

Applying *Practical* Formal Methods to the Specification and Analysis of Security Properties

Constance Heitmeyer

Naval Research Laboratory (Code 5546)
Washington, DC 20375 USA
heimtaylor@itd.nrl.navy.mil
<http://chacs.nrl.navy.mil/SCR>

Abstract. The SCR (Software Cost Reduction) toolset contains tools for specifying, debugging, and verifying system and software requirements. The utility of the SCR tools in detecting specification errors, many involving safety properties, has been demonstrated recently in projects involving practical systems, such as the International Space Station, a flight guidance system, and a U.S. weapons system. This paper briefly describes our experience in applying the tools in the development of two secure systems: a communications device and a biometrics standard for user authentication.

1 Introduction

In 1978, the requirements document for the flight program of the A-7 aircraft [13,14] introduced a special tabular notation for writing specifications. Part of the SCR (Software Cost Reduction) requirements method, this notation was designed to document the requirements of real-time, embedded systems concisely and unambiguously. During the 1980s and 1990s, SCR tables were used by several organizations in industry and government, e.g., Grumman [19], Bell Laboratories [15], Ontario Hydro [21], the Naval Research Laboratory [7], and Lockheed [5], to document the requirements of many practical systems, including a submarine communications system [7], the shutdown system for the Darlington nuclear power plant [21], and the flight program for Lockheed's C-130J aircraft [5].

While human effort is critical to creating requirements specifications and human inspection can detect many specification errors, effective and widespread development of precise, unambiguous specifications in industry requires powerful, robust tool support. Not only can software tools find specification errors that inspections miss, they can do so much more cheaply. To explore what form tools supporting the formal specification of requirements should take, we have developed a suite of software tools for constructing and analyzing requirements specifications in the SCR tabular notation [8]. The tools include a *specification editor* for creating the specification [9], a *simulator* for validating that the specification satisfies the customer's intent [8], a *dependency graph browser* for understanding the relationship between different parts of the specification [10],

and a *consistency checker* [11] to analyze the specification for properties such as syntax and type correctness, determinism, case coverage, and lack of circularity. The toolset also contains the *model checker* Spin [16], a *verifier* TAME [1], a *property checker* based on decision procedures called Salsa [2], and an *invariant generator* [17], all of which may be useful in analyzing specifications for critical application properties, such as safety and security properties.

The utility of the SCR tools has also been demonstrated in several projects involving real-world systems. In one project, NASA researchers used the SCR consistency checker to detect several missing assumptions and instances of ambiguity in the requirements specification of the International Space Station [4]. In a second project, engineers at Rockwell Aviation used the SCR tools to detect 28 errors, many of them serious, in the requirements specification of a flight guidance system [20]. In a third project, our group at NRL used the SCR tools to expose several errors, including a safety violation, in a moderately large contractor-produced specification of a U.S. weapons system [12]. Recently, we have begun using the SCR method and tools to analyze specifications for security properties. This paper briefly describes our experiences in applying the SCR tools to two secure systems: a communications device called CD and a biometrics standard.

2 Applying the SCR Tools to Secure Systems

2.1 Applying SCR to a Communications Device

COMSEC (Communications Security) devices, devices which manage encrypted communications, are vital to the correct operation of U.S. military systems. CD is a COMSEC device that is designed to provide cryptographic processing for a U.S. Navy radio receiver. In addition to generating keystreams compatible with another cryptographic device and supporting multiple channels and multiple cryptographic algorithms, CD downloads associated algorithms and keys into working storage, assigns them to designated communication channels, maintains the association between an algorithm and its keys, and clears algorithms and keys from memory. CD, based on a technology for implementing COMSEC devices in software as well as hardware, presents a new challenge in the development of COMSEC devices. While a solid base of experience exists for implementing trustworthy COMSEC devices in hardware, implementing COMSEC devices in software is rare.

To provide a high degree of assurance in the correctness of CD's specification, we applied the SCR tools [18]. Our results suggest that applying SCR in the development of COMSEC devices of moderate size and complexity is practical, effective, and low-cost. In approximately one person-month, we were able to represent a significant subset of a prose requirements document for CD in the SCR notation and to establish that the SCR specification satisfies a set of security properties. The SCR specification of CD is moderately complex, consisting of 39 variables (17 input variables, three auxiliary variables, and 19 output variables). Figure 1 provides a natural language formulation and a formal representation of each of the seven security properties that we verified with the SCR tools. Because

the SCR requirements specification of CD has been validated using simulation and verified to satisfy seven critical security properties, the SCR requirements specification of CD can help guide both the development of the source code for CD and the development of test cases for evaluating the conformance of the source code with CD's requirements.

No.	Description	Property
1	If CD is tampered with, then key 1 in keybank 1 is zeroized	@T(mTamper) ⇒ cKeyBank1Key1' = 0
2	When the zeroize switch is activated, key 1 in keybank 1 is zeroized	@T(mZeroizeSwitch = on) ⇒ cKeyBank1Key1' = 0
3	No key can be stored in location 1 of keybank 1 before an algorithm has been loaded into the first location of algorithm storage segment 1	cKeyBank1Key1 ≠ 0 ⇒ cAlgStoreSegment1 ≠ 0
4	If backup power has an undervoltage when primary power is unavailable, the CD enters either Alarm mode or Off mode	@T(mBackupPower = undervoltage) WHEN mPrimaryPower = unavailable ⇒ smOperation' = sAlarm OR smOperation' = sOff
5	If backup power is overvoltage then the CD is in Initialization, Standby, Alarm, or Off mode	mBackupPower = overvoltage ⇒ smOperation = sInitialization OR smOperation = sStandby OR smOperation = sAlarm OR smOperation = sOff
6	If primary power has an overvoltage then either the CD is in Initialization, Standby, Alarm, or Off mode, or the CD enters Initialization mode	@T(mPrimaryPower) = overvoltage ⇒ smOperation = sStandby OR smOperation = sAlarm OR smOperation = sOff OR smOperation' = sInitialization
7	If primary power has an undervoltage then either the CD is in Initialization, Standby, Alarm, or Off mode, or the CD enters Initialization mode	@T(mPrimaryPower) = undervoltage ⇒ smOperation = sStandby OR smOperation = sAlarm OR smOperation = sOff OR smOperation' = sInitialization

Fig. 1. Sample properties of CD.

2.2 Applying SCR to a Biometrics Standard

Positive identification and authentication of personnel is a critical issue for many systems. For example, U.S. government personnel must often interact with the commercial sector in situations where reliable personnel identification is critical for limiting access to sensitive information systems. While biometrics technology addresses the critical issue of personnel identification and authentication, prior to deployment, a biometrics product must be certified to satisfy published assurance standards. However, the labor-intensive process of evaluating and validating a biometrics product is very expensive and time consuming. One way to reduce these problems is to use automated methods to support product evaluations. Applying such methods should not only lead to a less costly and shorter process for evaluating biometrics and other security products but should also produce a more effective process.

To assess the utility of the SCR method and tools for evaluating a biometrics product for correctness, we applied SCR to the BioAPI specification [3], a standard which defines the interface between an authentication device that uses biometrics data and an application program. The goal of the biometrics API (application program interface) is to enable rapid development of biometrics applications, the flexible deployment of many biometrics devices across platforms and operating systems, and an improved ability to exploit price and performance advances in biometrics. From the BioAPI standard, we produced an SCR tabular specification, which captures the behavior of six major operations in the standard. The SCR specification consists of 20 variables: 10 input variables, one mode class variable, and nine output variables. In about two weeks, we were able to create the specification, to demonstrate with the consistency checker that the specification contained no missing cases and no ambiguity, and to verify a critical security property. The goal of this property is to demonstrate that “the system shall successfully authenticate a user before mediating actions initiated by that user.”

3 Observations

The SCR method and tools contributed to the specification and analysis of these two systems in a number of ways. We describe these ways below:

- **Requirements Capture.** Developing a formal requirements specification from the prose requirements document for CD was difficult, largely because the prose document was organized very differently than an SCR specification. Moreover, even though the prose document was high quality, a number of questions about the required behavior of CD arose. Two SCR tools were useful in correcting and extending our initial SCR specification of CD’s required behavior. First, we used an automatic invariant generator to construct state invariants from the draft specification. Analyzing these invariants identified a number of missed requirements and some incorrectly captured requirements. After correcting these problems, we used our simulator and a GUI builder to construct a simulation of CD. Because the CD program manager was very busy, he did not have the time to review our specification. Instead, we showed him several scenarios using our CD simulator. By viewing the simulation, he was able to quickly identify a number of errors in our CD specification which we subsequently corrected.
- **Formal Verification.** To verify the seven security properties listed in Figure 1, we ran TAME, a user-friendly interface to the theorem prover PVS. TAME was able to prove four of the seven properties directly. To prove the remaining properties, TAME needed several supporting invariant lemmas. Fortunately, each of the required lemmas belonged to the set of state invariants that we were able to construct with our invariant generation algorithm [17].
- **Detecting Incorrect Properties.** We were unable to prove that the CD specification satisfies an eighth security property. Although we tried apply-

ing the model checker Spin to the CD specification, Spin repeatedly ran out of memory due to the large state space of the CD specification and thus was unable to verify *or* refute any of the eight security properties. The false property was detected by running TAME and studying the problem transitions returned by TAME. By experimenting with the CD simulator, we were able to construct a counterexample that ended in one of the problem transitions and hence demonstrated that the eighth property was false.

- **Correct Formulations of Security.** Formulating a correct formal statement of a given security property can be difficult. In our work on the biometrics standard, the correct formulation of the security property (see above) required more time than verifying the property.
- **Code Validation.** The most important open problem is how to validate the source code that implements a secure system. While specifying the required behavior of a secure system and formally proving that the specification satisfies critical security properties can often be accomplished in a reasonable time, one still needs to demonstrate that the source code operates securely. One approach to code validation is specification-based testing. That is, one can derive a set of test cases from the specification and automatically use these test cases to determine whether the source code satisfies the specification. Some initial progress in developing an automatic test case generator from a requirements specification is reported in [6].

Acknowledgments

Jim Kirby developed both the CD specification and the specification of the BioAPI standard. Moreover, Jim, Myla Archer, and Ralph Jeffords used TAME and the invariant generator to verify that the CD specification and the BioAPI specification satisfy selected security properties. Ramesh Bharadwaj also verified the properties in Figure 1 using Salsa and constructed a counterexample for the eighth property of CD using the SCR simulator. I am grateful to Myla Archer and Jim Kirby for their comments on an earlier draft of this paper.

References

1. M. Archer, C. Heitmeyer, and E. Riccobene. Using TAME to prove invariants of automata models: Case studies. In *Proc. 2000 ACM SIGSOFT Workshop on Formal Methods in Software Practice (FMSP'00)*, August 2000.
2. R. Bharadwaj and S. Sims. Salsa: Combining constraint solvers with BDDs for automatic invariant checking. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '2000)*, Berlin, March 2000.
3. BioAPI Consortium. The BioAPI Specification. Version 1.00, March 30, 2000.
4. Steve Easterbrook and John Callahan. Formal methods for verification and validation of partial specifications: A case study. *Journal of Systems and Software*, 1997.
5. S. R. Faulk, L. Finneran, J. Kirby, Jr., S. Shah, and J. Sutton. Experience applying the CoRE method to the Lockheed C-130J. In *Proc. 9th Annual Conf. on Computer Assurance (COMPASS '94)*, Gaithersburg, MD, June 1994.

6. A. Gargantini and C. Heitmeyer. Automatic generation of tests from requirements specifications. In *Proc. ACM 7th Eur. Software Eng. Conf. and 7th ACM SIGSOFT Symp. on the Foundations of Software Eng. (ESEC/FSE99)*, Toulouse, FR, September 1999.
7. Constance L. Heitmeyer and John McLean. Abstract requirements specifications: A new approach and its application. *IEEE Trans. Softw. Eng.*, SE-9(5):580–589, September 1983.
8. Constance Heitmeyer, James Kirby, Jr., Bruce Labaw, and Ramesh Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *Proc. Computer-Aided Verification, 10th Annual Conf. (CAV'98)*, Vancouver, Canada, 1998.
9. C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Proc. 10th Annual Conf. on Computer Assurance (COMPASS '95)*, pages 109–122, Gaithersburg, MD, June 1995.
10. Constance Heitmeyer, James Kirby, Jr., and Bruce Labaw. Tools for formal specification, verification, and validation of requirements. In *Proc. 12th Annual Conf. on Computer Assurance (COMPASS '97)*, Gaithersburg, MD, June 1997.
11. C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, April–June 1996.
12. C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Trans. on Softw. Eng.*, 24(11), November 1998.
13. Kathryn Heninger, David L. Parnas, John E. Shore, and John W. Kallander. Software requirements for the A-7E aircraft. Technical Report 3876, Naval Research Lab., Wash., DC, 1978.
14. Kathryn L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Softw. Eng.*, SE-6(1):2–13, January 1980.
15. S. D. Hester, D. L. Parnas, and D. F. Utter. Using documentation as a software design medium. *Bell System Tech. J.*, 60(8):1941–1977, October 1981.
16. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
17. Ralph Jeffords and Constance Heitmeyer. Automatic generation of state invariants from requirements specifications. In *Proc. Sixth ACM SIGSOFT Symp. on Foundations of Software Engineering*, November 1998.
18. J. Kirby, Jr., M. Archer, and C. Heitmeyer. SCR: A practical approach to building a high assurance COMSEC system. In *Proceedings of the 15th Annual Computer Security Applications Conference (ACSAC '99)*. IEEE Computer Society Press, December 1999.
19. S. Meyers and S. White. Software requirements methodology and tool study for A6-E technology transfer. Technical report, Grumman Aerospace Corp., Bethpage, NY, July 1983.
20. Steve Miller. Specifying the mode logic of a flight guidance system in CoRE and SCR. In *Proc. 2nd ACM Workshop on Formal Methods in Software Practice (FMSP'98)*, 1998.
21. D. L. Parnas, G.J.K. Asmis, and Jan Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32(2), April–June 1991.