstrate the feasibility and 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). The security hardening profile, the UML-based security hardening plan and pattern, and the base model presented in this section have been specified using IBM Rational Software Architect [19]. This tool comes with a powerful UML modeler which allows developers to design UML 2.0 models and profiles. In addition, it supports XMI-based model interchange mechanism [34] (in other words, it can import and export XMI files). Since XMI files are supported by most of UML tools, the examples shown below are then compatible with most of UML tools. Following the approach presented in Section 2, we first present the security hardening solutions specified at the UML design level. Then we provide the aspect code implemented using AspectC++ [44] 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'). It obtains packages via local file storage, FTP, HTTP, etc. In the sequel, we are going to present the security hardening plan, pattern, and the aspect elaborated to secure the APT connections.

4.1. UML-Based Security Hardening Plan

The UML diagram presented hereafter depicts the APT package that contains some APT acquire methods, such as HTTP, FTP, RSH methods, etc. We have decided to add HTTPS support to this application using the plan `Secure_Connection_Plan`. This package contains all the classes that are involved in the HTTP connection. The class `HTTPMethod` is responsible for sending the HTTP request. The class `ServerState` provides methods to open/close a connection to the server. The connection is executed by the method `connect()` in the class `Connection`. The class `CircleBuf` performs reading/writing the request/response from/into buffers.

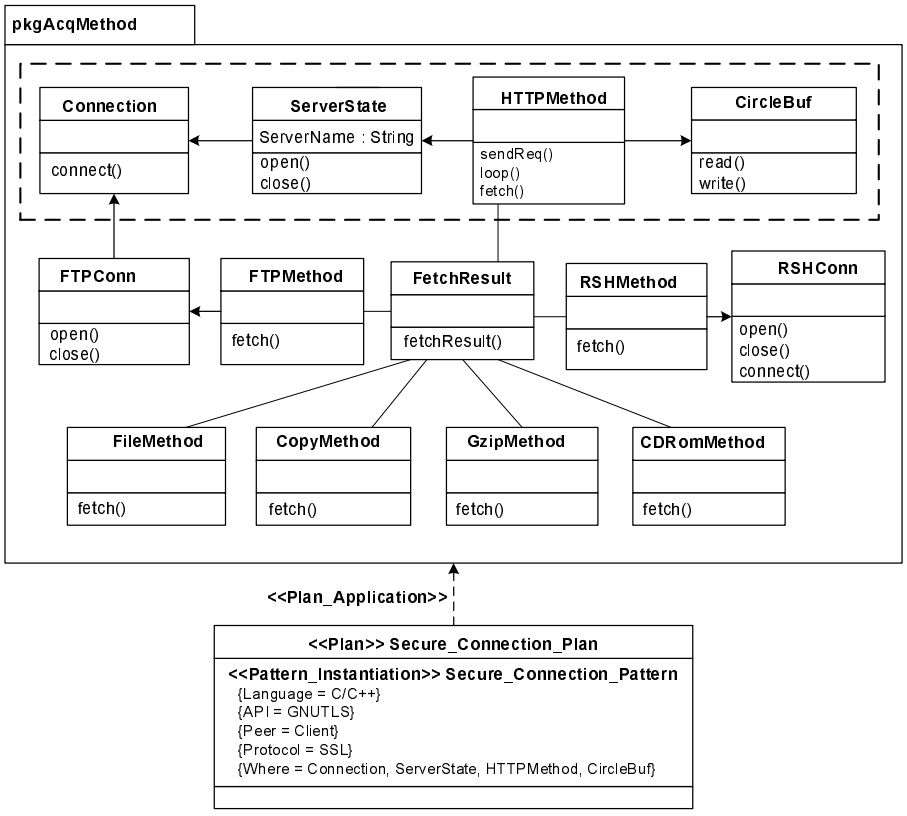

Figure 5. The APT Package Diagram with Secure Connection Plan.

The plan `Secure_Connection_Plan` is represented by a UML class stereotyped ≪`Plan`≫. The plan is applied to the APT package using a stereotyped association ≪Plan_Application≫. It uses `Secure_Connection_Pattern` that is represented by an attribute stereotyped ≪`Pattern_Instantiation`≫. As explained in Section 3, some tags (e.g., Language, API) are attached to the stereotype ≪`Pattern_Instantiation`≫ to allow the identification and instantiation of the selected pattern. The location (the classes) where the pattern should be applied is also

specified by a tag `Where`. In this example, the secure connection 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 6 presents the solution part of the `Secure_Connection_Pattern` for securing the connections of the aforementioned application using GnuTLS/SSL.

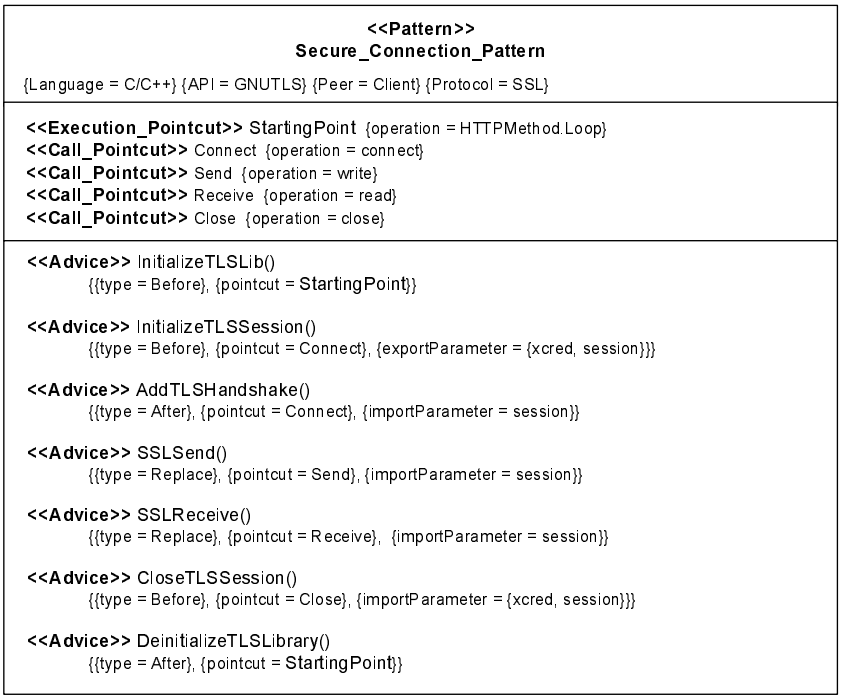

Figure 6. The Secure Connection Pattern.

The pattern is represented by a class stereotyped ≪`Pattern`≫. The tagged values attached to this stereotype specify the pattern's parameters. The pattern contains a set of security behaviors and the locations in the base model where they should be applied. The security behaviors are represented by operations stereotyped ≪`Advice`≫. The locations where the security behaviors will be injected are specified using attributes stereotyped ≪`Execution_Pointcut`≫/≪`Call_Pointcut`≫. This pattern explains in an abstract way the steps and actions to be performed for securing the connections. An example of these actions is replacing all the functions responsible for sending and receiving data on the secure channels by the ones providing TLS.

The behavior of the security methods used by the pattern is specified in behavioral diagrams, such as activity, sequence and state machine diagrams. An example is given in Figure 7. It depicts an activity diagram that specifies the detailed behavior of the

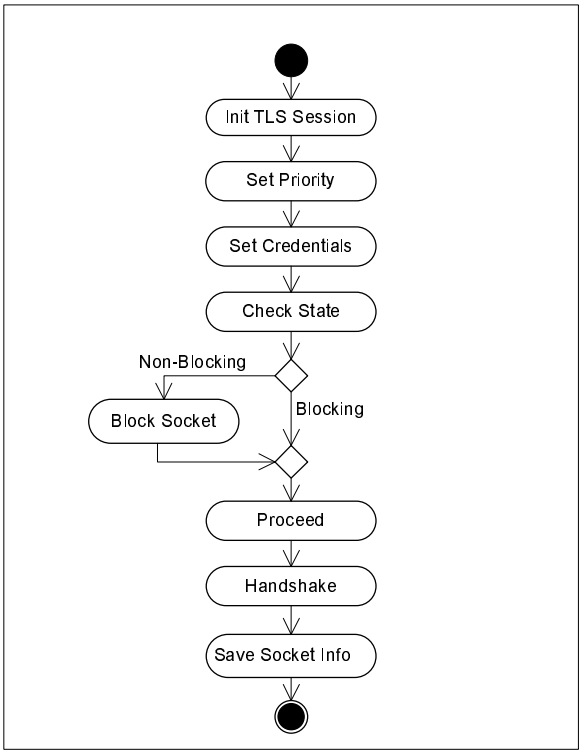

Figure 7. Behaviors of the Advices InitializeTLSSession and AddTLSHandshake.

advices `InitializeTLSSession()` and `AddTLSHandshake()` inserted around the function `connect()`.

4.3. Security Hardening Aspect

We realized the UML-based security hardening solution by implementing its corresponding aspect using AspectC++. Listing 1 shows the advice-pointcuts that constitute the aspect.

The first advice-pointcut matches the call to the content of the function `HttpMethod::Loop` to initialize the GnuTLS API at the beginning and de-initialize it at the end. The second advice-pointcut intercepts all the calls to the function `connect`, initializes the TLS session before and adds the TLS handshake after. The third advice-pointcut intercepts all the calls to the function `write` and replaces each one by a secure write using the TLS function `gnutls_record_send`. Similarly, the fourth advice-pointcut intercepts all the calls to the function `read` and replaces each one by a secure read using the TLS function `gnutls_record_recv`. Finally, the fifth advice-pointcut intercepts all the calls to the function `close`, terminates the TLS session before and de-initializes the created data structure after performing the call.

The reader will notice the appearance of `hardening_sockinfo_t`. These are the data structure and functions that we developed to distinguish between secure and non secure channels and export the parameter between the application's components at runtime (since the primitives `ImportParamter` and `ExportParameter` are not yet deployed into the weavers). We found that one major problem was the passing of parameters between functions that initialize the connection and those that use it for sending and receiving data. In order to avoid using shared memory directly, we opted for a hash table that uses the socket number as a key to store and retrieve all the needed information (in our own defined data structure). One additional information that we store is whether the socket is secured or not. In this manner, all calls to a `read()` and `write()` are modified for runtime checks that use the proper reading/writing functions.

Listing 1: Excerpt of Aspect for Securing Connections

```
aspect https{

advice execution( "% HttpMethod::Loop()" ) : around () {
//init gnutls lib
hardening_initGnuTLSSubsystem(NONE); hardening_socketInfoStorageInit();
tjp->proceed();
//deinit libs
hardening_socketInfoStorageDeinit(); hardening_deinitGnuTLSSubsystem();
}

advice call("% connect(...)") : around () {
    //variables declared
    hardening_sockinfo_t socketInfo;
    const int cert_type_priority[3] = { GNUTLS_CRT_X509,
        GNUTLS_CRT_OPENPGP, 0};
    //initialize TLS session info
    gnutls_init (&socketInfo.session, GNUTLS_CLIENT);
        gnutls_set_default_priority (socketInfo.session);
    gnutls_certificate_type_set_priority (socketInfo.session,
        cert_type_priority); gnutls_certificate_allocate_credentials (&
        socketInfo.xcred);
    gnutls_credentials_set (socketInfo.session, GNUTLS_CRD_CERTIFICATE,
        socketInfo.xcred);

    //check if non-blocking. If so, make blocking until we are done with
        the handshake
    int socketflags = fcntl(*(int *)tjp->arg(0),F_GETFL);
    if ((socketflags & O_NONBLOCK) != 0) fcntl(*(int *)tjp->arg(0),
        F_SETFL, socketflags ^ O_NONBLOCK);
    //Connect + Handshake
    tjp->proceed();
    if (*tjp->result() < 0) {
      if ((socketflags & O_NONBLOCK) != 0)  fcntl(*(int *)tjp->arg(0),
          F_SETFL, socketflags);
      return;
    }
    gnutls_transport_set_ptr (socketInfo.session, (gnutls_transport_ptr)
        (*(int *)tjp->arg(0)));
    int result = gnutls_handshake (socketInfo.session);
    if ((socketflags & O_NONBLOCK) != 0){
      fcntl(*(int *)tjp->arg(0),F_SETFL, socketflags); //restore non-
          blocking state if it was like that
```

```
      gnutls_transport_set_lowat(socketInfo.session,0); //now make gnutls
           aware that we are dealing with non-blocking sockets
    }
    //Save Information in hash table
    socketInfo.isSecure = true; socketInfo.socketDescriptor = *(int *)tjp
        ->arg(0);
    hardening_storeSocketInfo(*(int *)tjp->arg(0), socketInfo);
    *tjp->result() = result;
}

//replacing write() by gnutls_record_send() on a secured socket
advice call("% write(...)") : around () {
  hardening_sockinfo_t socketInfo = hardening_getSocketInfo(*(int *)tjp->
      arg(0));
  if (socketInfo.isSecure)
     *(tjp->result())  = gnutls_record_send(socketInfo.session, *(char**)
          tjp->arg(1), *(int *)tjp->arg(2));
  else
     tjp->proceed();
}

//replacing read() by gnutls_record_recv() on a secured socket
advice call("% read(...)") : around () {
  hardening_sockinfo_t socketInfo = hardening_getSocketInfo(*(int *)tjp->
      arg(0));
  if (socketInfo.isSecure)
     *(tjp->result())  = gnutls_record_recv(socketInfo.session, *(char**)
          tjp->arg(1), *(int *)tjp->arg(2));
  else
     tjp->proceed();
}

advice call("% close(...)") : around () {
  hardening_sockinfo_t socketInfo = hardening_getSocketInfo(*(int *)tjp->
      arg(0)); /* socket matched by sd*/
  if(socketInfo.isSecure ){
     gnutls_bye(socketInfo.session, GNUTLS_SHUT_RDWR);
  }
  tjp->proceed();
  if(socketInfo.isSecure ){
     gnutls_deinit(socketInfo.session);
         gnutls_certificate_free_credentials(socketInfo.xcred);
     hardening_removeSocketInfo(*(int *)tjp->arg(0));
     socketInfo.isSecure = false; socketInfo.socketDescriptor = 0;
  }
}

};
```

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 8 and 9 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 8 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 9 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.

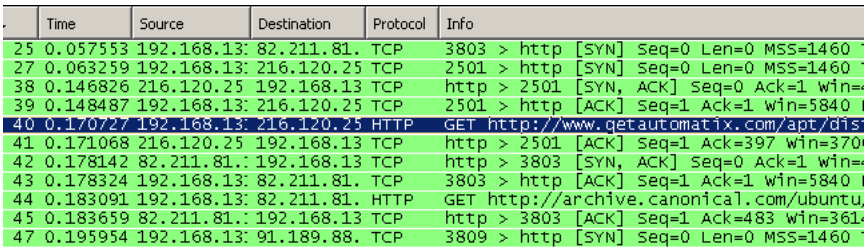

Figure 8. Packet Capture of Unencrypted APT Traffic

5. Conclusion and Future Work

In this paper, we presented a novel approach for software security hardening. The approach allows security hardening solutions to be taken care of during the earlier phases of software development life cycle. Aspect orientation is one of the key characteristics of this approach. Another characteristic is knowledge transfer and reuse. Indeed, security experts expertise is captured and packaged as security hardening solutions and provided to designers, having little or even no security expertise. Security expertise is provided as high-level solutions to particular security problems including all the details on how and where to apply them. The developers can refine the security solutions in a systematic way in order to end with the needed implementation of the security hardening solutions.

The approach provides the UML-based capabilities required for both, capturing security solutions and reusing them. These capabilities are provided thanks to the SHP profile, which is extending the UML meta-language with aspect-oriented specification capabilities. A security expert can use the SHP profile to design security solutions; as patterns, while the designer can use it to specify the needed security requirements as plans. Moreover, the specified security solutions can be systematically translated into se-

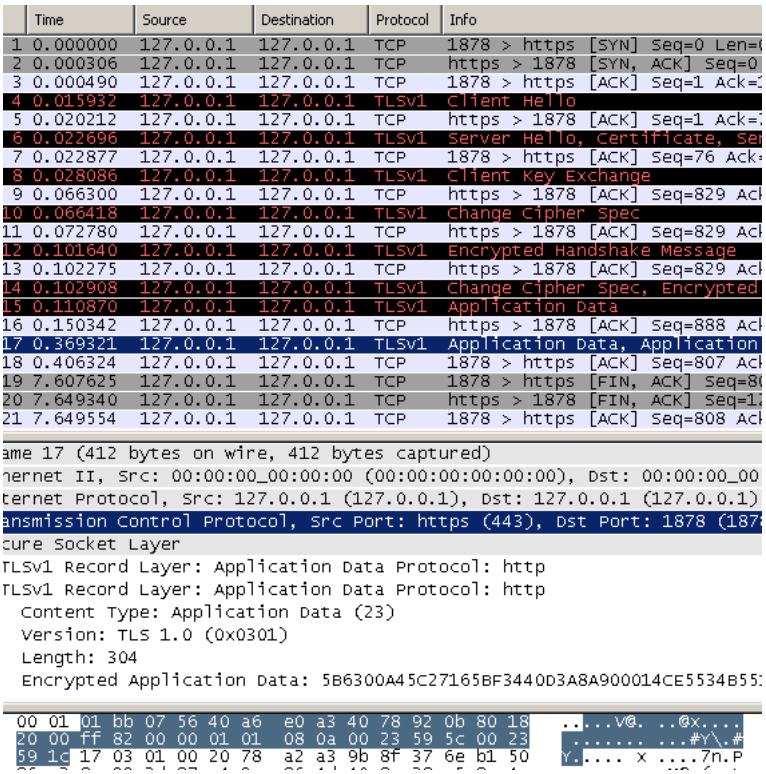

Figure 9. Packet Capture of SSL-protected APT Traffic

curity aspects code without the need of having an expertise in the security solution domain. As a result of our contribution, security solutions can be integrated into a software from the early phases of the development life cycle. This helps mainly in 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. As future work, we intend to automate the generation of the security code from the high-level solutions specified at the design level. We will also investigate other security problems, provide the corresponding security hardening solutions and apply them on more elaborated case studies.

References

[1] G. J. Ahn and M. E. Shin. UML-Based Representation of Role-Based Access Control. In *9th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE'2000*, pages 195–200, Gaithersburg, MD, USA. IEEE Computer Society.

[2] O. Aldawud, T. Elrad, and A. Bader. UML Profile for Aspect-Oriented Software Development. In *Proceedings of the 3rd International Workshop on Aspect-Oriented Modeling with UML (AOM@AOSD'2003)*, 2003.

[3] K. Alghathbar and D. Wijeskera. Consistent and Complete Access Control Policies in Use Cases. In *6th International Conference UML 2003. Model Languages and Applications*, pages 373–387, San Francisco, CA, USA, 2003.

[4] Aspect-Oriented Modeling Workshop. http://www.aspect-modeling.org/. Last visited: September 2008.

[5] E. Barra, G. Genova, , and J. Llorens. An Approach to Aspect Modelling with UML 2.0. In *Proceedings of the 5th International Workshop on Aspect-Oriented Modeling Workshop*, Lisbon, Portugal, 2004.

[6] M. Basch and A. Sanchez. Incorporating Aspects into the UML. In *Proceedings of the 3rd International Workshop on Aspect-Oriented Modeling*, Boston, MA, 2003.

[7] D. Bell and L. LaPadula. Secure Computer Systems: Mathematical Foundations Model. M74-244, Mitre Corp., 1975.

[8] B. Blakley, C. Heath, and members of The Open Group Security Forum. Security Design Patterns. Technical Report G031, Open Group, 2004.

[9] G. Brose, M. Koch, and K. P. Lohr. Integrating Access Control Design into the Software Development Process. In *Proceedings of the 6th Biennial World Conference on the Integrated Design and Process Technology (IDPT'2002)*, Pasadena, CA.

[10] C. Chavez and C. Lucena. A Metamodel for Aspect-Oriented Modeling. In *Proceedings of the 1st International Workshop on Aspect-Oriented Modeling with UML*, Enschede, The Netherlands, 2002.

[11] T. Doan, L. D. Michel, and S. A. Demurjian. A Formal Framework for Secure Design and Constraint Checking in UML. In *Proceedings of the International Symposium on Secure Software Engineering (ISSSE'06)*, Washington, DC.

[12] P. Epstein and R. S. Sandhu. Towards a UML Based Approach to Role Engineering. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 135–143. ACM Press, 1999.

[13] J. Evermann. A Meta-Level Specification and Profile for AspectJ in UML. *Journal of Object Technology*, 6(7):27–49, 2007.

[14] E. B. Fernández. A Methodology for Secure Software Design. In *Software Engineering Research and Practice*, pages 130–136, 2004.

[15] E. B. Fernandez and R. Warrier. Remote Authenticator/Authorizer. In *Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP'2003)*, 2003.

[16] D. Ferraiolo, R. Sandhu, S. Gavrila, R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and Systems Security*, 4(3):224–274, 2001.

[17] L. Fuentes and P. Sanchez. Elaborating UML 2.0 Profiles for AO Design. In *Proceedings of the International Workshop on Aspect-Oriented Modeling*, 2006.

[18] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[19] IBM-Rational Software Architect. http://www.ibm.com/software/awdtools/architect/swarchitect/. Last visited: September 2008.

[20] J. Jürjens. *Secure Systems Development with UML*. Springer Verlag, 2004.

[21] M. Kande, J. Kienzle, and A. Strohmeier. From AOP to UML - a Bottom-Up Approach. In *Proceedings of the 1st International Workshop on Aspect-Oriented Modeling with UML*, Enschede, The Netherlands, 2002.

[22] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An Overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, pages 327–353, London, UK, 2001. Springer-Verlag.

[23] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[24] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the International Conference on the Unified Modeling Language, UML'2002*, volume 2460 of *Lecture Notes in Computer Science*, pages 426–441. Springer Verlag, 2002.

[25] T. Lodderstedt, D. Basin, and J. Doser. Model-Driven Security: from UML Models to Access Control Infrastructures. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(1):39–91, 2006.

[26] C. Montangero, M. Buchholtz, L. Perrone, and S. Semprini. For-LySa: UML for Authentication Analysis. In *Global Computing: IST/FET International Workshop (GC'2004)*, volume 3267 of *LNCS*, pages 93–106. Springer Verlag, 2005.

[27] F. Mostefaoui and J. Vachon. Formalization of an Aspect-Oriented Modeling Approach. In *Proceedings of Formal Methods*, Hamilton, ON, 2006.

[28] D. Mouheb, M. Debbabi, L. Wang, and M. Pourzandi. UML-Based Approaches for the Development of

Secure Software and Systems: A Comparative Study. In *Proceedings of the 2nd Workshop on Practice and Theory of IT Security (PTITS'2008)*, pages 15–20, Montreal, Canada, 2008.

[29] A. Mourad, M. A. Laverdière, and M. Debbabi. New Primitives to AOP Weaving Capabilities for Security Hardening Concerns. In *Proceedings of the 9th International Conference on Enterprise Information Systems, Security in Information Systems Symposium (ICEIS-WOSIS 2007)*, Funchal, Madeira, Portugal, June 2007.

[30] National Computer Security Center, Department of Defense. A Guide to Understanding Discretionary Access Control in Trusted Systems. NCSC-TG-003.

[31] Object Management Group (OMG). MDA Guide, Version 1.0.1, 2003.

[32] Object Management Group (OMG). UML 2.0 OCL Specification, Version 2.0, 2006.

[33] Object Management Group (OMG). Unified Modeling Language: Superstructure, Version 2.1.2, 2007.

[34] Object Management Group (OMG). XML Metadata Interchange, Version 2.1.1, 2007.

[35] F. Painchaud, D. Azambre, M. Bergeron, J. Mullins, and R. M. Oarga. SOCLe : Integrated Design of Software Applications and Security. In *Proceedings of the 10th International Command and Control Research and Technology Symposium (ICCRTS'2005)*, McLean, VA, USA, 2005.

[36] J. Pavlich-Mariscal, T. Doan, L. Michel, S. Demurjian, and T. Ting. Role Slices: A Notation for RBAC Permission Assignment and Enforcement. In *Proceedings of the 19th Annual IFIP WG 11.3*, pages 40–53, Connecticut, U.S.A, 2005.

[37] J. Pavlich-Mariscal, L. Michel, and S. Demurjian. Enhancing UML to Model Custom Security Aspects. In *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling (AOM@AOSD'07)*, 2007.

[38] G. Florin F. Legond-Aubry L. Seinturier R. Pawlak, L. Duchien and L. Martelli. A UML Notation for Aspect-Oriented Software Design. In *Proceedings of the 1st International Workshop on Aspect-Oriented Modeling with UML*, Enschede, The Netherlands, 2002.

[39] I. Ray, R. France, N. Li, and G. Georg. An Aspect-Based Approach to Modeling Access Control Concerns. *Information and Software Technology*, 46(9):575–587, 2004.

[40] A. Reina, J. Torres, and M. Toro. Towards Developing Generic Solutions with Aspects. In *Proceedings of the 5th International Workshop on Aspect Oriented Modeling with UML (AOM@UML'2004)*, Lisbon, Portugal, 2004.

[41] S. Romanosky. Security Design Patterns Part 1. http://www.romanosky.net/, 2001. Last visited: September 2008.

[42] A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, M. Wimmer, and G. Kappel. A Survey on Aspect-Oriented Modeling Approaches. Technical Report, Vienna University of Technology.

[43] M. Schumacher. *Security Engineering with Patterns*. Springer, 2003.

[44] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: An Aspect-Oriented Extension to the C++ Programming Language. In *Proceedings of the 40th International Conference on Tools Pacific (CRPIT'02)*, pages 53–60, Darlinghurst, Australia, 2002.

[45] D. Stein, S. Hanenberg, and R. Unland. A uml-based aspect-oriented design notation for aspectj. In *Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD'02)*, pages 106–112, New York, NY, USA, 2002. ACM.

[46] C. Talhi, D. Mouheb, V. Lima, M. Debbabi, L. Wang, and M. Pourzandi. Usability of Security Specification Approaches for UML Design: A Survey. To appear in the Journal of Object Technology (JOT).

[47] H. Yan, G. Kniesel, and A. Cremers. A Meta-Model and Modeling Notation for AspectJ. In *Proceedings of 5th the International Workshop on Aspect-Oriented Modeling*, Lisbon, Portugal, 2004.

[48] A. Zisman. A Static Verification Framework for Secure Peer-to-Peer Applications. In *Second International Conference on Internet and Web Applications and Services (ICIW'07)*, page 8, 2007.