Mind the Gap: Formal Verification and the Common Criteria (Discussion Paper)

Bernhard Beckert
Karlsruhe Institute of Technology, Germany
beckert@kit.edu

Daniel Bruns
Karlsruhe Institute of Technology, Germany
bruns@kit.edu

Sarah Grebing
Universität Koblenz-Landau, Germany
sarahgrebing@uni-koblenz.de

Abstract

It is a common belief that the rise of standardized software certification schemes like the Common Criteria (CC) would give a boost to formal verification, and that software certification may be a killer application for program verification. However, while formal models are indeed used throughout high-assurance certification, verification of the actual implementation is not required by the CC and largely neglected in certification practice – despite the great advances in program verification over the last decade.

In this paper we discuss the gap between program verification and CC software certification, and we point out possible uses of code-level program verification in the CC certification process.

1 Introduction

Software certification is commonly seen as a powerful means to achieve high-quality software – particularly if it is employed in a security-critical context. The Common Criteria (CC) [5] are a widely recognized international standard for computer security, which has also attracted the interest of formal methods researchers over the last years. As the CC clearly prescribe the usage of formal methods at their highest assurance levels, this is seen by many as an opportunity to bring formal software verification to a broader audience. Many believe that software certification may indeed be a killer application for program verification.

However, while formal models are indeed used throughout high-assurance certification and it is mandatory to employ functional tests, verification of the actual implementation at code-level is not required by the CC and largely neglected in certification practice.

At the same time, the CC are being criticized by practitioners: The The CC were too costly, produced too much paperwork, and adhered to a legacy process model.

In this paper we discuss the gap between program verification and CC software certification, and we point out possible uses of code-level program verification in the CC certification process.

After reviewing the basics of software certification in general (Sect. 2) and the Common Criteria in particular (Sect. 3), we give an overview of the various (formal and informal) specification documents that are used in a CC evaluation (Sect. 4). We then review case studies in using formal methods for CC certification at the abstract level (Sect. 5), and we present results of our own investigation based on [9] into using code-level specification and verification in CC certification (Sect. 6). Finally, we draw conclusions from our investigation in Section 7 and show that there still remains a gap between verification and certification.

2 Software Product Certification

Certification is the assurance of quality properties for a technical entity based on verifiable evidence, according to well-defined standards, and usually involving an independent third party – the certification agency –, while the product's end-user requires only minimal knowledge of the techniques and tools used in the certification process itself. There are various reasons to certify software products: Authorities prescribe certification for legal reasons or reasons of public safety and security. Customers of software developers ask for certificates because for them software failures are costly, in particular if they integrate the software into their own products. And finally, certification is a marketing argument as consumers demand dependable systems.

Improved availability of comprehensive product certification in software engineering would have considerable advantages. In particular, software engineers have been promoting a paradigm shift from process- to product-oriented certification for a long time [15, 23]. Software quality assurance should take a step forward from the "pass/fail" stereotype of normative assurance to a descriptive appearance in which a certificate describes whether the product "does what one wanted" [24].

3 Common Criteria

The *Common Criteria for Information Technology Security Evaluation* (CC) are an international standard for the security of software and hardware products established in 1999. It supersedes the previous standards ITSEC (Europe) and TCSEC (US). The Common Criteria are now officially recognized by 24 countries and are administered by respective domestic authorities, e.g., the Bundesamt für Sicherheit in der Informationstechnik (BSI) in Germany. Assessment of products may in turn be delegated to accredited third-party agencies. For an overview see, e.g., the book by Herrmann [10].

As compared with other certification schemes, the Common Criteria put a strong emphasis on the validation of the product's specification. In this sense, they are notably more product-oriented and less process-oriented than other standards. They clearly distinguish between responsibilities of developers and evaluators, and they name evaluation deliverables and presentation items which are meant to demonstrate the claimed properties. Although essentially of strong normative character, the Common Criteria may also be seen as a step towards a descriptive certification as a description of the product's security features is contained in the public document (the *Security Target*) which is issued alongside a certificate.

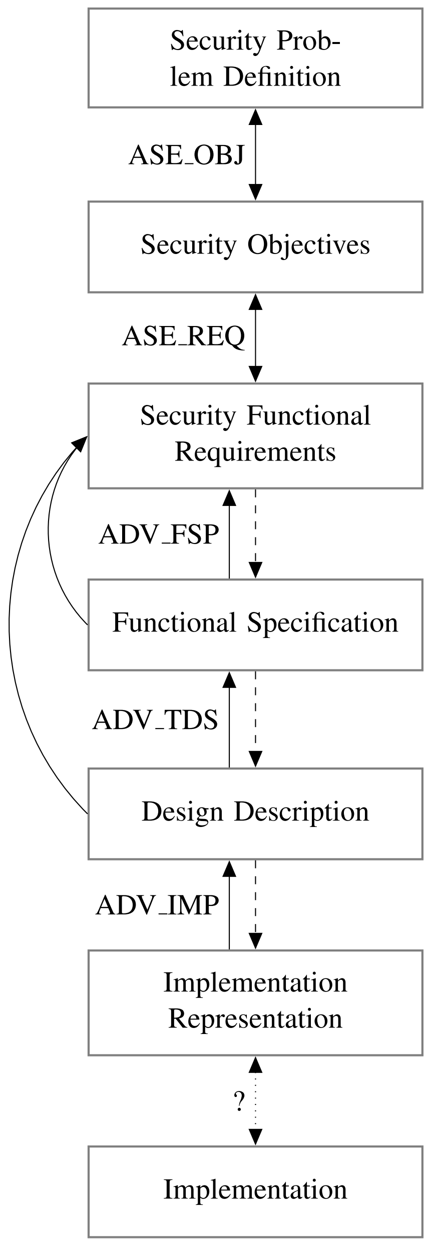
There are seven *Evaluation Assurance Levels* (EALs) defined in Common Criteria, where EAL 7 is the strongest. Although commonly misunderstood as a measure of security, EALs are a measure of assurance quality. They define the degree of rigor and depth that has been applied to assure the claimed security properties.

4 Formal Specification and Verification in Common Criteria

4.1 Abstract-level Specification and Verification

Assessment at the two highest levels, EAL 6 and 7, requires formal methods – to some extend. These are primarily found in three documents related to development (apart from development, the CC cover the whole software life-cycle): the formal *security policy model* (SPM), the *functional specification* (FSP), and the *target design presentation* (TDS). These documents relate to different levels of abstraction, see Fig. 1.

Security Policy Model (SPM). The SPM is a, usually product-independent, formalization of the overall security policy, i.e., it gives a formal definition of what is meant by "security" in the context of the assessed product. Or, alternatively, SPM is a system invariant which defines a secure state for that product category. It then has to be shown at the lower abstraction levels that no insecure state is reachable.



Dashed arrows (- -►) denote a refinement relation.

Solid arrows (—►) denote a proof of correspondence. (The level of rigor for such proofs depends on the level of formality of the related models.)

Names on the arrows refer to the assurance package that includes the proof:

ASE_OBJ:	Security Objectives
ASE_REQ:	Security Requirements
ADV_FSP:	Functional Specification
ADV_TDS:	Design Description
ADV_IMP:	Implementation Representation
ADV_SPM:	Security Policy Model (not shown)

It is not clarified within the CC framework how the implementation representation as the lowest level of specification documents relates to the actual implementation.

Figure 1: Different layers of specification documents that are required at higher-level evaluations (EAL 5 and up).

Functional Specification (FSP). The FSP can be seen as a refinement of the Security Policy Model (SPM). It addresses single security functions, e.g., access control or event logging. The specifications contained in the FSP are usually formalizations of *security functional requirements* (SFR), which are standardized components defined by the Common Criteria written in natural language (see Fig. 2 for an example). The FSP can thus be seen as description of which security functionality is provided by the product. It then has to be shown that this functionality fulfills the overall policy defined by the SPM. If both the FSP and the SPM are formal models, this may be done using a formal proof.

Design Presentation (TDS). The TDS is a model of the implementation design. From EAL 5 on, the

TDS is required to be written in semi-formal style, i.e., most probably UML class diagrams. At EAL 7, the TDS has to be complemented by a formal "high-level design presentation". It is however not entirely clear where to draw the dividing line between high and low level – thus to what extend formal methods are to be applied ("high-level" in this context refers to larger subsystems, whereas "low-level" refers to the world of single modules or classes). Again, there has to be a formal proof of correspondence between the formal high-level design model and the formal functional specification (FSP). These proofs are in practice often considered the main bottleneck of the Common Criteria – with or without formal methods.

4.2 Code-level Specification and Verification

In fact, the Common Criteria is not concerned with software verification at all. The whole validation process is instead based on the documents mentioned in Sect. 4.1 above, i.e., an abstract representation of the implementation at the lowest level. Whether the program code actually conforms to the specified security functionality is never formally proven. The proponents of the Common Criteria see verification is as a possible – but complementary or even orthogonal – means to assurance. This has widely provoked criticism among software engineers as being "testing the paperwork" (e.g. [11]) rather then assessing the concrete product.

Even though a considerable amount of program testing is mandatory, at neither assurance level test cases are required to formally relate to the security model or the design specification, i.e., being model-based or systematically constructed by some other means using information from the (intended) design. As explained above, all the provided formal models are rather abstract, which makes model-based testing or even test-case generation difficult. DeLong and Rushby [7] have also shown that the higher the assurance level is, the wider is what they call the "abstraction gap" between the evaluations of specification documents and the actual product.

5 Previous Work in Abstract-level Formal Methods for CC Certification

Some case studies and real-world certifications using formal methods have been done, though not very many. The most noteworthy example consists of an implementation of a Java Card Virtual Machine, which is actually in the field, along with formal specifications [4]. It has been successfully assessed according to Common Criteria. Although only at EAL 4+[1], all formal methods required on EAL 7 have

Figure 2: An example for a standardized generic functional requirement (SFR) and its instantiation: The SFR "Security Audit Event Selection" as included in Part 2 of the Common Criteria [5, Part 2, Sect. 8.6] (above); and an instantiation with concrete actions (below). See Section 6 for further discussion of this SFR.

FAU_STG.4: The [security functionality] shall *[selection, choose one of: "ignore audited events", "prevent audited events [...]", "overwrite the oldest stored audit records"]* and *[assignment: other actions to be taken in case of audit storage failure]* if the audit trail is full.

FAU_STG.4: The [security functionality] shall prevent audited events [...] and notify, for example by throwing an error message, if the audit trail is full.

[1]EAL *n*+ stands for "EAL augmented", i.e. in addition to all requirements at EAL *n*, some assurance packages from higher levels have also been applied.

been applied. The formal models have been given in terms of Abstract State Machines [3] and consist of four different machines at the different levels of abstraction, which are refinements of each other: one for the security objectives, one for the SPM, one for both the FSP and the high-level part of the TDS, and one for the low-level part of the TDS. While the proof of refinement from functional specification to design is trivial in this case, the main effort lies in the proof of correspondence between the functional specification and the security policy model.

In a similar, more academic, case study [16] on Java Card technology, all specifications are given in the form of B machines [1] (except for the semi-formal low-level design description, which is written in UML). Again, these machines are proven to be refinements of each other.

In the certification effort by Woodcock et al. [21, 26] – possibly the most well known – the Mondex electronic purse smart card was successfully assessed at ITSEC level E6, which may be compared to EAL 7. Specifications are given in the Z specification language [25]; proofs were done with pen and paper.

All these works profit from a well-defined formal notion of refinement in the respective specification formalism. But, in neither case, source code has been investigated, although all papers name this to be the next important step in a successful thorough evaluation.

There are also projects in which real-world software is verified "in a strong certification context", such as Verisoft [22] or L4.verified [12]. While they were successful in verification, their products are yet to be certified.

6 A Case Study in Code-level Specification and Verification

As said above, the original certification of the Mondex electronic purse did not include program verification. This was the starting point of a verification challenge which was taken on by several groups [8, 17, 19, 20]. Schmitt and Tonin [20] reimplemented the Mondex protocol in Java Card and added code-level specifications on the basis of the Z specification by Woodcock et al. [21] (the publicly available Z specification is not the original one used in the actual certification). These code-level specifications are written in the Java Modeling Language (JML) [13], a widely used all-purpose specification language based on the design-by-contract paradigm and written directly into source files as comments. Schmitt and Tonin have also formally verified functional correctness of the implementation using the KeY tool [2]. We will discuss an excerpt from the specification below.

Based on this work, we have investigated how formal specification and verification relates to the different requirements of the CC and how code-level specification and verification may be used in a real world certification process.

As the implementation is not part of the hierarchy of formal CC documents, we instead begin with the implementation representation (IMP). It shall "capture the detailed internal workings", as defined in [5, Part 1, Sect. 4.1]. The main goal here is to ensure that the security functionality, which is described in a higher level of the design, is actually implemented. Of course, this assurance – if required to be of a formal kind – can only be made in a sound way if the program code is verified. In fact, code-level specification may readily be used as an implementation representation, provided that it covers the full functional behavior of a module or method.

In the example in Fig. 3, the method `abort_if_necessary` watches over the event log of the card. Normally all events are logged and the system changes its internal state. The more interesting case is when the logging capacities are exhausted, which is indicated by the index reaching the boundary `exLog.length` (Line 3). In this exceptional case the method is expected to terminate abruptly by throwing an exception (Line 5) while the index and internal state are kept in their current state (Lines 7–8). Other sources of exceptional behavior are ruled out (Lines 4 and 6).

Figure 3: Code-level specifications in JML (Mondex case study)

```
/*@ public exceptional_behavior
  @   requires (status == Epv || status == Epa) &&
  @       logIdx == exLog.length;
  @   signals_only ISOException;
  @   signals (ISOException e)
  @       e.getReason() == SW_LOG_FULL &&
  @       \old(logIdx) == logIdx &&
  @       \old(status) == status;
  @*/
private void abort_if_necessary () throws ISOException;
```

At the next higher level is a design description (TDS) of the system. In the TDS, the product is decomposed into sub-systems and modules. In our example, a module corresponds to a single Java method and sub-systems correspond to collections of methods. While sub-systems have to be modelled formally at EAL 7, this is not required for modules:[2]

> **ADV_TDS.5.7C**: The design shall provide a semiformal description of each module in terms of its purpose, interaction, interfaces, return values from those interfaces, and called interfaces to other modules, supported by informal, explanatory text where appropriate.

It is, of course, not illegal to use a formal specification at module level, thus the above code-level specifications might also qualify as a design description as well. If needed, semiformal documents like class diagrams can be derived from the given Java class structure. As another opportunity, the required "informal, explanatory text" could be derived through automated translation of formal specifications to natural language.

A further level of abstraction is the functional specification (FSP). It is meant to reflect the functional requirements (SFR) in a formal style. This correspondence has to be demonstrated by a "rigorous argument" in CC jargon. Recall the example consisting of the requirements given in Fig. 2 and the code-level specification in Fig. 3). The specification states that, in case of a full log (or "audit trail"), no event occurs (as this would mean changing the state) and this situation is signalled through means of a thrown exception. It remains to show, however, that all audited events occur within the scope of that method. Finally, the CC require a formal proof of correspondence between the design description and the functional specification. In this case this is trivial since both have the same model.

7 Conclusion

In this paper, we have shown that it is theoretically possible to use code-level specifications as development models in the Common Criteria. However, they are not required, and whether they are adequate in real-world certification will depend on the evaluator. The standard itself tends to be very vague on *how* formal methods are used. In our opinion it is also possible to replace functional testing by formal verification altogether even if verification is not required explicitly.

While it is clear that most security policies are only expressible at a high level of abstraction, they are only enforceable if there is a sound trail of evidence, tracing from the abstract-level specification to the implementation. If the principal goal of the CC for high-assurance reliably secure software is to be reached, certification has to include state-of-the-art verification techniques.

[2]Cf. [5, Part 3, Sect. 12.6.1, p. 113]

We are not aiming at the creation of an even higher assurance level beyond EAL 7. This would bear the risk of even lowering the overall acceptance of formal verification in the industry. Statistics [6] show that of over a thousand certified products, the vast majority has reached EAL 4+ or below, while EAL 5 is at least quite common for smart cards and their applications. Only three hardware devices and no software products were awarded EAL 6 or higher until now. This shows that hopes for a broader usage of formal methods through means of the Common Criteria were not fulfilled. And even in academic contexts, there are few results. As explained in the work by Chetali and Nguyen [4], reaching an actual certification is far more work-intensive, and verification is just a fraction of that. In the end, they reached only EAL 4+. A full EAL 7 evaluation would also require other laborious tasks such as integration and penetration tests.

In the end, achieving wide-spread use of formal methods in certification requires changes to the CC itself. And that is to a great extend a political question. The possibility to include verification in the up-coming Version 4 of the CC (to be finalized end-2010) has been discussed within the CC community, cf. [14], but the central aim of the new revision seems to be at the opposite end: to lower requirements for the lower assurance levels. This situation is similar with the international standard DO-178B [18] for the safety certification of avionics software. It does not provide guidance on how to use formal methods (neither for the development nor the certification of software). Fortunately, revision of the DO-178B standard is discussed at the moment which – it is hoped – will lead to an increased use of formal methods in the certification process.

References

[1] J.-R. Abrial. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, Aug. 1996.

[2] B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[3] E. Börger and R. F. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.

[4] B. Chetali and Q. H. Nguyen. Industrial use of formal methods for a high-level security evaluation. In J. Cuéllar, T. S. E. Maibaum, and K. Sere, editors, *FM 2008: Formal Methods, 15th International Symposium on Formal Methods*, volume 5014 of *Lecture Notes in Computer Science*, pages 198–213, Turku, Finland, May 26–30 2008. Springer.

[5] *Common Criteria for Information Technology Security Evaluation*, July 2009. Version 3.1. Release 3.

[6] Common Criteria Portal. `http://www.commoncriteriaportal.org/`. Accessed March 22, 2010.

[7] R. DeLong and J. Rushby. High-assurance development and evaluations: Rethinking the Common Criteria and EAL7. In *9th International Common Criteria Conference*, Jeju, Korea, Sept. 2008. Presentation.

[8] C. George and A. E. Haxthausen. Specification, proof, and model checking of the Mondex electronic purse using RAISE. *Formal Aspects of Computing*, 20(1):101–116, 2008.

[9] S. Grebing. Towards tool support in the certification process of verified software. Studienarbeit, Universität Koblenz-Landau, 2010. To appear.

[10] D. S. Herrmann. *Using the Common Criteria for IT Security Evaluation*. Auerbach Publications, 1st edition, Dec. 27 2002.

[11] W. Jackson. Under attack. *Government Computer News*, 26(8), Aug. 10 2007.

[12] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. In J. N. Matthews and T. E. Anderson, editors, *22nd ACM Symposium on Operating Systems Principles, SOSP 2009*, pages 207–220, Big Sky, Montana, USA, 2009. ACM.

[13] G. T. Leavens, A. L. Baker, and C. Ruby. Preliminary design of JML: a behavioral interface specification language for Java. *SIGSOFT Softw. Eng. Notes*, 31(3):1–38, 2006.

[14] D. Martin. CCDB report and overview of CC version 4 work areas. In *9th International Common Criteria Conference*, Jeju, Korea, Sept. 2008. Presentation.

[15] B. Meyer. The grand challenge of trusted components. In *25th International Conference on Software Engineering (ICSE-03)*, pages 660–667, Piscataway, NJ, May 3–10 2003. IEEE Computer Society.

[16] S. Motré and C. Téri. Using B method to formalize the Java Card runtime security policy for a Common Criteria evaluation. In *23rd National Information Systems Security Conference*, Nov. 2000.

[17] T. Ramananandro. Mondex, an electronic purse: specification and refinement checks with the Alloy model-finding method. *Formal Aspects of Computing*, 20(1):21–39, 2008.

[18] J. Rushby. Formal methods and their role in the certification of critical systems. Technical Report SRI-CSL-95-01, Computer Science Laboratory, SRI International, Menlo Park, USA, Mar. 1995.

[19] G. Schellhorn, H. Grandy, D. Haneberg, and W. Reif. The Mondex challenge: Machine checked proofs for an electronic purse. In J. Misra, T. Nipkow, and E. Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods*, volume 4085 of *Lecture Notes in Computer Science*, pages 16–31, Hamilton, Canada, Aug.21–27 2006. Springer.

[20] P. H. Schmitt and I. Tonin. Verifying the Mondex case study. In *IEEE Conference on Software Engineering and Formal Methods, SEFM*, pages 47–58. IEEE Computer Society, 2007.

[21] S. Stepney, D. Cooper, and J. Woodcock. An electronic purse: Specification, refinement, and proof. Technical monograph PRG-126, Oxford University Computing Laboratory, July 2000.

[22] Verisoft XT project summary. `http://www.verisoftxt.de/`, 2007. Accessed March 22, 2010.

[23] J. M. Voas. Certifying high assurance software. In *22nd Annual IEEE Computer Software and Applications Conference (COMPSAC)*, pages 99–105. IEEE Computer Society, 1998.

[24] K. C. Wallnau. Software component certification: 10 useful distinctions. Technical Report CMU/SEI-2004-TN-031, Carnegie Mellon University, September 2004.

[25] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* International Series in Computer Science. Prentice Hall, 1996.

[26] J. Woodcock, S. Stepney, D. Cooper, J. A. Clark, and J. Jacob. The certification of the Mondex electronic purse to ITSEC level E6. *Formal Aspects of Computing*, 20(1):5–19, 2008.