Common Criteria CompliAnt Software Development (CC-CASD)

Kristian Beckers, Stephan Faßbender, Denis Hatebur, Maritta Heisel
Duisburg, Germany
{firstname.lastname}@uni-due.de

Isabelle Côté
ITESYS
Dortmund, Germany
{firstname.lastname}@itesys.de

ABSTRACT
In order to gain their customers’ trust, software vendors can certify their products according to security standards, e.g., the Common Criteria (ISO 15408). However, a Common Criteria certification requires a comprehensible documentation of the software product. The creation of this documentation results in high costs in terms of time and money.

We propose a software development process that supports the creation of the required documentation for a Common Criteria certification. Hence, we do not need to create the documentation after the software is built. Furthermore, we propose to use an enhanced version of the requirements-driven software engineering process called ADIT to discover possible problems with the establishment of Common Criteria documents. We aim to detect these issues before the certification process. Thus, we avoid expensive delays of the certification effort. ADIT provides a seamless development approach that allows consistency checks between different kinds of UML models. ADIT also supports traceability from security requirements to design documents. We illustrate our approach with the development of a smart metering gateway system.

Categories and Subject Descriptors
D.2.0 [Software Engineering]: Standards; D.2.1 [Software Engineering]: Requirements/Specification; K.4.2 [Software Engineering]: Security

Keywords

1. INTRODUCTION
Software vendors have to gain their customers’ trust. One way of gaining it is with a security certification. The ISO 15408 Standard - Common Criteria for Information Technology Security Evaluation (short CC) - [14] provides a certification schema for software systems. However, a CC certification demands a detailed documentation of the software system. The system is called Target of Evaluation (ToE) in the CC. A ToE has to be described in detail including its environment. In addition, the attackers, threats, assumptions, security objectives, and security functional requirements of the ToE have to be elicited and reasoned in detail. The CC demands a description of the implementation of a ToE.

The standard provides a specific structure the documentation has to follow. For example, each specification of a ToE has to start with an introduction that has to contain a description of the ToE, its interfaces and the operational environment, e.g. the Operation System, the ToE works in. Furthermore, the CC follows a security model that requires a reasoning effort. For example, the creator of the document has to state why he/she thinks that the security objectives are complete and why all of them have implemented security solutions that address them.

We propose to use an extension of the requirements-driven software engineering process ADIT [7, 10]. Our contributions are two-fold: a) We provide an extension of the UML-profile UML4PF [7, 10] that introduces the terminology and the attributes of the CC. We also implement these extensions into the UML4PF support tool [8]. Hence, we extend ADIT with CC-specific model elements that can be used in the software engineering process. This provides the means to generate texts, figures or tables that can be re-used to create CC documentation. b) We re-use ADIT’s traceability and consistency checks between different models of the software engineering process. For example, ADIT provides the means to trace a software component to the requirement it fulfills. Moreover, ADIT provides checks that can analyze if all model elements, which are used in the architecture, refine a model element of the context description in the analysis phase. We improved this mechanism to support several CC elements. For example, we can check that security objectives can be traced back to the attackers and threats that caused them.

The rest of the paper is organised as follows. Section 2 presents background on Problem Frames, ADIT, and the CC. Section 3 presents our UML profile for the CC, and Sect. 4 explains which parts of the CC we support. Sect. 5 shows the application of our approach to a smart grid scenario. Sect. 6 presents related work, and Sect. 7 concludes and gives directions for future research.

2. BACKGROUND
In the following we introduce the ADIT development process (see Sect. 2.1) and the Common Criteria (see Sect. 2.2).

---

1The CC uses the term threat agent for attacker. However, we use attacker as a synonym for threat agent in this work.
2.1 The ADIT Process

ADIT (Analysis, Design, Implementation, and Test) is a model-driven, pattern-based development process also making use of components. The process is based entirely upon the UML notation [19] and defines relations between different kinds of models. The ADIT process defines consistency checks between these models including tool support [8]. The support tool is called UML4PF and contains a UML profile of the same name. The ADIT process also supports traceability between development phases. For example, the tool support provides several OCL [20] queries that can answer the question, to which requirement a design artifact refers. We focus on the analysis phase in this work and present it in the following.

A1 - Problem Elicitation and Description The process begins with a description of the desired functionality of the software to be built, the so-called machine. This description is refined into requirements and domain knowledge, which consists of facts and assumptions. We use a context diagram and domain knowledge diagrams using our UML profile and tool support. This step is based upon Problem Frames [15].

A2 - Problem Decomposition The second step decomposes the context diagram into problem diagrams (also according to [15]). Each problem diagram represents a specific problem that the requirement expresses.

A3 - Abstract Software Specification Problem diagrams present the structure of a problem and not its behavior. In this step UML sequence diagrams are used to describe the behavior of the machine. These serve also as the basis for our software specification.

A4 - Technical Context Diagram This step describes the technical environment of the machine. For example, a web application machine may use the Apache web server. We use again the UML-based notation mentioned in steps A1 and A2. We also use components and their APIs to describe the technical means, e.g., the API of the Apache web server.

A5 - Operation and Data Specification The purpose of this step is to set up the necessary internal data structures represented as analysis class diagrams. Furthermore, we specify the operations identified in Step Abstract Software Specification by providing pre- and postconditions for each operation. We use OCL [20] to express these operation specifications.

A6 - Software Life-Cycle We use life-cycle expressions proposed in the Fusion method [6] to describe the overall behavior of the machine.

Problem Frames.

Problem frames (PFs) are a means to describe software development problems. PFs were proposed by Jackson [15], who describes them as follows: “A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement.”. We represent problem frames by using UML class diagrams extended by stereotypes as proposed by Hata and Heisel [11]. All elements of a problem frame diagram act as placeholders, which must be instantiated to represent concrete problems. In doing so, one obtains a problem description that belongs to a specific kind of problem.

The class with the stereotype machine represents the thing to be developed (e.g., the software). The classes with some domain stereotype, e.g., CausalDomain or BiddableDomain represent problem domains that already exist in the application environment. Domains are connected by interfaces consisting of shared phenomena. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain, as indicated by an exclamation mark.

Jackson distinguishes the domain types CausalDomains that comply with some physical laws, LexicalDomains that are data representations, and BiddableDomains that are usually people. The stereotype «causalDomain » indicates that the corresponding domain is a CausalDomain, and the stereotype «biddableDomain » indicates that it is a BiddableDomain. In our formal meta-model of problem frames [12] Domains have names and abbreviations, which are used to define interfaces. Hence, the class Domain has the attributes name and abbreviation of type string.

2.2 The Common Criteria

The ISO/IEC 15408 - Common Criteria for Information Technology Security Evaluation is a security standard that can achieve comparability between the results of independent security evaluations of IT products (machines). These are so-called Targets of Evaluation (TOEs). The Common Criteria are based upon a general security model. The model considers ToE Owners that value their Assets and wish to minimise Risk to these Assets via imposing Countermeasures. Attackers wish to abuse Assets and give rise to Threats for Assets. The Threats increase the Risks to Assets.

Documentation of the security model is the basis for CC certification. The CC security model can be expressed in two different types of documents. The security needs of TOE owners are described in the so-called Security Target (ST). STs can be a refinement of a so-called Protection Profile (PP). A PP states the security needs for an entire class of ToEs, e.g., client VPN application. A PP states the security requirements of TOE owners and ToE developers or vendors publish their security claims in an ST.

The document structure of ST and PP is the same on the level of chapters. The first chapter is an Introduction that contains the description of the ToE and its environment. The chapter Conformance Claims describes to which PPs the ST or PP is compliant.

The chapter Security Problem Definition refines the external entities, e.g., stakeholders in the environment. In addition, the chapter lists all Assets, Assumptions about the ToE and its environment and, all Attackers, the Threats they cause to Assets, and Organizational Security Policies of the ToE’s environment. The chapter Security Objectives contains the Security Objectives for the ToE and its Operational Environment. An example for an Operational Environment is the operating system the ToE uses.

A Security Functional Requirement (SFR) is a description of a security function (or solution) in the CC. For example, how a stakeholder might be authenticated. The CC provides many generic SFRs in the form of gap texts. The gaps have to be filled for the ToE that is described in the ST/PP. The chapter Extended Component Definition describes SFRs that complement the set of SFRs that are already present in the CC. Hence, these are specific security functions of the ToE that is described in the ST/PP. The chapter Security Functional Requirements (SFR) presents instantiations (filled gaps) of all relevant SFRs from the set that the CC provides. The Security Assurance Requirements (SAR) describe the implementation of the SFRs. These description is provided for a particular Evaluation Assurance Level (EAL). An EAL is a numerical rating ranging from 1 to 7, which states the depth of the evaluation. EAL 1 is the most basic level and EAL 7 the most stringent one.
3. EXTENDING ADIT’S UML PROFILE WITH COMMON CRITERIA SUPPORT

We extend the UML profile and support tool UML4PF of the ADIT process (see Sect. 2.1) to support the Common Criteria standard. Fig. 1 depicts the extensions to UML4PF, which contain several sub-extensions.

Firstly, we introduce the ToE (see UML4PF ToE domain extension), which is derived from the machine domain in the Jackson terminology (see Sect. 2.1). Another derived element from Jackson’s terminology is the Countermeasure, which is a specific kind of CausalDomain.

Secondly, we introduce the elements of the CC basic security model (see Sect. 2.2) in the UML4PF attacker/threat extension. Assets have descriptions and a need for protection. The CC has the concept of SecondaryAssets. Harm to SecondaryAssets do not cause a loss to the ToEOwner directly, but the harm can cause harm to an Asset. This in turn can cause a loss to a ToEOwner. Assets are subject to Risks, which have likelihood and consequences according to the CC. The attributes, e.g., likelihood and consequences, of a Risk can also be increased by a Threat. Threats have an abbreviation and a description.

We extended the CC basic security model. Threats can be further divided into controlThreats and observeThreats. ControlThreats take control of a domain, while observeThreats only observe information about the behavior of a domain. For example, an observeThreat is the eavesdropping of confidential information, whereas the manipulation of a key exchange is a controlThreat.

The UML4PF attacker/threat extension contains also enumerations to represent the attributes required to describe an Attacker according to the CC. The attributes, e.g., ElapsedTime or WindowOfOpportunity have also numeric values attached in brackets. These values are defined by the CC and are used to determine the EAL for a system. For example, an attacker with a combined score between 10 and 13 results in a recommendation to implement the security assurance classes AVA VAN.1 and AVA VAN.2. This recommendation results in an EAL requirement of at least EAL 2.

We extended the CC basic security model also with an Attacker classification. PhysicalAttackers threaten the physical elements of the system, e.g., hardware or buildings that host computers. NetworkAttackers threaten Network connections or ConnectionDomains in our models. SoftwareAttackers threaten CausalDomains, e.g., the ToE or another software component. SocialEngineeringAttackers threaten biddable domains, e.g., users of the system.

Thirdly, we collect information that is required for the generation of tables for ST/PP documents. The UML4PF CC-System extension contains attributes. The instantiations of which can be used to generate tables. These tables are the main parts of the PP/ST Reference in the Introduction and the Conformance Claims chapter. The UML4PF ToE domain extension contains the information to generate a ToE reference for the Introduction. For example, the name and the authors of the ToE are required for a ToE reference.

The UML4PF OSP extension enables the collection of all information for CC compliant Organizational Security Policies (OSP)s. The policies are imposed by a BiddableDomain in the environment of the ToE. OSPs are sets of rules, regulations, or guidelines that the environment imposes on the ToE. Hence, they are part of the domain knowledge alongside facts and assumptions.

We also added several consistency checks for modeling attackers using our profile. These checks are implemented in OCL [20]. For example, we check that attackers only threaten domains they can reach. The network attacker can only threaten connections, e.g., WLAN connections or connection domains.

4. EXTENDING ADIT’S PHASES TO SUPPORT THE COMMON CRITERIA

We analyzed the documents resulting from the ADIT software development process and the document requirements of the Common Criteria. We present the results in Tab. 1. The table lists the chapters and sections of Security Targets and Protection Profiles in the first two columns. The sections that are only relevant for a Security Target are annotated with an “(ST)”, the sections only relevant for a Protection Profile are annotated with a “(PP)”.

The following columns state the related ADIT phase and the output document types of this phase. The “?” sign states that the ADIT process does not produce a document that can support the CC. We listed the extensions of the ADIT process that we made for this work in italics.

These represent the contribution of our work. We introduced specific classes to represent CC-Systems and specific CC artifacts, e.g., organizational security policies, in the UML4PF profile. The Introduction, Conformance Classes and Security Problem Definition chapters of the ST and PP are CC-specific. Hence, these require numerous extensions to ADIT as extension to the UML4PF profile and in the form of OCL expressions. The Security Objectives chap-
5. APPLICATION

For simplicities' sake, we provide only a short example of relevant ADIT artifacts we use for supporting the establishment of a CC-compliant ST/PP document.

We show an example how a ToE description can be generated using the UML4PF CC-system extension and a list of assets of the ToE and its environment. Hence, we provide example artifacts for a ST/PP: the ToE-Overview in the Introduction and the Assets in the Security Problem Definition.

Running Example.

The Smart Metering Gateway is a part of the smart grid. This is a commodity network that intelligently manages the behavior and actions of its participants. The commodity consists of electricity, gas, water, or heat that is distributed via a grid (or network). The benefit of this network is envisioned to be more economic, sustainable and to secure supply of commodities. Smart metering system meters the consumption or production of energy and forwards the data to external entities. This data can be used for billing and steering the energy production. This system description is taken from a PP [4]. We used the information in the PP to generate our artifacts and to check that our results match the example in the PP. Hence, our results can be used to instantiate the resulting model in order to create an ST.
Figure 2: The Context Diagram of the Smart Metering Gateway

TOE Overview.

The TOE overview contains a description of the machine to be built in its environment. We propose to use the ADIT Phase A1 for this purpose. This phase relies upon the problem frame approach as introduced in Sect. 2. We provide a context diagram and technical context diagram to describe the TOE (see Fig. 2). The TOE is a SmartMeteringGateway, which serves as a bridge between the Wide Area Network (WAN) and the Home Area Network (HAN) of the TOE owner. The TOE owner is the commodity Consumer of the smart grid scenario. The Meter is connected to the machine via a Local Metrological Network (LMN) and measures the consumption of commodities of the Consumer. The LMN is an in-house network for equipment that can be used for energy management. The Controllable Local System (CLS) is a system that can be controlled using the SmartMeteringGateway and is connected to the SmartMeteringGateway with the HAN. For example, an air conditioning unit or an intelligent fridge. The Consumer can also access the SmartMeteringGateway. A detailed interface description, e.g., IF_GW_WAN can be found in [4].

The TOE Reference for a PP can be generated from the information collected in the context diagram and technical context diagram. We use an OCL expression to extract the collected information. We present Tab. 2 as an output of the example reference.

```
Property.allInstances() -> select (getAppliedStereotypes(), name -> includes ('Asset'))
-> collect(c |
  let st : Stereotype = c.getAppliedStereotypes() -> select(name = 'Asset') ->
  asSequence() -> first() in
  c.name.oclAsType(String), c.oclAsType(Asset)
).concat(';')
```

Listing 1: Collecting assets and their attributes.

We generate a table using the information collected by the OCL expression presented in 1. This expression selects all classes, which name includes the String Asset. The expression further creates a sequences that contains the name of the classes and the values of the attributes description and needForProtection. These values are separated using a “;”.

<table>
<thead>
<tr>
<th>Asset</th>
<th>Description</th>
<th>Need for Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meter Data</td>
<td>Meter readings that allow calculation of the quantity of a commodity, e.g. electricity, gas, water or heat consumed over a period. Meter Data comprise Consumption or Production Data (billing-relevant) and grid status data (not billing-relevant). While billing data needs to have a relation to the consumer grid status data, it does not need to be directly related to a consumer.</td>
<td>According to their specific need.</td>
</tr>
<tr>
<td>Gateway time (secondary asset)</td>
<td>Date and time of the real-time clock of the Gateway Time is used in Meter Data records sent to external entities. Date and time of the real-time clock of the Gateway Time.</td>
<td>Integrity and Authenticity (when time is adjusted to an external reference time).</td>
</tr>
<tr>
<td>Meter config (secondary asset)</td>
<td>Configuration data of the Meter to control its behaviour including the Meter identity.</td>
<td>Integrity and Authenticity and Confidentiality</td>
</tr>
</tbody>
</table>

Table 3: Table for Assets of the PP/ST

Re-using ADITs Traceability and Consistency checks.

For the ADIT development process more than 70 traceability and consistency checks between the different diagrams have been defined [7]. For example, it is checked that each element in a problem diagram has a relation to the context diagram. A possible relation is that an element in the problem diagram is part of an element in the context diagram. Such relations support the refinement and detailed descriptions of security properties. These are essential in order to ensure an adequate protection against threats.
6. RELATED WORK

Mellado et al. [16, 17] created the Security Requirements Engineering Process (SREP). SREP is an iterative and incremental security requirements engineering process. In addition, SREP is asset-based, risk driven, and follows the structure of the Common Criteria. The approach uses use cases to model security objectives, and misuse cases to model threats. The authors also developed a template for ranking threats, attacks, and risks. The authors propose a Security Resources Repository (SRR) that can store elicited threats, attacks, and risks. The approach differs from our work in the sense that SREP is a method that supports the security reasoning according to the CC. However, it is not a holistic software development process.

Yin and Qiu [21] model so-called early-phase security requirements with an extended i* model. The authors also describe security policies using a formal model and so-called late-stage security requirements in an extended UML model. The extended i* framework adds the modeling elements security flaw that can have a relation to goals and soft goals. These can be influenced by a threat and eliminated by a security goal, e.g., confidentiality. The policies provide three templates for so-called stream control that specifies rules for network traffic. For example, allowed IP addresses. The extended UML considers explicitly for each element if it belongs to the ToE, external entities or communication entities. The approach differs from ours, because it focuses on generating CC policies for stream control. The approach does not aim at providing a holistic support for the generation of ST/PP documents.

Bialas [2] introduces an ontology that supports the CC security problem definition (SPD). The SPD contains threats, security policies, and assumptions concerning the ToE. The ontology provides relations between security related elements, e.g., risks and threats. The relations can be used to create a SPD. For example, the ontology can provide threats for specific risks. In addition, the ontology can be queried for known countermeasures for these risks. The author extended the approach to the IT security development framework that is compliant to the entire CC [3]. The approach differs from ours, because the author focuses on creating just the SPD and not a holistic support for generating ST/PP documents.

Ardi and Shahmehri [1] extend the CC Security Target document with the knowledge of existing vulnerabilities. In particular the authors add threats from known vulnerabilities to the Security Problem Definition, security objectives from vulnerabilities and information on how to consider these vulnerabilities in the Security Objectives section. The authors use vulnerability cause graphs and security activity graphs to refine the information from the vulnerabilities. This work can complement our own. We can use the information about existing vulnerabilities in our process as well.

Chang and Fan [5] design an ontology that is intended to decrease the time for CC certification. The ontology supports four different use cases. The first is to query CC content using a hierarchical tree. The second use case considers a markup tool that allows the user to mark specific parts of the CC. These marks can include a choice of predefined comments that can be used to ease the review of CC documents. The third use case considers a CC review tool that can provide a checklist of required documents and for evaluating a specific EAL. The CC review tool also contains information about required documents. This include already written and approved documents and documents that have to be revised. The last use case concerns a review report tool. This tool provides a judgement of the review process using the data from the previous use case. This work can complement our own. We could use their ontology after generating documents with our method.

Rottke et al. [18] present a problem-driven requirements engineering method for CC compliant systems. This high level approach also considers Problem Frames. The method focuses on creating reliable models for context and problem descriptions. This work differs from ours, because we focus on the entire CC process and do not limit our approach to context and problem descriptions.

7. CONCLUSION

We have extended the ADIT software development process with a UML profile for Common Criteria systems. Thereby, we built on the existing UML4PF tool and its UML profile for dependability [10]. We provide UML elements to model security as described in the Common Criteria. The ADIT process relies upon the Problem Frame method for a structured elicitation and analysis of requirements.

Our method offers the following main benefits:

- A structured process for elicitation of the required knowledge for a Common Criteria certification
- Computer-aided generation of tables and figures for each chapter of a Common Criteria Protection Profile or Security Target
- Support for the reasoning of Common Criteria Security Objectives from ToE descriptions to Common Criteria Security Objectives
• Traceability from elements of to the ToE description to threats on to Common Criteria Security Objectives
• Consistency checks from elements of the ToE description to Common Criteria Security Objectives
• Explicit consideration of domain knowledge in terms of assumptions and facts
• Re-use of patterns, e.g., problem diagrams for different projects

The work presented here will be extended to support further security standards, e.g., the ISO 27001 standard [13]. We also aim to support the choice of relevant Common Criteria Security Functional Requirements for given problem diagrams. In addition, we want to support the decision if the Common Criteria Security Functional Requirements have to be extended for a given ToE.

Acknowledgment
This research was partially supported by the EU project Network of Excellence on Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980) as well as the Ministry of Innovation, Science, Research and Technology of the German State of North Rhine-Westphalia and EFRE (Grant No. 300266902 and Grant No. 300267002).

8. REFERENCES