Using Argumentation to Evaluate Software Assurance Standards

Patrick J. Graydon[a,b], Tim P. Kelly[a]

[a] *University of York, Department of Computer Science, Heslington, YO10 5GH, United Kingdom*
[b] *School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden*

Abstract

Context: Many people and organisations rely upon software safety and security standards to provide confidence in software intensive systems. For example, people rely upon the Common Criteria for Information Technology Security Evaluation to establish justified and sufficient confidence that an evaluated information technology product's contributions to security threats and threat management are acceptable. Is this standard suitable for this purpose?

Objective: We propose a method for assessing whether conformance with a software safety or security standard is sufficient to support a conclusion such as adequate safety or security. We hypothesise that our method is feasible and capable of revealing interesting issues with the proposed use of the assessed standard.

Method: The software safety and security standards with which we are concerned require evidence and discuss the objectives of that evidence. Our method is to capture a standard's evidence and objectives as an argument supporting the desired conclusion and to subject this argument to logical criticism. We have evaluated our method by case study application to the Common Criteria standard.

Results: We were able to capture and criticise an argument from the Common Criteria standard. Review revealed 121 issues with the analysed use of the standard. These range from vagueness in its text to failure to require evidence that would substantially increase confidence in the security of evaluated software.

Conclusion: Our method was feasible and revealed interesting issues with using a Common Criteria evaluation to support a conclusion of adequate software security. Considering the structure of similar assurance standards, we see no reason to believe that our method will not prove similarly valuable in other applications.

Keywords: Safety standards, Security standards, Assessing standards, Assurance arguments, Common Criteria

1. Introduction

Many people and organisations depend upon software assurance standards to provide justified and adequate confidence that systems possess safety or security properties. For example, people rely on evaluations conforming to the *Common Criteria for Information Technology Security Evaluation* [1, 2, 3] to show that a hardware, software, or mixed information technology product's contributions to security threats and their management are acceptable. Software safety and security assurance standards are typically evaluated using ad hoc review. While these reviews catch some errors, we will show that they miss others. It is important to ensure that conformance with a standard justifies the confidence we place in conforming systems. Our thesis is that a more structured method of determining whether conformance with a standard supports a conclusion such as adequate safety or security is both feasible and capable of identifying issues of interest. In this paper, we contribute:

1. A discussion of the features of software assurance standards that demand evaluation more rigorous than ad hoc review provides
2. A structured method for evaluating software assurance standards
3. An evaluation of our method by case study application to the internationally-recognised Common Criteria standard

In section 2, we discuss why ad hoc review is unsuited to software assurance standards. In section 3 we discuss related work, including assurance argumentation. In section 4, we present our method, which builds upon argumentation technology. In section 5, we discuss our case study evaluation method. In section 6, we describe how we applied our method to the Common Criteria standard. In section 7, we present the results of our case study, including examples that illustrate the kinds of issues that our method identified. Finally, we discuss the significance and limitations of our findings in section 8 and conclude in section 9.

2. Software Assurance Standards Demand a Different Approach

Software standards are frequently written by standards committees and subjected to ad hoc review by interested parties. In many cases, this process serves the community well. However, software assurance standards – typically focused on safety or security – have distinctive features that make ad hoc review an unsuited to evaluating them. Table 1 illustrates these differences by comparing two

Table 1: W3C XML Versus RTCA DO-178B

	W3C XML	**RTCA DO-178B**
Purpose	Ensure compatibility	Ensure suitability for a safety-critical application
Strategy	Specify externally-visible aspects of conforming documents	Specify aspects of the software and development process
Properties	Boolean	A matter of degree
Specification	Mainly formal or semi-formal	Mainly informal
The standard is fit for purpose if	It clearly distinguishes conforming documents from non-conforming documents and achieves buy-in	Conforming software's contributions to hazards and their management are acceptable

specimen standards: the W3C Extensible Markup Language (XML) standard [4] and the RTCA DO-178B standard for airborne software [5].

The W3C XML standard aims to ensure compatibility between producers and consumers of documents. To achieve this, it specifies externally-visible attributes of conforming documents. For example, the standard specifies the syntax of tags, attributes, identifiers, and escape sequences. In contrast, RTCA DO-178B aims to ensure that airborne software is fit for use from a safety perspective. Software contributions to safety – like software contributions to security – cannot be directly measured. Accordingly, the standard instead uses an indirect approach: it specifies properties of both the software and the process used to produce it. For example, RTCA DO-178B requires developers to decompose software requirements over the software structure and conduct requirements-based functional testing that achieves a specified level of coverage [5]. (For flight-critical software, the standard requires Modified Condition/Decision Coverage [6].)

Standards like the W3C XML standard are fit for purpose if they are sufficiently clear and if there is enough interest in producing or consuming the specified thing. Ad hoc review is useful as a means of identifying vagueness (or at least failure to use formalisms where appropriate). Moreover, the reviewers' comments will identify concerns that might preclude wide adoption. In contrast, software

assurance standards are fit for purpose if conformance to them helps assessors to determine that the software's contributions to system hazards or threats and their management are adequate. It is not clear that ad hoc review is the best practical way to make this determination. Just as structured software reviews are more effective than ad hoc reviews [7], we contend that a structured review process will be a more effective way of evaluating software assurance standards than current practice.

3. Related Work

This work builds upon existing assurance argument technology. Assurance arguments have been used to show that systems are adequately safe or secure to operate. Researchers have used arguments to model standards. Researchers and practitioners have also criticised standards, including the Common Criteria standard that we used to evaluate our method. However, we are aware of no other effort to assess this or any other software safety or security standard through a process of argument capture and criticism.

3.1. Safety And Security Arguments

A safety argument is an assurance argument intended to demonstrate that a system is adequately safe to operate [8]. This argument and the evidence it references form a complete safety case. Both the argument and evidence are necessary: without evidence, the conclusion of safety is not grounded, while without argument, the evidence is not explained. Figure 1 in subsection 3.2 presents an example safety argument fragment.

In some domains, developers are required to produce safety arguments before their systems are deployed. For example, the UK Defence Standard 00-56 contains the following requirement [9, §9.1]:

> The Contractor shall produce a Safety Case for the system on behalf of the Duty Holder. The Safety Case shall consist of a structured argument, supported by a body of evidence, that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment.

Some researchers have suggested using assurance arguments to demonstrate security rather than safety [10]. While industry adopted safety arguments more broadly than security arguments, there is significant research interest in security arguments and security cases [11, 12, 13].

3.2. The Goal Structuring Notation

Assurance arguments can be recorded in a number of notations, including natural-language text. However, many people find it easier to perceive the logical structure of an assurance argument when it is rendered in graphical form. Two prominent graphical notations are commonly used to record safety arguments: the Claims Argument Evidence (CAE) notation and the Goal Structuring Notation (GSN) [14, 15].

Figure 1 presents a fragment of an example safety argument recorded in GSN. This argument is for a hypothetical parts delivery robot for use in a factory. At the top, rectangular goal element GSafety represents the main claim of the argument, namely that the parts robot is adequately safe to operate. This claim is offered in a context defined by three rounded context elements: CSystem references a description of the parts courier robot; CEnvironment defines the factory-floor operating environment; and CSafety explains what is meant by 'adequate safety' in this system.

This example argument uses a typical safety argument strategy: strategy parallelogram GHazard42AM represents an argument over identified hazards. One subgoal supporting GSafety through this strategy is GStopsShort. To show that the courier robot stops short of a human worker in its path, we cite test evidence represented by circular solution SSTesting. In a full argument, we would probably supplement this test evidence with the results of reviews, analysis, and testing of the system components (including its software).

3.3. Capturing Standards' Arguments

Other researchers have captured standards' arguments. For example, Ankrum and Kromholz modelled the arguments of three standards – RTCA DO-178B [5], ISO 14971 [16], and the Common Criteria [1, 2, 3] – as a pattern for safety and security arguments [17]. Galloway et al. modelled the argument underlying DO-178B to justify substituting formal analysis evidence in the place of the testing evidence required by the standard [18].

Researchers have also captured parts of the Common Criteria standard to serve purposes other than criticising the standard. For example, Schumacher has extracted design patterns from the Common Criteria's security functional requirements [19]. Morimoto et al. have formalised the standard's security functional requirements to facilitate formal verification [20].

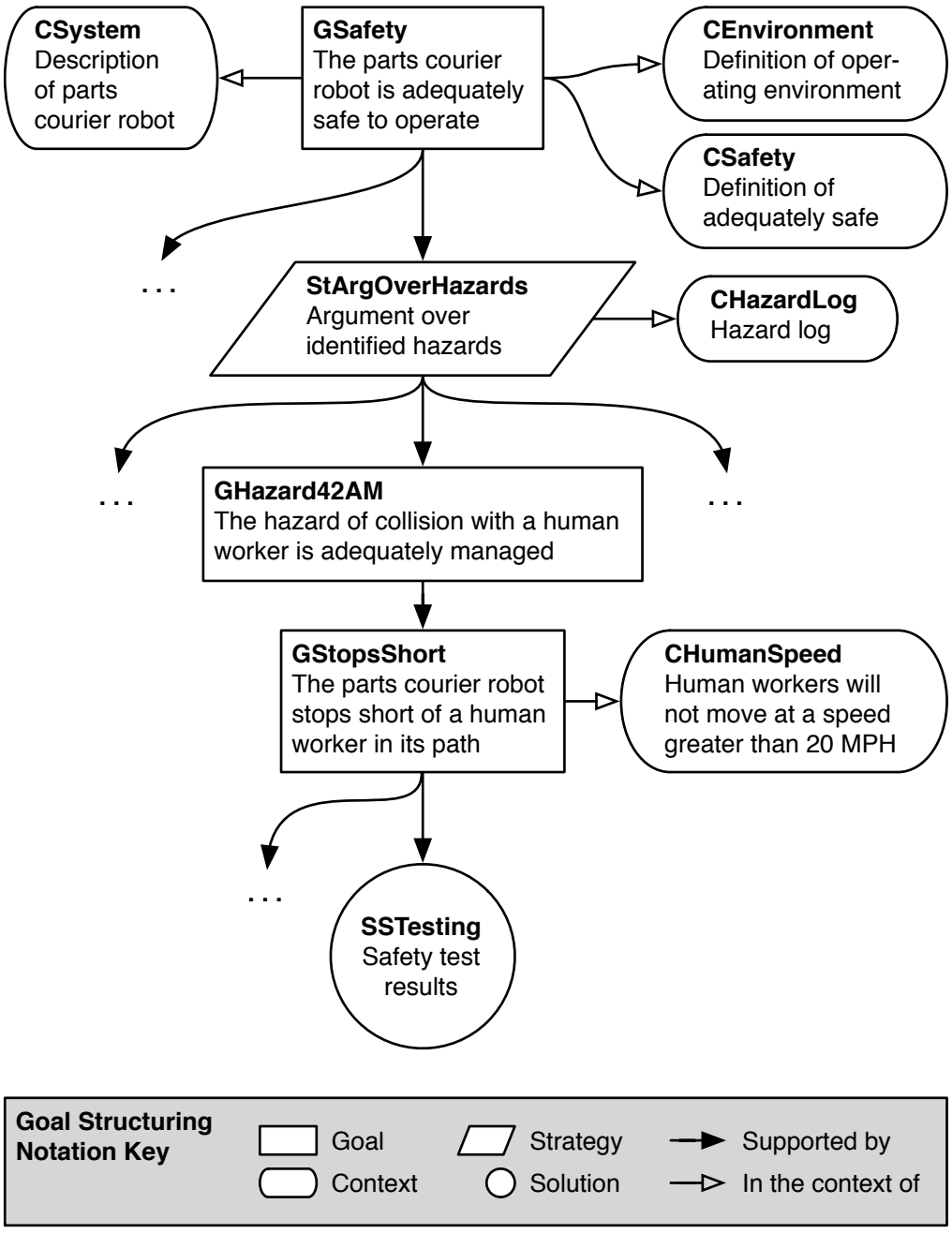

Figure 1: An Example Safety Argument Rendered In GSN

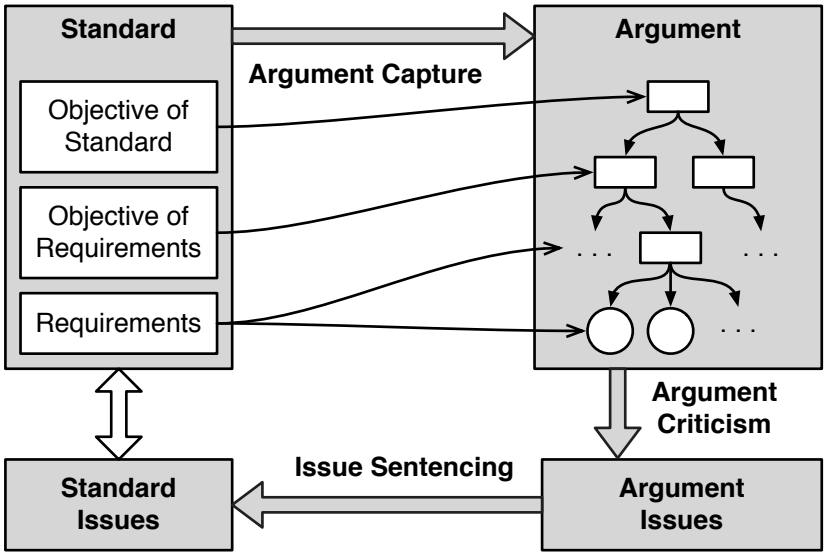

Figure 2: Method For Evaluating Software Assurance Standards

3.4. Criticism of the Common Criteria Standard

Others have criticised the Common Criteria standard that we used to evaluate our method. For example, Barnum has called for a more objective and consistent assessment of vulnerabilities [21]. We do not claim that all of the issues we report are novel. Our contribution lies instead in demonstrating the feasibility and efficacy of our method of analysing standards.

4. A Method for Evaluating Standards

Figure 2 illustrates our method for evaluating software assurance standards. The method comprises three steps: *argument capture*, *argument criticism*, and *issue sentencing*.

4.1. Argument Capture

The analyst begins by identifying the claim that the standard's users want conformance to support. For example, users of a software safety standard might want conformance to support a claim that the software's contributions to system-level hazards and their management are acceptable. This claim need not be stated in the standard's text; the objective is to evaluate the standard as it will be used. If

a standard is used in multiple ways, it must be evaluated again for each new use. The analyst records this claim as the main claim of the captured argument.

Next, the analyst identifies the standard's requirements, the evidence discussed in or implied by those requirements, and the specific claims that the requirements make of that evidence. The analysts records these as claims and evidence supporting those claims. For example, consider RTCA DO-178B's requirement for "reviews and analyses" of the source code that cover (among other things) the "use of uninitialized variables or constants" [5, §6.3.4f]. This requirement corresponds both to a claim that no variables or constants are used before they are initialised and to review or analysis evidence supporting that claim.

Finally, the analyst identifies the objectives of each group of requirements. Even when standards do not include statements of the form "the objective of {*group of requirements*} is ...", they are often divided into sections or subsections, each with explanatory text. The analyst determines what the requirements are meant to accomplish and represents this as a both a set of intermediate claims and the argument structure linking low-level claims to the main claim.

While there are many notations for recording arguments, we have used the Goal Structuring Notation (GSN) in our case study evaluation [8]. In GSN, rectangular goal elements represent claims and circular solution elements can be used to cite evidence. The analyst records the argument structure by linking each claim to the sub-claims or evidence that supports it using arrows. Figure 3 gives an example.

Throughout this process, the standard should be captured as accurately as possible. The standard's text should be used in the captured argument as much as practical, *especially* where it is confusing. Where the standard's text describes several objectives of several requirements without identifying which is satisfied by which, the argument should do likewise. Where the reasoning in the text skips steps, so should the argument. In this way, flaws in the standard become flaws in the argument so that these can be identified, reported, and redressed.

4.2. Argument Criticism

The analyst reviews the captured argument much as one would a safety argument [22]. The review proceeds in phases, each focused on a fragment of argument containing a few related reasoning steps. For each such fragment, the analyst:

1. Considers ways in which (in his or her experience) the argument might be vague or subject to misinterpretation

2. Attempts to draw out implicit assumptions
3. Judges the necessity and reasonableness of each assumption (implicit or explicit)
4. Searches the argument for well-known fallacies [23]
5. Identifies where 'independent' lines of reasoning depend upon common sub-arguments
6. Considers expected but omitted practice to see if the argument could practicably be strengthened
7. Determines whether negative experience with similar systems might provide counterevidence
8. Judges (subjectively) the strength of the argument

To determine whether the standard should required additional evidence, the analyst should rely upon the As Confident As Reasonably Practicable (ACARP) paradigm first proposed by Brian Randell. That is, analysts should first judge whether evidence is manifestly sufficient or grossly insufficient based on the risk addressed. In the broad grey area between those absolute limits, analysts should weigh the costs and benefits of additional evidence in the context of the system in question. Every useful form of evidence is obligatory unless the cost of its provision would be grossly disproportionate to the product of remaining risk and the expected gain in confidence. The ACARP paradigm is not universally recognised or practiced. However, it is the only reasonable, general-purpose means of determining how much evidence a standard should require.

It might be feasible to provide a certain form of evidence in some cases and not in others. The analyst should raise an issue wherever, in his or her judgment, provision of evidence might be reasonable in at least a large proportion of cases. Standards can be modified to accommodate the remaining cases by requiring the developers to either supply each form of evidence (or its equivalent) or show that the cost of providing it would be grossly disproportionate to the gain in confidence. Some standards already use this mechanism [9].

4.3. Issue Sentencing

Analysts might err when capturing an argument. After criticising the captured argument, the analyst re-examines each identified issue to determine what defect

in the standard, if any, it reflects. For example, if the argument uses a term without defining it, the analyst should determine whether this term is defined in the standard or its normative references. If an issue reflects any deficiency in the standard (even if it *also* reflects an argument capture error), the analyst should clarify its description as needed and then report it as a finding.

5. Case Study Assessment

Our thesis is that a structured method of determining whether conformance with a standard supports a conclusion such as adequate safety or security is both feasible and capable of identifying issues of interest. To test this, and to collect observations that would help us refine the method, we conducted a case study. The subject of our study is the internationally-recognised Common Criteria security standard [1, 2, 3]. We selected this standard for two reasons:

1. It is a subject of interest for a broad, international community
2. It is representative of the indirect approach used by software safety and security standards

5.1. Case Study Method

To determine whether our method was feasible and capable of identifying interesting issues with the assumed use of the specified standard, we counted and characterised the identified issues. Our method might be infeasible if capturing the argument forces the analyst to speculate to such a degree that issues in the argument reflect mainly erroneous speculation. To assess the noise added during argument capture, we counted issues both before and after issue sentencing. We would consider fewer than 10% of issues stemming from mis-capture to indicate excellent performance, and more than 90% to indicate that argument capture is infeasible.

Our method might be incapable of identifying interesting issues if criticising the standard's logic provided no more insight than ad hoc review of the standard's text. The Common Criteria is an established standard and has been reviewed and revised several times. As a result, we consider our method capable of revealing interesting defects if it finds any such issues in this well-reviewed standard.

6. Capturing the Common Criteria Standard

The Common Criteria is a three-part standard for evaluating the security of information technology products. Part 1 defines the security model and the terms used in the standard [1]. Parts 2 and 3 define template security requirements and their objectives [2, 3].

The Common Criteria standard is complemented by the separate *Common Methodology for Information Technology Security Evaluation* [24]. The Common Methodology "defines the minimum actions to be performed by an evaluator in order to conduct a Common Criteria evaluation, using the criteria and evaluations defined in the Common Criteria" [24, ¶3]. Its statement of scope anticipates readers using the Common Methodology in order to clarify their understanding of the Common Criteria.

Readers of a standard should seek clarification where necessary. However, standards should be written as clearly as practicable to minimise the need for such clarification. Consequently, we did not include material from the Common Methodology in our captured argument. While we discuss the Common Methodology alongside the sample issues presented in sections section 7, this discussion is for completeness only: our evaluation was of the Common Criteria standard in isolation.

6.1. The Common Criteria Security Model

The Common Criteria standard describes a security model [1, §7]. In this model, a hardware, software, firmware, or mixed *Target Of Evaluation* (TOE) must meet its security objectives by satisfying its *security functional requirements* in the context of *assumptions* about the operational environment. Separate *security assurance requirements* describe the assurance properties that the TOE, the process of its development, and/or its development artefacts must have to establish adequate confidence that the security functional requirements have been met. The TOE implements *TOE Security Functionality* (TSF) that enforces the security functional requirements. The standard also uses the term TSF to identify the portion of the TOE that implements the TSF. Access to the TSF is through the *TSF interface*.

Each TOE is evaluated against a *security target* prepared for that TOE. The security target includes [1, §A.2]:

- A description of the TOE
- A definition of the security problem to be addressed

- The security objectives for the TOE
- The security objectives for the operational environment
- A statement of the TOE's security functional requirements
- A statement of the TOE's security assurance requirements
- Common Criteria-style templates for any requirements that cannot be derived from the templates in Parts 2 and 3 of the standard

The security target and TOE are evaluated separately [1, §10]. An security target evaluation shows that an security target is sound and internally consistent and that it correctly instantiates any templates it is based upon. A TOE evaluation shows that the TOE conforms to its security target.

Some security targets are derived from a *protection profile* [1, §9]. A protection profile serves a template for security targets for a class of systems. Writing a security target that conforms to a protection profile contributes to evaluation results that enable comparisons across systems in that class. However, the use of a protection profile is optional and contributes no direct evidence of security. Accordingly, we have elected not to model or analyse the standard security assurance requirements related to protection profile evaluation (i.e. class APE).

6.2. The Common Criteria's Requirements

The Common Criteria's requirements are embodied in the template security assurance requirements presented in Part 3. These requirements are organised into the eight *classes* shown in Table 2. Each class is further divided into *families*. Each family comprises a number of individual requirements, called *elements*. The standard defines three kinds of elements:

- *Developer Action Elements* specify things that the developer must do. For example, developers must supply design documentation to the evaluators [3, ADV_TDS.1.1D].
- *Content and Presentation Elements* specify properties of development artefacts that assessors must check. For example, the design must identify all subsystems of the TSF [3, ADV_TDS.1.2C].

Table 2: Template Security Assurance Requirement Classes (From [3])

Identifier	Description
APE	"Protection profile evaluation"
ASE	"Security target evaluation"
ADV	"Development" of the TOE
AGD	"Guidance documents" for preparation and operation of the TOE
ALC	Development "life-cycle support"
ATE	"Tests" of the TOE
AVA	"Vulnerability assessment" of the TOE
ACO	"Composition"

- *Evaluator Action Elements* specify things that the evaluator must do. For example, most families require the evaluator to "confirm that the information provided meets all requirements for context and presentation of evidence" [3, ASE_INT.1.1E]. (That is, evaluators must confirm that the requirements specified by the content and presentation elements have been met.)

Many families are defined at a number of levels, with each level adding additional rigour to the level below. For example, level 6 of family ADV_FSP ("functional specification") adds rigour to the lower levels by requiring that the functional specification required by those levels *also* be presented in a formal style [3, ADV_FSP.6.2C].

Security target authors select a set of security assurance requirements for a given TOE by choosing choosing levels from a subset of the template requirements, taking care to satisfy dependency requirements. In order to make security assurance requirement choices consistent across applications (and the resulting assurance comparable), the standard includes seven pre-defined *Evaluation Assurance Levels* (EALs). Each EAL comprises a set of security assurance requirements. For example, EAL 7 includes the requirements of the ATE_FUN family ("functional tests") at level 2 [3, Table 8]. The EALs are are numbered from 1 to 7, with higher-numbered EALs meant to represent more assurance than lower-numbered EALs [3, §8.1].

Class ACO provides a mechanism for evaluating systems composed of components that have been evaluated in isolation. Because the argument for composite

Table 3: Extract From Security Assurance Requirement Family ALC_DVS (From [3])

Element	Text
ALC_DVS.1.1D	The developer shall produce and provide development security documentation
ALC_DVS.1.1C	The development security documentation shall describe all the physical, procedural, personnel, and other security measures that are necessary to protect the confidentiality and integrity of the TOE design and implementation in its development environment
ALC_DVS.2.2E	The evaluator shall confirm that the security measures are being applied

TOEs differs significantly from that of TOEs evaluated all at once, we elected not to capture class ACO. Our evaluation thus excludes that part of the standard.

6.3. Capturing the Common Criteria Argument

We identified the standard's discussion of its security model (presented in Chapter 7 of Part 1 [1]) as indicative of the claims that users of the standard would like to be able to make. We synthesised the main claim of our captured argument from Figure 3 of Part 1 of the standard. This claim reads:

> The TOE correctly implements countermeasures sufficient to minimise risk to assets.

For each developer action element or evaluator action element requiring production of an artefact, we captured a solution representing that artefact. For example, Figure 3 shows the argument fragment we captured from family ALC_DVS ("development security"), which is summarised in Table 3. The solution SDSDoc represents the artefact introduced by element ALC_DVS.1.1D.

For each evaluator action element requiring confirmation of a specific property, we captured a goal solved by the developer's report. For example, GDSPApplied and solution SERonSV represent element ALC_DVS.2.2E.

For each content and presentation element dictating a property of an artefact, we captured a goal solved by the relevant artefact. For example, goal GDSPComplete and solution SDSDoc represent element ALC_DVS.1.1C.

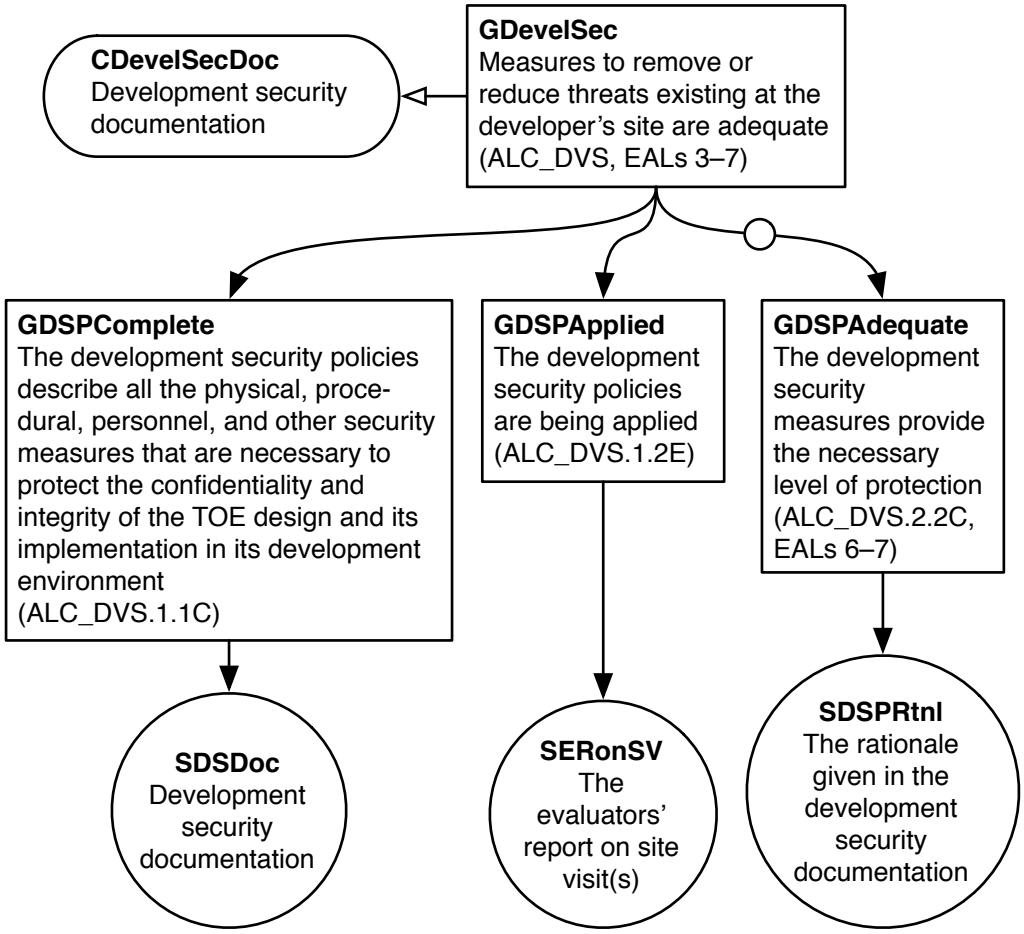

Figure 3: Argument Fragment Capturing Family ALC_DVS

We captured each standard's objectives as intermediate goals. The objectives statements for some standards do not explicitly name an objective. For example, the objectives statement of family ALC_DVS reads [3]:

> Development security is concerned with physical, procedural, personnel, and other security measures that may be used in the development environment to protect the TOE and its parts. It includes the physical security of the development location and any procedures used to select development staff.

In such cases, we derived goal text either from other text in the standard or paraphrased our understanding of the objectives. For example, the text of goal GDevelSec is taken from a sentence in the application notes for family ALC_DVS that reads, "this family deals with measures to remove or reduce threats existing at the developer's site" [3, ¶373].

We modelled modelled the security assurance requirements included in all seven EALs as shown in Figure 3. Annotations in parenthesis describe the EALs at which parts of the argument apply. Circle annotations like the one between goal GDevelSec and subgoal GDSPAdequate indicate subgoals that do not apply at the same EALs as the goals they solve. We did not model requirements such as family ALC_FLR and class ACO that do not appear in any EAL.

6.4. The Captured Argument Is Not a Security Argument

The argument we captured is not a security argument. A security argument would focus on a single TOE and include details such as security objectives, security functional requirements, and product design features. Ideally, a security argument would be organised so as to trace each threat to security objectives, security functional requirements, and finally to evidence. This organisation permits more attention to be paid to the parts of the argument that concern the greatest threats.

Because the argument captured from a standard is not a safety or security argument, it seems to focus on procedural rather than technical aspects. For example, consider our captured argument's treatment of testing. Figure 4 presents part of argument we captured from class ATE. Because the Common Criteria standard is not TOE-specific, the captured argument argues *about* the testing procedure rather than over test results.

The captured argument's focus on procedural rather than technical details makes it ill-suited to be a specific system's security argument. However, this focus is precisely what is needed to facilitate criticising a standard that will be applied to many systems. The details of the procedures used are precisely what determines the strength of conclusions about a system's properties.

6.5. Criticising the Common Criteria Argument

We reviewed the Common Criteria argument using the process described in subsection 4.2, with one exception. The analyst (Graydon) has a general background in high-integrity systems development but cannot claim a particular expertise in software security issues. As a result, he could not systematically exam-

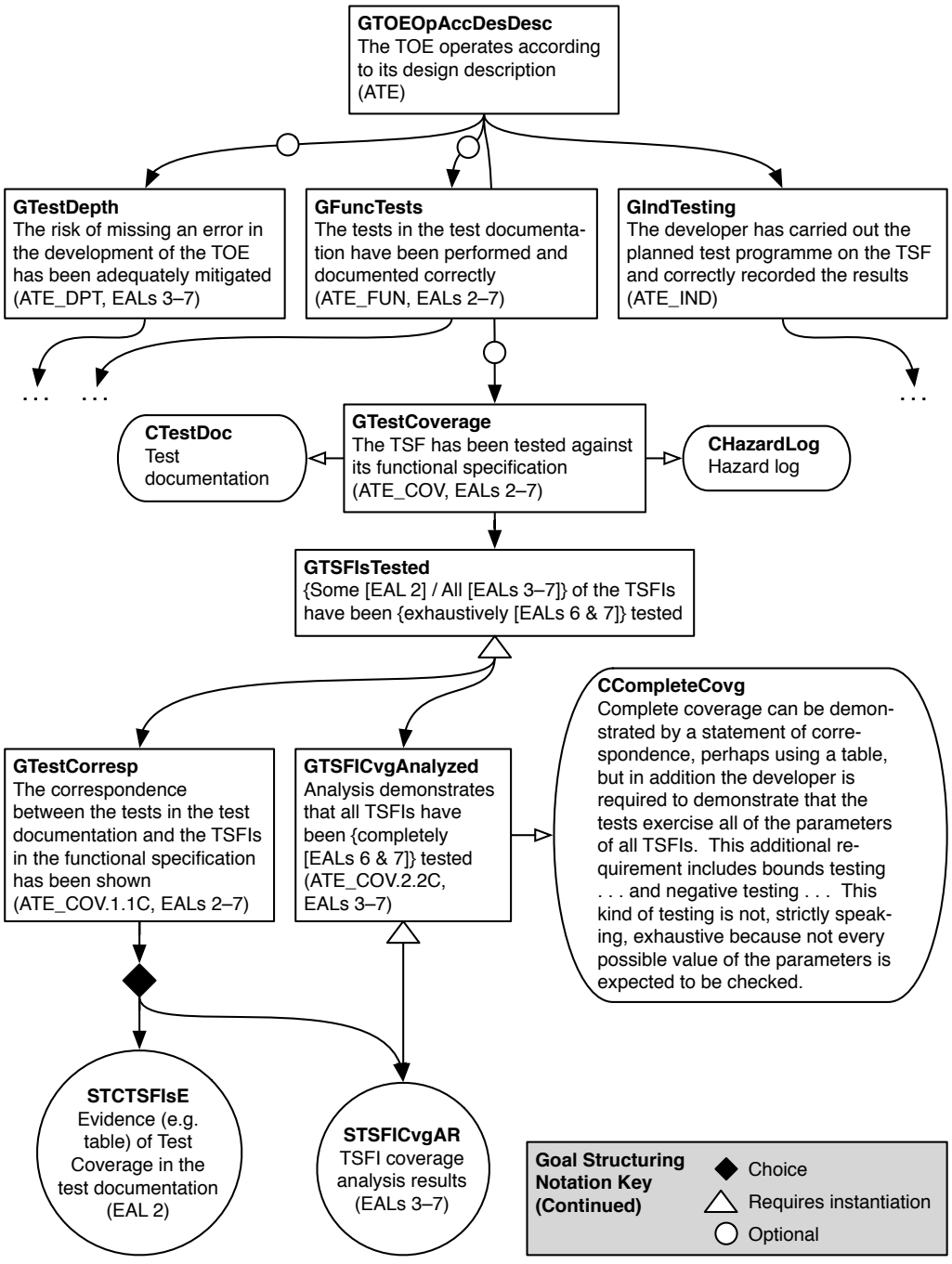

Figure 4: Argument Fragment Capturing Part Of Class ATE

Table 4: Classification of Issues Found

Missing or inadequate evidence	60
Misleading or inadequate explanation	28
Other issues (mainly vagueness)	33
TOTAL	121

ine the argument for ways in which negative experience might call its claims into question.

7. Results of the Case Study

During argument criticism, we identified 173 issues. We discarded 52 (30%) of these during issue sentencing, suggesting that a significant but not overwhelming amount of noise was added during argument capture.

The following subections present 15 examples selected from the remaining 121 issues. We consider these issues among the most serious we found. We know of no objective metric for security standard issues. As a result, we cannot prove that these are "interesting". We present these examples so that readers can form their own opinions.

We describe each issue and suggest a potential improvement to the standard. We do not claim to suggest the best improvements practicable: identifying an issue requires showing that it *could* be improved, not determining how best to improve it. We present these potential improvements mainly to illustrate the issues we found but also in the hope that users of the Common Criteria standard might find them useful.

We did not precisely measure the time needed to evaluate the standard. Nevertheless, the case study activities were conducted by one researcher over several weeks. Considering the importance of standards, such effort is not prohibitive.

7.1. Example Issues of Missing or Inadequate Evidence

Table 4 presents a breakdown of the 121 issues we found. Of these, 60 are issues of missing or inadequate evidence. That is, either: (a) the reasoning in the standard's argument is invalid because it omits an entire premise by failing to demand appropriate evidence; or (b) providing additional evidence could practicably increase confidence for systems of the criticality signified by each EAL. In this section, we present ten example issues of missing or inadequate evidence.

7.1.1. Issue: System-Level Testing With Requirements Coverage

The Common Criteria does not require system-level testing that complements lower-level testing. The standard requires testing evidence showing that the TOE satisfies its specification [3, ATE_COV.1.1C]. It also requires the evaluator to determine (using an unspecified procedure) that "the functional specification is an accurate and complete instantiation of the [security functional requirements]" [3, ADV_FSP.1.2E]. These forms of evidence, taken together, support a claim that the security functional requirements are satisfied. However, indirect support gives rise to uncertainty: if either the testing *or* the specification evaluation is flawed, a defective implementation might pass unnoticed.

Potential Improvement: The Common Criteria standard should require software testing at the unit, integration, and system levels. Standards for safety-critical software typically this evidence because each form imperfect: traceability and specification review evidence might miss errors, unit testing is based on derived requirements that might be incorrect, and system tests typically exercise fewer paths through the software than unit tests do. Together, each form compensates for the others' weaknesses.

7.1.2. Issue: Adequacy of Structural Coverage

The Common Criteria does not require evidence of structural (code) coverage in the case of software TOEs. As the standard notes, it is imperative that functional tests exercise the implementation's internals [3, ¶415]:

> The objective [of test depth requirements] is to counter the risk of missing an error in the development of the TOE. Testing that exercises specific internal interfaces can provide assurance not only that the TSF exhibits the desired external security behaviour, but also that this behaviour stems from correctly operating internal functionality.

However, the Common Criteria requires only minimum coverage of the *design*. At EALs 3–7, functional tests must cover "all subsystems in the TOE design" [3, ATE_DPT.1.2C]. At EALs 5–7, analysis must also show that "all TSF modules in the TOE design have been tested" [3, ATE_DPT.3.3C]. There is no reason to believe that tests covering subsystems or modules would in all cases adequately exercise all parts of the code implementing those subsystems or modules.

Potential Improvement: The Common Criteria standard should require analysis of structural coverage of software tests and prescribe minimum coverage levels at

each EAL. Developers of software in safety applications routinely analyse structural coverage. For example, DO-178B calls for testing of flight-critical software that achieves modified condition / decision coverage [5, 6]. Even in the case of less-critical software, testing must achieve statement coverage [5]. Automated tools for analysing test suite structural coverage are readily available [25, 26]. Demonstrably achieving basic levels of coverage – such as statement coverage – should be practical even in less-critical applications (e.g. EAL 3).

7.1.3. Issue: Adequacy of Formal Proofs

The Common Criteria requires no evidence of the adequacy of formal proofs. At EAL 7, the standard requires developers to prove that the TOE design refines its formal specification [3, ADV_TDS.6.10C]:

> The proof of correspondence between the formal specifications of the TSF subsystems [given in the design documentation] and of the functional specification shall demonstrate that all behaviour described in the TOE design is a correct and complete refinement of the [TSF interface] that invoked it.

Seemingly-simple hand-written and checked proofs might contain errors. Humans or software tools might mistranslate formal representations. Software proof checking tools might be imperfect [27]. In an informative appendix, standard suggests using tools when hand proofs would be "long winded and incomprehensible" [3, ¶605]. However, it does not require or even advise evaluating whether any proof checker or translation tool is fit for use.

Potential Improvement: The Common Criteria standard should require formal proofs to be checked using an automated proof checker. The standard should also require evidence that any tools used are fit for purpose.

7.1.4. Issue: Sufficiency of the Security Assurance Requirements

The Common Criteria requires no evidence that security assurance requirements are adequate to demonstrate acceptable security. Other assurance standards such as DO-178B prescribe a package of assurance requirements based on the consequences of a failure [5]. In contrast, the Common Criteria takes a more goal-oriented approach. Security target authors first select a subset of the template requirements captured in our argument and augment these with any number of custom requirements. The authors then write a rationale explaining these choices [3, ASE_REQ.2.8C]. However, this explanation need not be compelling.

As the Common Methodology interprets the standard, "any explanation is correct, as long as ... [there are no] obvious inconsistencies with the remainder of the [security target]" [24, ¶305].

One could take the view that those deploying the TOE, not those evaluating it, should judge the adequacy of the security assurance requirements. After all, TOEs might be built for a range of applications and only users know the risks in each application. However, the standard requires evaluation of a security target that must contain a rationale for the assurance requirements. This creates the (false) impression that the assurance requirements have been judged appropriate for applications similar to those described by the target.

Potential Improvement: The Common Criteria standard should require evidence that the chosen security assurance requirements provide as much confidence as is reasonably practicable. To provide this evidence, the evaluators might review the rationale to determine whether adding a security assurance requirement from Part 3 of the standard would practicably justify greater confidence. If developers supplied the rationale in the form of a structure argument, it could be criticised using a process similar to that presented in subsection 4.2.

7.1.5. Issue: Correctness of the Security Policy Model

The Common Criteria does not require evidence sufficient to show that the formal security policy model is correct. This model, required at EALs 6 and 7, is meant to formalise a part of the security functional requirements [3, ¶269]:

> Inadequacies in a TOE can result ... from a failure in understanding the [security functional requirements] ... Throughout the design, implementation, and review processes, the modelled security requirements may be used as precise design and implementation guidance, thereby providing increased assurance that the modelled security requirements are satisfied by the TOE. The precision of the model and resulting guidance is significantly improved by casting the model in a formal language and verifying the security requirements by formal proof.

The developer must "identify the relevant portions of the statement of security functional requirements that make up" each modelled policy [3, ADV_SPM1.2D]. However, the standard requires no evidence that the model both correctly encodes the identified requirements.

Potential Improvement: The Common Criteria standard should require evidence showing that the security policy model correctly models the selected security functional requirements. The standard should also require evidence showing that the security policy model covers all portions of the security functional requirements amenable to such modelling. Moreover, if developers are to use the security policy model throughout development, the standard should require evidence showing that they have done so.

7.1.6. Issue: Clarity of the Security Problem Definition

The Common Criteria requires no evidence that the security problem definition is clear. Clarity is essential because differing interpretations of the security problem might lead to misplaced confidence in a purported solution. Perhaps recognising this, the standard sets clarity as an objective (emphasis ours) [3, ¶167]:

> Evaluation of the security problem definition is required to demonstrate that the security problem intended to be addressed by the TOE and its operational environment, [sic] is *clearly* defined.

The evaluator is required to confirm that the security problem description contains specified elements. For example, the security problem definition must "describe the assumptions about the operational environment of the TOE" [3, APE_SPD.1.4C]. However, the evaluator is not required to analyse the clarity of the statement.

Potential Improvement: The Common Criteria standard should require evaluators to confirm that the security problem statement is acceptably clear. The standard's writers might have chosen not to require a subjective judgement that might be inconsistent across reviewers. However, imperfect evidence is more convincing than no evidence.

7.1.7. Issue: Appropriateness of the Architecture

The Common Criteria provides no guidance on how completely the security-related portion of the TOE must "protec[t] itself from tampering" and "preven[t] bypass of the security functional requirement-enforcing functionality" [3, ADV_ARC.1.4C, ADV_ARC.1.5C]. These properties of the architecture are crucial: "without a sound architecture, the entire TOE functionality would have to be examined" [3, ¶216]. However, while these properties are a matter of degree, the standard is written in absolute terms.

To illustrate this problem, suppose that a cryptographic function computed by a smart card chip must remain secret. Is hiding the implementation under obscuring layers of silicon enough? Given that researchers have reverse-engineered chips by shaving layers off the chip and photographing each [28], is obfuscation of the transistor-level design required?

Potential Improvement: The Common Criteria standard might address this issue by dictating an evaluation process, thus operational defining the required degree. Additionally or alternatively, the standard could require the use of well-understood, 'best practice' architectural solutions wherever practicable.

7.1.8. Issue: The Implementation Representation Must Match the TOE

The Common Criteria does not require complete evidence that the TOE was generated from the given implementation representation (e.g. source code). It is imperative that these artefacts correspond; if they do not, evidence based on testing, analysis, or review of the implementation representation tells us nothing about the delivered TOE.

At EALs 4–7, the implementation representation must "define the TSF to a level of detail such that [it] can be generated without further design decisions" [3, ADV_AIMP.1.1C]. However, knowing that an implementation *can* be generated without further design decisions is not the same as knowing that it *was*.

At EALs 3–7, the implementation representation must "describe how the [configuration management] system is used for the development of the TOE" [3, ALC_CMC.3.5C]. It is not clear that such a description would adequately detail how the TOE was produced from the implementation representation.

At EALs 4–7, the configuration management system must "support the production of the TOE by automated means" [3, ALC_CMC.4.5C]. Because developers might fail to follow directions, we agree that build processes should automated to the degree practicable. However, knowing that the configuration management system *supports* automated production is not the same as knowing that the production *was* automated.

At EALs 6 and 7, the evaluator must "determine that the application of the production support procedures results in a TOE as provided by the developer for testing activities" [3, ALC_CMC.5.2E]. However, the evaluator might overlook subtle differences.

Potential Improvement: The Common Criteria standard should require evidence showing that a given software TOE was produced from given source code. This

evidence could be provided practicably, even at lower EALs. For example, developers could automate build and source control processes and collect build logs. Where a build step is difficult or impossible to automate, developers could initial steps on a detailed checklist.

7.1.9. Issue: Adequacy of Implementation Guidelines

The Common Criteria requires no evidence showing that implementation guidelines are fit for purpose. Developers must identify and use implementation guidelines at EALs 5–7, presumably to help forestall the introduction of defects. In an application note, the standard defines acceptable implementation guidelines [3, ¶398]:

> Implementation guidelines may be accepted as an implementation standard if they have been approved by some group of experts (e.g. academic experts, standards bodies). Implementation standards are normally public, well accepted and common practise in a specific industry, but developer-specific implementation guidelines may also be accepted as a standard; the emphasis is on the expertise.

Despite presenting this definition, the standard does not require evidence that the implementation guidelines were developed by experts. Moreover, a group of experts might define a standard that does not minimise or even markedly reduce implementation defects [29, 30].

Potential Improvement: The Common Criteria standard should require the use of implementation guidelines that preclude software practices that are known to be risky. These include, but are not limited to [31, 32]:

- The use of implementation-defined code constructs
- Unchecked array accesses
- The use of unfiltered input in SQL queries

7.1.10. Issue: Evidence of Tool Correctness

The Common Criteria does not require adequate evidence showing that development tools are fit for purpose. A defective tool might either introduce a security defect or fail to detect one. Recognising this, one of the standard's objectives is to ensure that tools "used to develop, analyse and implement the TOE" are not "ill-defined, inconsistent or incorrect" [3, ¶394]. To achieve this, it requires tools

used at EALs 4–7 to be "well-defined" [3, ALC_TAT.1.1C]. The standard defines well-defined tools as [3, ¶396]

> tools that are clearly and completely described. For example, programming languages and computer aided design (CAD) systems that are based on a standard published by standards bodies are considered to be well-defined.

The standard also requires the documentation of each tool to "unambiguously define the meaning of all statements as well as all conventions and directives used in the implementation" [3, ALC_TAT.1.2C]. This requirement "is especially applicable to programming languages so as to ensure that all statements in the source code have an unambiguous meaning" [3, ¶397].

This evidence shows that a tool's inputs and outputs are unambiguous. However, it cannot show that the tool produces the correct output for each given input.

Potential Improvement: The Common Criteria standard should require evidence that each tool is fit for the purpose for which it is used. The strength of this evidence should be commensurate with the risk posed by a tool failure.

7.2. Example Issues of Missing or Inadequate Explanation

Twenty-eight of the issues we found are issues of misleading or inadequate explanation of the purpose of an item or items of evidence. That is, the standard's objectives statements present a claim about what the evidence shows that is either: (a) broader than what the evidence shows; or (b) narrower than what the argument as a whole requires. In this section, we present three example issues of missing or inadequate explanation.

7.2.1. Issue: Objectives for the Functional Specification

The Common Criteria does not clearly state the properties that must be established by analysis of the functional specification. The standard defines the following objectives for the functional specification [3, ¶225]:

> [The functional specification] provides assurance directly by allowing the evaluator to understand how the TSF meets the claimed security functional requirements. It also provides assurance indirectly . . . :
>
> - The [TSF interfaces] may be used to gain better understanding of how the TSF is protected against corruption . . . and/or bypass;

- [As] an important input for ... testing;
- [As an input] to search for vulnerabilities.

The Common Criteria requires more attention to the TSF than to the remainder of the TOE. Given this strategy, it is crucial to identify all interfaces to security functionality as such. It is also crucial to establish that the specified behaviour implements the security functional requirements.

Potential Improvement: The Common Criteria standard should more clearly specify the objectives of analysis of the functional specification.

7.2.2. Issue: Objectives of Testing

The Common Criteria does not accurately describe the objectives of testing. The standard defines the objective of testing as "confirmation that the TSF operates according to its design descriptions" [3, ¶401]. This objective is misleading because testing is based on the functional specification, not (solely) the design description artefact. This objective is also insufficient. As described in subsubsection 7.1.1, testing should confirm that the TOE meets its high-level security functional requirements.

Potential Improvement: The Common Criteria standard should more clearly specify the objectives of testing.

7.2.3. Issue: Objective of "Exhaustive" Testing

The Common Criteria specifies "exhaustive" testing as an objective but explicitly does not require it. At EALs 6 and 7, one objective of testing is "to confirm that the developer performed exhaustive tests of all interfaces in the functional specification" [3, ¶411]. However, the standard requires only that these interfaces be "completely" tested [3, ATE_COV.3.2C]. The standard's application notes clarify (emphasis in the original):

> The developer is required to show how tests in the test documentation correspond to all of the [TSF interfaces] in the functional specification. This can be achieved by a statement of correspondence, perhaps using a table, but in addition the developer is required to demonstrate that the tests exercise all of the parameters of all [interfaces]. This additional requirement includes bounds testing ... and negative testing ... This kind of testing is not, strictly speaking, *exhaustive* because not every possible value of the parameters is expected to be checked.

Potential Improvement: The Common Criteria standard should clearly specify the coverage that testing is required to achieve. The text in the application notes is a good start, but the text of the element itself must be clarified to call for this coverage, rather than a coverage that cannot be achieved practically.

7.3. Example Other Issues (e.g. Issues of Vagueness)

We could not classify 33 of the 121 issues we found as either issues of missing or inadequate evidence or issues of misleading or inadequate explanation. Most of these 33 issues are issues of vagueness in the standard's language. We also identified a questionable assumption and text with an intended meaning that contradicts its literal meaning. In this section, we present two example example issues classified as "other".

7.3.1. Issue: "Tested" Is Undefined

The Common Criteria does not precisely define the test coverage required at EALs 3–5. At these EALs, "analysis of the test coverage shall demonstrate that all [TSF interfaces] in the functional specification have been tested" [3, ATE_COV.-2.2C]. However, no definition of "tested" is supplied or referenced. The Common Methodology interprets "tested" in the least-rigorous possible way: "All [TSF interfaces] that are described in the functional specification have to be present in the test coverage analysis and mapped to tests in order for completeness to be claimed" [24, ¶1275]. In many EAL 3–5 applications, additional testing rigour would practicably provide increased confidence.

Potential Improvement: The standard should be revised to clarify the test coverage required at EALs 3–5.

7.3.2. Issue: "Focused" and "Methodical" Vulnerability Analysis

The Common Criteria uses undefined terms to describe the rigour required with which the evaluator must analyse the TOE for vulnerabilities. At EALs 2 and 3, the evaluator must perform [3, AVA_VAN.2.3E]

> an independent vulnerability analysis of the TOE using the guidance documentation, functional specification, TOE design and security architecture description to identify potential vulnerabilities in the TOE.

At EAL 5, the analysis must also be "focused" [3, AVA_VAN.3.3E]. At EALs 6 and 7, the analysis must be "methodical" rather than focused [3, AVA_VAN.4.3E]. The standard does not define either term. Without a precise definition of these

terms, it is impossible to know how much confidence a focused or methodical analysis would inspire.

The standard similarly fails to define the terms it uses to define attack potentials. After conducting the vulnerability analysis, evaluators must "conduct penetration testing based on the identified potential vulnerabilities to determine that the TOE is resistant to attacks" by an attacker possessing "Basic" attack potential at EALs 1–3, "Enhanced-Basic" attack potential at EAL 4, "Moderate" attack potential at EAL 5, and "High" attack potential at EALs 6 and 7 [3, AVA_VAN.1.3E, etc.]. Neither the text of ADV_VAN nor Part 1 defines "resistant" or "Basic", "Enhanced-Basic", "Moderate", or "High" attack potential.

Potential Improvement: Lack of expertise in vulnerability analysis precludes us from suggesting better criteria for vulnerability analysis. However, it might be helpful to require analysts to have appropriate experience with similar systems. It might also be helpful to require the analyst to consider the TOE from appropriate perspectives at multiple levels of abstraction. Definitions for "Basic", "Enhanced-Basic", "Moderate", and "High" attack potential should be provided.

8. Discussion

The Common Criteria standard has already been subjected to multiple rounds of ad hoc review and revision. As a result, our study is not a head-to-head comparison of the efficacy of ad hoc review and our structured method. Instead, our study should be interpreted as showing that our method can find issues that remain even after multiple rounds of review and revision conducted using current practices.

No review process is perfect, and we do not claim that analysts using our method will never report spurious issues or fail to report real ones. For instance, an analyst who is not familiar with a particular practice might fail to call for its use. In this respect, our analyst's relative lack of security experience should have made him less likely to find interesting issues, not more likely. Analysts might also err in judgments of cost, feasibility, or the confidence inspired by particular evidence. However imperfect our method, our experience shows that it can find issues that existing techniques have apparently missed.

While we have evaluated our method on only one standard, we aim for it to be applicable to standards with the characteristic approach used by software safety and security standards. To determine whether other standards would be amenable to our argument capture process, we briefly examined the structure of RTCA DO-178B [5] and the IEC 61508 Part 3 software safety standard [33]. Both RTCA

DO-178B and IEC 61508 describe in detail the evidence that they require. However, while IEC 61508 goes into detail about the significance of this evidence, RTCA DO-178B says comparatively less about this than either IEC 61508 or the Common Criteria. For example, RTCA DO-178B requires configuration management to ensure that changes are "recorded, approved, and implemented", but does not say *why* they must be [5, §7.1e]. This might force the analyst capturing its argument to make reasoned guesses about the purpose of evidence. Others have reported extracting the argument from this standard, suggesting that this problem is surmountable [18]. Moreover, the lack of explanation should itself be viewed as an issue requiring redress. Indeed, one might hypothesise that this lack of explanation is one of the reasons why RTCA DO-178B is viewed as a standard that requires much interpretation by developers and assessors [17].

9. Conclusion

We have defined a method for assessing software safety and security standards by capturing and criticising their arguments. To demonstrate the feasibility and efficacy of this method, we have conducted a case-study application of it to the Common Criteria standard. Our results indicate both that the method is feasible and that it can reveal significant issues in a standard that has already been subjected to several rounds of ad hoc review. Some of the 121 issues that we identified negatively affect the confidence that can be justified by conformance to the standard.

We do not claim that the Common Criteria is without merit or utility. It might be argued that experienced evaluators would apply a standard correctly despite its flaws. However, the issues identified both highlight areas in which the standard might be improved and aid judgment of how much confidence is justified by evaluations following the standard as written.

We hope that future versions of the Common Criteria will include both revised security assurance requirement templates and EALs that address evidence shortfalls and clarified objective statements. However, users of the standard need not wait. Extended elements can be used to mandate the missing evidence before the standard is revised.

Acknowledgement

We thank John Knight for inspiring this work. We thank Howard Chivers, Karsten Nohl, and the anonymous reviewers for their constructive criticism. We

also thank the Swedish Foundation for Strategic Research (SSF) for supporting this work as part of the SYNOPSIS project.

References

[1] CCMB-2009-07-001, Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and general model, Version 3.1, Revision 3, Final, Common Criteria, 2009.
URL http://www.commoncriteriaportal.org/cc/

[2] CCMB-2009-07-002, Common Criteria for Information Technology Security Evaluation, Part 2: Security functional components, Version 3.1, Revision 3, Final, Common Criteria, 2009.
URL http://www.commoncriteriaportal.org/cc/

[3] CCMB-2009-07-003, Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1, Revision 3, Final, Common Criteria, 2009.
URL http://www.commoncriteriaportal.org/cc/

[4] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (fifth edition) [online] (November 2008).

[5] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, RTCA, Inc., Washington, DC, USA, 1992.

[6] K. J. Hayhurst, D. S. Veerhusen, J. J. Chilenski, L. K. Rierson, A practical tutorial on modified condition / decision coverage, Technical Memorandum TM-2001-210876, NASA, Hampton, VA, USA (May 2001).
URL http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20010057789_2001090482.pdf

[7] J. C. Knight, E. A. Myers, An improved inspection technique, Communications of the ACM 36 (11) (1993) 51–61. doi:10.1145/163359.163366.

[8] T. Kelly, R. Weaver, The Goal Structuring Notation — a safety argument notation, in: Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases, 2004.
URL http://www-users.cs.york.ac.uk/tpk/dsn2004.pdf

[9] Defence Standard 00-56, Safety Management Requirements for Defence Systems, Issue 4, Part 1: Requirements, Ministry of Defence, UK, 2007.

[10] S. Lautieri, D. Cooper, D. Jackson, SafSec: Commonalitites between safety and security assurance, in: Proc. of the 13th Safety-critical Systems Symposium (SSS), Springer-Verlag, Southampton, UK, 2005, pp. 66–75.

[11] R. Alexander, R. Hawkins, T. Kelly, Security assurance cases: Motivation and the state of the art, issue 1.1, Technical Report CESG/TR/2011/1, University of York, York, UK (April 2011).

[12] J. L. Vivas, I. Agudo, J. Lopez, A methodology for security assurance-driven system development, Requirements Engineering 16 (1) (2011) 55–73. doi:10.1007/s00766-010-0114-8.
URL http://www.nics.uma.es/biblio/citekey/vivas2010

[13] J. Goodenough, H. Lipson, C. Weinstock, Arguing security — creating security assurance cases, Electronic document: https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance/643-BSI.html (2007).

[14] P. G. Bishop, R. E. Bloomfield, A methodology for safety case development, in: F. Redmill, T. Anderson (Eds.), Industrial Perspectives of Safety-critical Systems: Proc. of the 6th Safety-Critical Systems Symposium, Springer-Verlag, Birmingham, UK, 1998, pp. 194–202.
URL http://www.adelard.com/papers/sss98web.pdf

[15] K. Attwood, et al., GSN Community Standard Version 1, Origin Consulting Limited, York, UK, 2011.
URL http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf

[16] ISO 14971:2007, Medical devices — Application of risk management to medical devices, International Organization for Standardization, 2007.

[17] T. S. Ankrum, A. H. Kromholz, Structured assurance cases: Three common standards, Proc. of the 9th Int'l Symposium on High-Assurance Systems Engineering (HASE)doi:10.1109/HASE.2005.20.

[18] A. Galloway, R. Paige, N. Tudor, R. Weaver, I. Toyn, J. McDermid, Proof vs testing in the context of safety standards, in: Proc. of the 24th Digital

Avionics Systems Conference (DASC), 2005, pp. 10.E.1–10.E.14. doi:10.1109/DASC.2005.1563405.

[19] M. Schumacher, Security patterns and security standards, in: A. O'Callaghan, J. Eckstein, C. Schwanninger (Eds.), Proc. of the 7th European Conference on Pattern Languages of Programms (EuroPLoP), Universitaetsverlag Konstanz (UVK), 2002, pp. 289–300.

[20] S. Morimoto, S. Shigematsu, Y. Goto, J. Cheng, Classification, formalization and verification of security functional requirements, in: V. Geffert, J. Karhumäki, A. Bertoni, B. Preneel, P. Návrat, M. Bieliková (Eds.), SOFSEM 2008: Theory and Practice of Computer Science, Vol. 4910 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 622–633.
URL http://dx.doi.org/10.1007/978-3-540-77566-9_54

[21] S. Barnum, Refining the Common Criteria for objective and predictable security measurement, Presentation to Software Assurance Working Group Session (2010).
URL https://buildsecurityin.us-cert.gov/swa/wg_presentations_dec2010/sean%20barnum%20ISO%2020004%200verview%20-%20SwA%20Working%20Groups%2012-2010%20(Barnum)x.pdf

[22] P. Graydon, J. Knight, M. Green, Certification and safety cases, in: Proc. of the 28th International Systems Safety Conference (ISSC), Minneapolis, MN, USA, 2010.
URL http://www.cs.virginia.edu/~jck/publications/ISSC.2010.pdf

[23] W. Greenwell, J. C. Knight, C. M. Holloway, J. J. Pease, A taxonomy of fallacies in system safety arguments, in: Proc. of the 24th International System Safety Conference (ISSC), Albuquerque, NM, USA, 2006, pp. 430–439.
URL http://www.cs.virginia.edu/~cmh7p/paper-issc06-fallacies-as-printed.pdf

[24] CCMB-2009-07-004, Common Methodology for Information Technology Security Evaluation, Version 3.1, Revision 3, Common Criteria, 2009.
URL http://www.commoncriteriaportal.org/cc/

[25] Gcov intro – using the GNU Compiler Collection (GCC) [online, cited March 2012].

[26] Rapicover | Rapita Systems [online, cited July 2012].

[27] PVS bugs list [online, cited April 2012].

[28] K. Nohl, D. Evans, S. Plötz, H. Plötz, Reverse-engineering a cryptographic RFID tag, in: Proc. of the USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 2008, pp. 185–193.
URL http://www.cs.virginia.edu/~evans/pubs/usenix08/usenix08.pdf

[29] L. Hatton, Safer language subsets: an overview and a case history, MISRA C, Information and Software Technology 46 (7) (2004) 465–472. doi:10.1016/j.infsof.2003.09.016.

[30] C. Boogerd, L. Moonen, Assessing the value of coding standards: An empirical study, in: Proc. of the IEEE International Conference on Software Maintenance (ICSM), 2008, pp. 277–286. doi:10.1109/ICSM.2008.4658076.

[31] 2011 CWE/SANS top 25 most dangerous software errors, Web page (November 2011).
URL http://cwe.mitre.org/top25/

[32] International Organization for Standardization, Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use, Technical Report 24772:2010, ISO/IEC (2010).

[33] IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems — Part 3: Software requirements, International Electrotechnical Commission (IEC), 2008.
URL http://webstore.iec.ch/Webstore/webstore.nsf/Artnum_PK/43984