

Application of Lightweight Formal Methods to Software Security

David P. Gilliam and John D. Powell
Jet Propulsion Laboratory, California Institute of Technology
David.Gilliam@jpl.nasa.gov, John.Powell@jpl.nasa.gov

Matt Bishop
University of California at Davis
bishop@cs.ucdavis.edu

Abstract

Formal specification and verification of security has proven a challenging task. There is no single method that has proven feasible. Instead, an integrated approach which combines several formal techniques can increase the confidence in the verification of software security properties. Such an approach which specifies security properties in a library that can be re-used by 2 instruments and their methodologies developed for the National Aeronautics and Space Administration (NASA) at the Jet Propulsion Laboratory (JPL) are described herein. The Flexible Modeling Framework (FMF) is a model based verification instrument that uses Promela and the SPIN model checker. The Property Based Tester (PBT) uses TASPEC and a Text Execution Monitor (TEM). They are used to reduce vulnerabilities and unwanted exposures in software during the development and maintenance life cycles.

1. Introduction

Specifying software properties is a challenging task. Even more challenging is specifying informal specifications formally. [van Lamsweerde, 2000, p.148] This difficulty is due to the imprecision of natural language and the difficulty in ensuring that the specifications are correct. [Hussmann, 1977, p. 9; Schach, 2005, p.151] Applied to security, formal specification is particularly complex as security requirements mostly state what must not happen. [Rushby, 2001, p. 3] The problem of trying to specify security properties formally was made apparent during the 1970's when the United States government commissioned development of a provably secure multics system using mathematical modeling. [Bell-LaPadula, 1976] Their approach addressed only confidentiality, and then only partially. The number of follow-on discussions on security property specifications is witness

to this problem. [Biba, 1977; McLean, 1990; Payne, 1995; Rushby, 2001; Deng, 2003; Nicol, 2004] The need to formally specify and verify security properties is easily seen by the growing list of software vulnerabilities. [Mitre, 2004] It is apparent that better specification and verification of security properties will lead to more secure software. Formal specifications and methods can fill this role and improve the quality of software making it more dependable. [Easterbrook, 1996, p. 3; Nicol, 2004, p. 49]

The following discussion will make a case for the use of integrated lightweight formal methods to verify security properties in software. A definition of formal specifications and methods for software verification will be presented. Next, we will focus on the value of lightweight formal methods that can be used more readily for verification activities. Finally, the discussion will focus on 2 formal, integrated techniques, model checking and property-based testing, that are being used at the Jet Propulsion Laboratory (JPL) for verification of security properties. Applying these lightweight formal techniques is "a cost-effective means for improving the overall quality of the software." [Easterbrook, 1996, p. 6]

1.1 Formal Specifications

The term 'formal specification' is overloaded and often used quite loosely. It is defined here as "the expression in some formal language and at some level of abstraction, of a collection of properties some system should satisfy." [van Lamsweerde, 2000, p.147] Formal specification applies formal, "precise rules of interpretations that allow many of the problems with natural language to be overcome." [van Lamsweerde, 2000, p.147] Formal mathematical languages and analysis are used in formal specification to obtain precision in expressing specifications and to verify them. The process of formal specification and analysis is often termed 'formal methods,' defined by NASA as, "the use

of techniques from formal logic and discrete mathematics in the specification, design, and construction of computer systems and software.” [NASA-GB-002-95, 1995, p.5] There are 3 parts in formal specification: a) a set of notations, b) set of techniques, and c) set of procedural guidelines. [Husmann, 1977, pp. 13-14]. Formal methods are techniques using precise notations to specify artifacts in syntactical form, while semi-formal or “lightweight” formal methods use both formal and informal techniques. [Husmann, 1977, p. 14]

1.2 Why Use Formal Specifications

Formal specifications define precisely what “security” means in the context of the system being analyzed. Applying formal methods enables the analyst to see precisely how the system satisfies the specifications. For example, formally specifying dependability properties can help assure that these properties are not violated as well as aid in assuring that they do not conflict. [Rushby, 2002, p. 10] The application of formal methods to security was, in fact, a driving force to the development of formal methods, largely funded by government. The “results included the development of formal security models; tools for reasoning about security, and applications of these tools to proving a system secure. Security provided a challenging research application for the formal methods community.” [Wing, 1998, p. 26] Proving a system secure, though, is very challenging.

1.3 When and How to Use Formal Specifications

When and how to use formal methods depends on the artifacts and the type of verification required as well as the expertise that is available. The approach must provide an easily understandable and verifiable framework, and fit within cost and schedule constraints. It should be a “precise yet understandable way of specifying correct behavior, and an exhaustive method of determining that the system model satisfies this specification for all input patterns.” [McMillan, 1992, p. 11] The framework has 3 key elements “1) a mathematical model of the system to be verified, 2) a formal language for framing the correctness problem, and 3) a methodology for proving the statement of correctness.” [McMillan, 1992, p. 12] Formal mathematical languages such as Z, Computational Tree Logic (CTL), and Linear Temporal Logic (LTL), which are based on a typed set theory and the concept of a schema, [Spivey, 1988; Gerth, 1995] provide a framework and the capability to formally verify security

specifications through formal, provable logic. Appendix A provides an example.

1.4 The Case for Lightweight Formal Methods Techniques

Formal specification is a complex task. For time and resource constrained projects, use of formal approaches can delay the task. Further, formal approaches can be problematic when used with large and complex systems. [George, 2003, p. 3] They require both experts and time. However this task is made easier by mechanizing formal methods techniques. [Rushby, 1999]

1.5 Formal Methods

Formal specification languages for model checkers support specification of systems through development of models. Several model checkers have been developed to mechanize the application of finite state verification, such as the SPIN model checker which uses the language Promela [Holzmann, 1997], SAL (Symbolic Analysis Laboratory), a model checker from SRI [Rushby, 2002], Murphi [Winters, 2000], and SMV (Symbolic Model Verifier), a model checker that uses CTL. [McMillan, 1993] Both Murphi and SMV were developed for hardware validation, while SPIN was developed for communication protocols and software structures. [Schneider, 1998, p.3]

1.6 Lightweight Formal Methods

These mechanized tools have been termed by some as “lightweight techniques”, such as state and transition based models, Petri nets, and property-based testing. [George, 2003, p. 4] “‘Lightweight’ methods may involve both greater automation of formal analysis, and more focused application of formal techniques. As a result they can be more cost effective for a broad range of problems” than more formal techniques. [Kuhn, 2003, p. 1] Lightweight techniques can provide a high degree of assurance while maintaining some fidelity to the actual artifact. [van Lamsweerde, 2000, p. 155] Additionally, the potential for re-use of these instruments when there are late design changes, and during the maintenance phase, have the potential to reduce overall costs for software projects. [Gilliam, 2002, p. 158] The techniques to be applied need to fit within resource constraints and the types of artifacts that require verification. [Easterbrook, 1988, p. 10] A framework that applies integrated lightweight methods provides even higher confidence in the verification of the artifacts. Such an integrated approach has been the

focus of a software security research effort at JPL. [Gilliam & Powell, 2002, Gilliam, Powell, Haugh, and Bishop, 2003, Powell, 2002]

Previous WETICE papers have discussed the development of a software security assessment instrument (SSAI) at JPL and the various components and usage. These components include a Vulnerability Matrix comprised of vulnerabilities and their properties, a collection of security assessment tools that are available in the public domain and the pros and cons of each of the identified tools, a Flexible Modeling Framework (FMF) which uses model checking techniques, a Property-Based Testing (PBT) instrument, and a software security property checklist for use in the development and maintenance life cycles. [Gilliam, Wolfe, and Sherif, 2003; Gilliam & Powell, 2002; Gilliam, Kelly, Powell, and Bishop, 2001; Gilliam, Kelly, and Bishop, 2000] The research identified in this paper presents the need for the application of lightweight formal techniques to software security, and the prototyping process for the SSAI previously reported to the WETICE Enterprise Security (ES) Workshop.

2. Model Checking and Property-Based Testing

The purpose and use of tools like model checkers and testers is to allow for mechanization of formal specifications to reduce cost and schedule while increasing efficiency for formal verification activities and to assure that the software artifacts are free from potential conflicts and violations in the specifications. [Holzmann, 2002, p. 369; Hamon, 2004, p. 5] Model checking involves:

- Building a state-based model of the system
- Identifying properties to be verified
- Checking the model for violations of the specified properties.

Model checkers such as SPIN [Holzmann, 2004] automate the process of verifying a property over its corresponding model.

Model checkers perform an exhaustive search of a state space generated by a model. State space is the set of total reachable system states represented in the model. A given state consists of all variables in the model and their associated values at a given point in time. Software model checkers automatically explore all paths from a start state by examining transitions in the state space to determine the reachability of a state that violates the property. [Nicol, 2004, p. 52] When properties are violated, the checker gives the counterexample and stops. The properties are verified as holding or not holding for each transition. This automation provides

high value for large-scale, complex systems where specifications become more complex. [Rushby, 2002, p.175] However it does not obviate the need for experts since the development of the verification model is non-trivial, even for experts. [Holzmann, 2001, p. 3]

2.1. State Space Problem

The modeling objective is to verify properties with respect to the model over as many scenarios as feasible. However, as the size and complexity of the model increases, the state space to be checked grows at an exponential rate. "This exponential growth in the state space known as the state explosion problem is the limiting factor in applying automatic verification methodologies to large systems." [McMillan, 1992, p. 14] Modeling of concurrent software systems quickly becomes impracticable when: "a) the possible number of concurrent processes increases, b) the functionality in 1 or more processes grows, and c) the interactivity between or complexity of 1 or more processes increases." [Gilliam, 2003, p. 202] Security properties exacerbate the problem due to their nature. They express what must not happen, [Rushby, 2001, p. 1] i.e., 'no access shall be granted until authentication is verified'.

2.2. Flexible Modeling Framework

To address the state explosion problem, JPL has developed a Flexible Modeling Framework (FMF) that uses a "divide and conquer approach" while seeking to maintain fidelity to the software artifact. [Powell, 2002, p. 3] The FMF uses compositional verification to analyze models and verify the results for models that represent the system (S). The basis of the compositional approach is the verification of a system (S) with regard to a subset of its environment (e) in a manner that allows those results to be extrapolated to the environment at large (E). The FMF approach narrows the focus to those components for which security properties have been identified and which can be modeled.

Use of this combinatorial approach allows interactions between components to be examined bringing to light potential questions about their relationships. These questions enable decisions to be made early in the life cycle. Further, efficient, localized updates of the system model can more easily be generated. Issues affected by subsequent changes can then be revisited though required re-verification of affected combinations. This last feature is a failsafe and not a substitute for good practices such as

documentation of decisions and emergent requirements. [Gilliam & Powell, 2003, p. 203]

2.3. Property-Based Testing (PBT)

Property-based testing (PBT) is a technique that verifies that specified security properties are not violated in the coding phase of the life cycle. Properties are invariants that are to hold during program execution. Implementation difficulties and environmental considerations may affect conformance to properties (and hence the security of execution) and thus the properties may not always hold. PBT provides additional assurance that the software is correct and satisfies the specified properties when execution follows the tested control and data flow paths. [Gilliam, 2003, p. 203]

A PBT instrument developed by UC Davis in cooperation with JPL, mechanizes verification of security properties in code. The PBT expresses properties in a low-level test language called TASPEC. Like the FMF, the PBT focuses the testing on the security properties of interest. Intuitively, the PBT instrument looks at the execution of the program sequences as a series of state transitions. If any state transition causes a violation of a property, an error message is generated. The PBT examines data from program executions to determine this.

The goal of the PBT is to test as many paths of control as possible. First, a program called the instrumenter analyzes the security properties and the program, and inserts code to emit messages indicating changes of state relevant to the security properties. The program is then 'sliced', creating a second program that satisfies the properties if, and only if, the original program satisfies those properties. The second program contains only those paths of control and data flow that affect the properties. This focuses the testing on paths of execution relevant to the security properties rather than on all possible paths of execution. The instrumented, sliced program is then compiled and executed. During execution, the messages indicating changes of state are saved to a file.

Second, a test execution monitor (TEM) program is given the properties in TASPEC and the messages indicating changes of state from the instrumented program's run. The TEM checks each state transition and verifies that the properties held during execution. If the properties did hold, then they held throughout the execution. If not, the TEM can determine where in the program the failure occurred (see Figure 5). [Gilliam, 2003, p. 204] The testing either validates the properties or shows they do not hold.

3. Prototyping the FMF and PBT

While we believe that these semi-formal verification techniques aid in ensuring that specified security properties in software are not violated, the instruments themselves must be prototyped to show that they do perform as intended and do so in a cost-effective way. Both their value and their relative cost-effectiveness must be verified for these instruments to be useful and of benefit in the development and maintenance life cycles.

Currently, these tools are being piloted with a COTS JAVA-based application. As the application was already developed and available, it provided an avenue to verify whether or not the instruments would provide value for verification of the security of the application being evaluated. By extension, it is hoped that the value and cost-effectiveness of the current activity can be determined.

The verification process required working with the developers of the application to obtain from them their software and architecture artifacts to extrapolate properties for the FMF and PBT. Additional information and explanation of some of the properties in the architectural artifacts was obtained from the developers. From these artifacts and in working with the developers, a model of the software was developed. Further, in view of the purpose of the application, security properties were specified independently.

The current model is still being completed. To expedite the process and to verify that the specified properties are not violated in the software code itself, the specified properties of the software, the model, and the identified security properties are being provided for use with the FMF and PBT instruments.

The goal is to combine the use of the two instruments to complement the verification efforts in order to provide a higher level of assurance of the security of the software. We expect this approach will improve the overall security of software. The results of the current investigation will be provided to the National Aeronautics and Space Administration's (NASA) Independent Verification and Validation (IV&V) Center through whom this research has been funded.

5. Conclusion

Through use of formal techniques in application to security, security assessment instruments and tools in the software development and maintenance life cycles can improve the security of software if used correctly. The SSAI being developed at JPL and UC Davis is hoped to be a step forward in this direction. Through the

application of these instruments in a coordinated effort, a higher level of assurance for security can be achieved, which is the ultimate goal of this research effort.

Tools and instruments that can be used during both the development and maintenance life cycle beginning with a security checklist in the inception and requirements phases through retirement will create an environment of stronger security.

7. Acknowledgements

The research described in this paper is being carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

7. References

- Bell, D. E. and LaPadula, L. J. (March, 1976). Secure computer systems: Unified exposition and multics interpretation. Technical Report Mitre TR-2997, Mitre Corporation, Bedford, MA.
- Biba, K. J. (April, 1977). Integrity considerations for secure computer systems. ESD-TR-76-372, ESD/AFSC. Hanscom AFB, Bedford, MA (available as MITRE MTR-3153, NTIS AD A039324).
- Botting, J. R. (2004). Modal logic. Retrieved November 22, 2004, from http://www.csci.csusb.edu/dick/math/logic_9_Modalities.html.
- Deng, Y., Wang, J., Tsai, J. P., Beznosov, K.(2003). An approach for modeling and analysis of security system architectures. IEEE Transactions on Knowledge and Data Engineering. Vol. 15, No. 5, 1099 – 1119.
- Easterbrook S. M., and Callahan, J. R. (1996). Formal methods for verification and validation of partial specifications: A case study. Report to NASA Independent Verification and Validation Facility, Fairmont, WV. Retrieved November 14, 2004, from www.cs.toronto.edu/~sme/papers/1998/NASA-IVV-97-010.pdf.
- George, V., and Vaughn, R. (2003). Application of lightweight formal methods in retirement engineering. Crosstalk: The journal of defense software engineering, January 2003. Retrieved November 23, 2004 from <http://www.stsc.hill.af.mil/crosstalk/2003/01/George.html>.
- Gerth, R., Peled, D., Vardi, M.Y., and Wolper, P. (1995). Simple on-the-fly automatic verification of linear temporal logic. Proceedings of the 15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification 15 (pp. 3-18). London: Chapman and Hall, Ltd.
- Gilliam, D. P., and Powell, J. D. (2002). Integrating a flexible modeling framework (FMF) with the network security assessment instrument to reduce software security risk. Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. WETICE 2002, 153 – 158.
- Gilliam, D. P., Powell, J. D., Haugh, E., and Bishop M. (2003). Addressing software security and mitigations in the life cycle. Proceedings of the 28th Annual NASA Goddard IEEE Software Engineering Workshop (SEW), 201 – 206.
- Hamon, G., de Moura, L., and Rushby, J. (May, 2004). Generating Efficient Test Sets with a model checker. Computer Science laboratory (CSL) Technical Note. SRI International. Retrieved November 14, 2004, from <http://www.csl.sri.com/users/rushby/biblio.html>.
- Holzmann, G. J. (1997). The model checker spin. IEEE Transactions on Software Engineering, Volume: 23, Issue: 5, 279 – 295.
- Holzmann, G. J. (2001). From code to models. IEEE Proceedings of the Second International Conference on Application of Concurrency to System Design, 3 – 10.
- Holzmann, G. J. (2004). The SPIN model checker: Primer and reference manual. Boston, MA: Addison-Wesley.
- Holzmann, G. J., & Smith, M. H. (2002). An automated verification method for distributed systems software based on model extraction. IEEE Transactions on Software Engineering, Volume: 28, No. 4, April 2002, 279 – 295.
- Hussmann, H. (1997). Formal foundations for software engineering methods. Goos, G., Hartmanis, J., and van Leeuwen, J. (eds.), Lecture Notes in Computer Science, 1322. Berlin: Springer.
- Kuhn, D. R., Craigen, D., & Saaltink, M. (2003). Practical application of formal methods in modeling and simulation. National Institute of Standards and Technology Paper. Retrieved November 23, 2004, from <http://csrc.nist.gov/staff/kuhn/kuhn-craigen-saaltink-03.pdf>
- McLean, J. (January, 1990). The specification and modeling of computer security. IEEE Computer, Vol. 23, Issue 1, 9-16.
- McLean, J. (1990). Security models and information flow. Research in Security and Privacy. Proceedings of the IEEE Computer Society Symposium, 180-187.
- McLean, J. (January, 1999). Twenty years of formal methods. Proceedings of the IEEE Symposium on Security and Privacy, 115 – 116.

McMillan, K. L. (1992). Symbolic model checking: An approach to the state explosion problem. CMU-CS-92-131, Submitted to Carnegie Mellon University in partial fulfillment of the degree of the requirements for the degree of Doctor of Philosophy in Computer Science. Carnegie Mellon University. Later published (1992) as *Sympolic model checking*. Kluwer Academic Publishers: Norwell, Mass. Retrieved Nov. 22, 2004, from www-cad.eecs.berkeley.edu/~kenmcmil/thesis.ps.

Mitre Corporation (2004). Common Vulnerabilities and Exposures (CVE) List. Retrieved November 22, 2004, from <http://www.cve.mitre.org/cve/downloads/>.

NASA-GB-002-95 (1995). Formal methods specification and verification guidebook for software and computer systems. Volume 1: Planning and technology insertion. Release 1.0. Washington, D.C.: National Aeronautics and Space Administration.

Nicol, D. M., Sandera, W. H., and Kishor, S.T. (2004). Model-Based evaluation: From dependability to security. *IEEE Transactions on Dependable and Secure Computing*. Vol. 1, No. 1, 48 – 65.

Payne, C. N., Jr., Moore, A. P., and Mihelcic, D. M. (1995). An experience modeling critical requirements. *Proceedings of the Ninth Annual Conference on Computer Assurance, 1994. COMPASS '94 'Safety, Reliability, Fault Tolerance, Concurrency and Real Time, Security'*, 245-255.

Powell, J. D. (2003). Reducing software security risk through an integrated approach research initiative: Model based verification of the secure socket layer (SSL) protocol. Research deliverable to the NASA IV&V Facility. Fairmont, WVA.

Powell, J. D. (2004). Integrated approach to reducing software security risk. Research presentation at the Jet Propulsion Laboratory (JPL). February, 2004.

Powell, J. D., and Gilliam, D. P., (2002) Component based approach to modeling for model checking. *The Sixth Biennial World Conference on Integrated Design & Process Technology*. Pasadena, California, 2002.

Rushby, J. (September, 1999). Mechanized formal methods: Where next?. *The World Congress on Formal Methods*. Toulouse, France. Retrieved November 14, 2004, <http://www.csl.sri.com/users/rushby/biblio.html>.

Rushby, J. (March, 2001). Security requirements specifications: How and what? Invited Paper from *Symposium on Requirements Engineering for Information Security (SREIS)*, Indianapolis, IN.

Rushby, J. (February, 2002). Using model checking to help discover mode confusions and other automation surprises. *Reliability and System Safety*. Vol. 75, No. 2, 167-177.

Schach, S.J. (2005). *Object-oriented and classical software engineering*. 6th ed. New York: McGraw-Hill.

Schneider, F., Easterbrook S. M., Callahan, J. R., and Holzmann, G. J. (1998). Validating requirements for fault tolerant systems using model checking. *IEEE Third International Conference on Requirements Engineering*, 4 – 13.

Spivey, J. M. (1988) *Understanding Z: A specification language and its formal semantics*. Cambridge Tracts in Theoretical Computer Science 3. Cambridge: Cambridge University Press

Van Lamsweerde, A., (2000). Formal Specification: A roadmap. In Finkelstein, A. (ed.) *Proceedings of the Conference on the Future of Software Engineering* (pp. 147 – 159). ACM Press.

Wing, J. M. (1998). A symbiotic relationship between formal methods and security. *Proceedings on Computer Security, Dependability and Assurance: From Needs to Solutions*, 7-9 July 1998, 11-13 Nov. 1998, 26 - 38.

Winters, D. D., and Hu, A. J., (2000). Source-Level Transformations for Improved Formal Verification. *IEEE International Conference on Computer Design*, September 2000.