

## Addressing Software Security Risk and Mitigations in the Life Cycle

David Gilliam & John Powell  
California Institute of Technology, Jet Propulsion Laboratory  
[david.p.gilliam@jpl.nasa.gov](mailto:david.p.gilliam@jpl.nasa.gov), [john.d.powell@jpl.nasa.gov](mailto:john.d.powell@jpl.nasa.gov)

Eric Haugh & Matt Bishop  
University of California at Davis  
[haugh@cs.ucdavis.edu](mailto:haugh@cs.ucdavis.edu), [bishop@cs.ucdavis.edu](mailto:bishop@cs.ucdavis.edu)

Traditionally, security is viewed as an organizational and Information Technology (IT) systems function. It is viewed as firewalls, intrusion detection systems (IDS), system security settings and patches to the operating system (OS) and applications running on it. However, until recently, little thought has been given to the importance of security in the software and system life cycle, especially as a formal life cycle approach. Formal approaches to security in the software life cycle are only now beginning to emerge. Yet, to date, an end-to-end life cycle security risk mitigation instrument is not known to be available commercially.[1] The NASA Office of Safety and Mission Assurance (OSMA) has funded the Jet Propulsion Laboratory (JPL) with a Center Initiative, "Reducing Software Security Risk through an Integrated Approach" (RSSR), to address this need. The Initiative is a formal approach to addressing software security in the life cycle through the instantiation of a Software Security Assessment Instrument (SSAI) for the development and maintenance life cycles.

This SSAI, to date, has six elements: 1) a Vulnerability Matrix (VMatrix), 2) a Model-Based Verification (MBV) instrument with a Flexible Modeling Framework (FMF) that uses Model Checking and the SPIN model checker to check for properties in the requirements specifications that lead to vulnerabilities or unwanted exposures; 3) a Property-Based Tester (PBT) for JAVA and C Code for verification of security properties to verify the code that violations of these properties have not been re-introduced into it; 4) a Software Security Checklist (SSC) having two phases: Phase 1 addresses the development and maintenance life cycles, and Phase 2 addresses the external release of software; 5) a list of Security Assessment Tools (SAT's); and 6) Formal training for software and system engineers on software security and the use of these and other tools and instruments in the life cycle.

### ***Vulnerability Matrix***

The VMatrix is a vulnerability database whose purpose is to provide information about various vulnerabilities including exploits used to gain access to systems, how to protect against the exploits and the Common Vulnerabilities and Exposures (CVE) listing. The information is being transferred to the UC Davis Database of Vulnerabilities, Exploits, and Signatures (DOVES) where it will be maintained and updated as new exploits are discovered. This information is used to extract properties and requirements that express potential network vulnerabilities. These properties can then be utilized by the PBT tool and the FMF.

### ***Security Assessment***

The SATs are a collection and an assessment of publicly available software security code checking tools available on the Internet that can be used to test for potential weaknesses of software code. Included is a description of each tool and its intended application. The SAT's are updated as additional tools become available.

### ***Model Checking (MC) and the Flexible Modeling Framework (FMF)***

Software model checkers automatically explore all paths from a start state in a computational tree that is specified in an MC model. The computational tree may contain repeated copies of sub-trees. State of the art Model Checkers such as SPIN exploit this characteristic to improve automated verification efficiency. The objective is to verify system properties with respect to models over as many scenarios as feasible. Since the models are a selective representation of functional capabilities under analysis, the number of feasible scenarios is much larger than the set that can be checked during testing. Model Checkers differ from traditional formal techniques by the following characteristics:

- Model checkers are operational as opposed to deductive

- Model checkers provide counter examples when properties are violated (error traces)
- Their goal is oriented toward finding errors as opposed to proving correctness since the model is correct.

An innovative verification approach, which employs MC as its core technology, is offered as a means to bring software security issues under formal control early in the life cycle. [2,3] The FMF seeks to address the problem of formal verification of larger systems by a divide and conquer approach. [4] First, by verifying a property over portions of the system, then incrementally inferring the results over larger subsets of the entire system. As such, the FMF is: 1) a system for building models in a component based manner to cope with system evolution over time and, 2) an approach of compositional verification to delay the effects of state space explosion. This methodology allows property verification results of large and complex models to be examined and extrapolated appropriately. (See Figure 1)

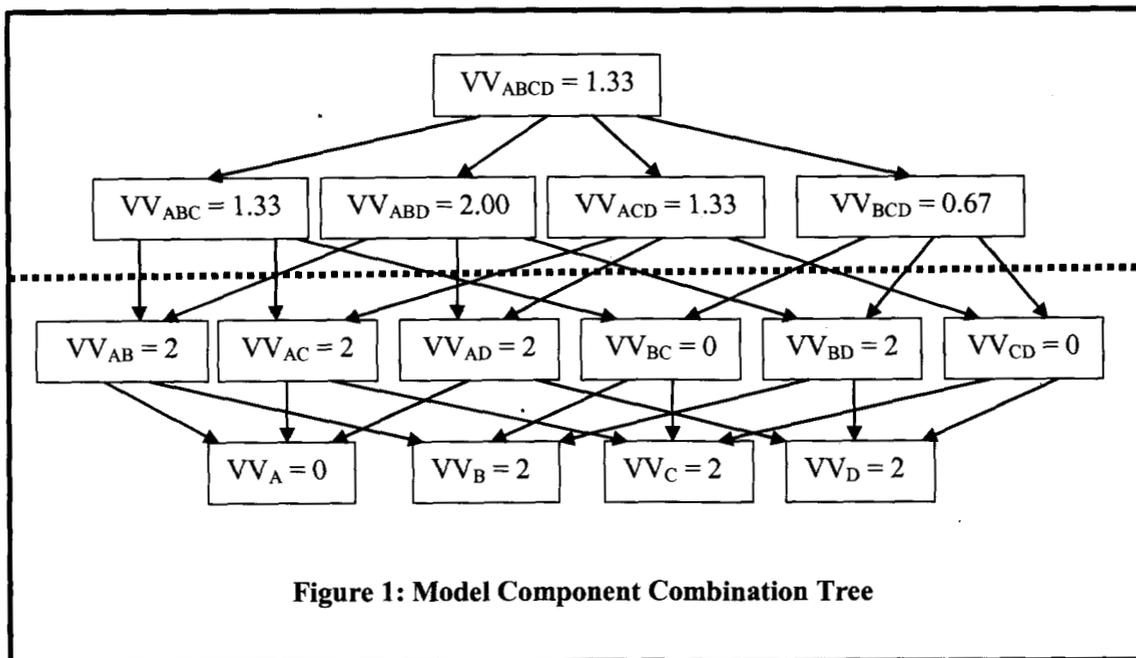


Figure 1: Model Component Combination Tree

Modeling in a component-based manner involves building a series of small models, which later will be strategically combined for system verification purposes. This strategic combination correlates the modeling function with modern software engineering and architecture practices whereby a system is divided into major parts, and subsequently into smaller detailed parts, and then integrated to build up a software system. An initial series of simple components can be built when few operational specifics are known about the system. However, these components can be combined and verified for consistency with properties of interest such as software security properties.

The approach of compositional verification used in the FMF seeks to verify properties over individual model components and then over strategic combinations of them. The goals of this approach are to: 1) infer verification results over systems that are otherwise too large and complex for MC from results of strategic subsets (combinations) while minimizing false reports of defects; 2) retain verification results from individual components and component combinations to increase the efficiency of subsequent verification attempts in light of modifications to a component.

### Property-Based Testing

Property-based testing is a testing technique designed to detect violations of given properties. In this context, the properties desired are obtained from the checklist, or from the properties used in the model checking. The properties are viewed as invariants, that are to hold as the program executes. PBT views the execution of the program as a sequence of state transitions. If any of these transitions cause a violation of the properties, PBT causes an error message to report the failure.

First, the properties are expressed in a low-level testing language called TASPEC. A tool called the instrumenter takes these properties and a program to be tested. The instrumenter then modifies the program so that, when any event occurs that affects whether the state of the program's execution satisfies the properties, a text representation of the change in state is emitted. The instrumented program is then compiled and executed. After the execution is complete, the changes of state have been saved to a file.

The testers then execute a second program called the test execution monitor (TEM). This program is given the properties (in TASPEC) and the changes of state generated from the test run of the program. The TEM then checks each change of state to ensure that, if the properties held when the program began execution, then they held throughout the execution. If not, the TEM can determine where in the program the failure occurred.

PBT is different than formal verification. It recognizes that implementation difficulties, and environment considerations, may affect conformance to the properties (and hence the security of execution). A key observation is that testing does not validate that a program will always meet the properties, unless all possible paths of execution are traversed. But it does provide additional assurance that the implementation is correct, and does satisfy the properties, when execution follows the tested control and data flow paths.

#### ***Software Security Checklist (SSC)***

The SSC has two foci: 1) a checklist for software developers to write secure code for applications (including tools to integrate security into the various stages of the software life cycle); and 2) a checklist to verify that software released by NASA does not allow unauthenticated access into NASA networks, or provide other information about NASA's, processes, systems, networks, or other sensitive data (such as IP Address space, HR data, or processes that can be exploited).

A checklist for the development and maintenance life cycles is a womb to tomb process that begins with a pre-requirements study to be able to elicit the appropriate requirements from the stakeholders and applicable documents, standards, regulations, laws, etc. and specify them; and it ends with decommissioning software and systems and the impact on the computing environment, including re-verification of systems from which critical software has been decommissioned. In between there are a number of critical design and programming issues as well as tracing security requirements, and performing test and verification of them, including the maintenance life cycle phase.

A checklist for the external release of software (i.e., software that is developed for release external to the organizational environment) is specified by NASA in NPG 2210, NASA Procedures and Guidelines: External Release of NASA Software.”[5] However, there is no guidance provided on the contents of the checklist nor a release authority process. The research initiative delivered to NASA a draft of a potential checklist and release authority process to be used for the external release of software. A process for evaluation of code for potential security issues was also provided. The evaluation of the source code included looking for problems that might expose NASA and NASA partners to potential security exposures. Included in the release checklist is a sample set of PERL scripts to aid in looking for potential items in the software that may violate security requirements or present security risks such as embedded Center IP addresses, Human Resource information like phone numbers, use of known vulnerable libraries, and weak subroutines.

#### **Software Security Assessment Instrument (SSAI)**

This collection of tools and utilities, collectively named the Software Security Assessment Instrument (SSAI), can be used individually or in concert to ensure the security of network aware application software and systems as shown in Figure 2 below. Working together, the various tools and utilities provide a distinct advantage whereby each tool's output may be used for input for the other tools. The use of these tools and instruments results in a more comprehensive assessment of the software undergoing analysis.

The SSAI is being piloted by the JPL's Multi-Mission Encryption Communication System (MECS), a JPL project supporting flight and ground communications systems. In addition, the instrument is being evaluated for use in the Deep Space Mission Systems (DSMS). An evaluation of the applications and protocols used to support flight and ground support systems is currently being assessed for further piloting of the SSAI and integrating the technology into this environment.

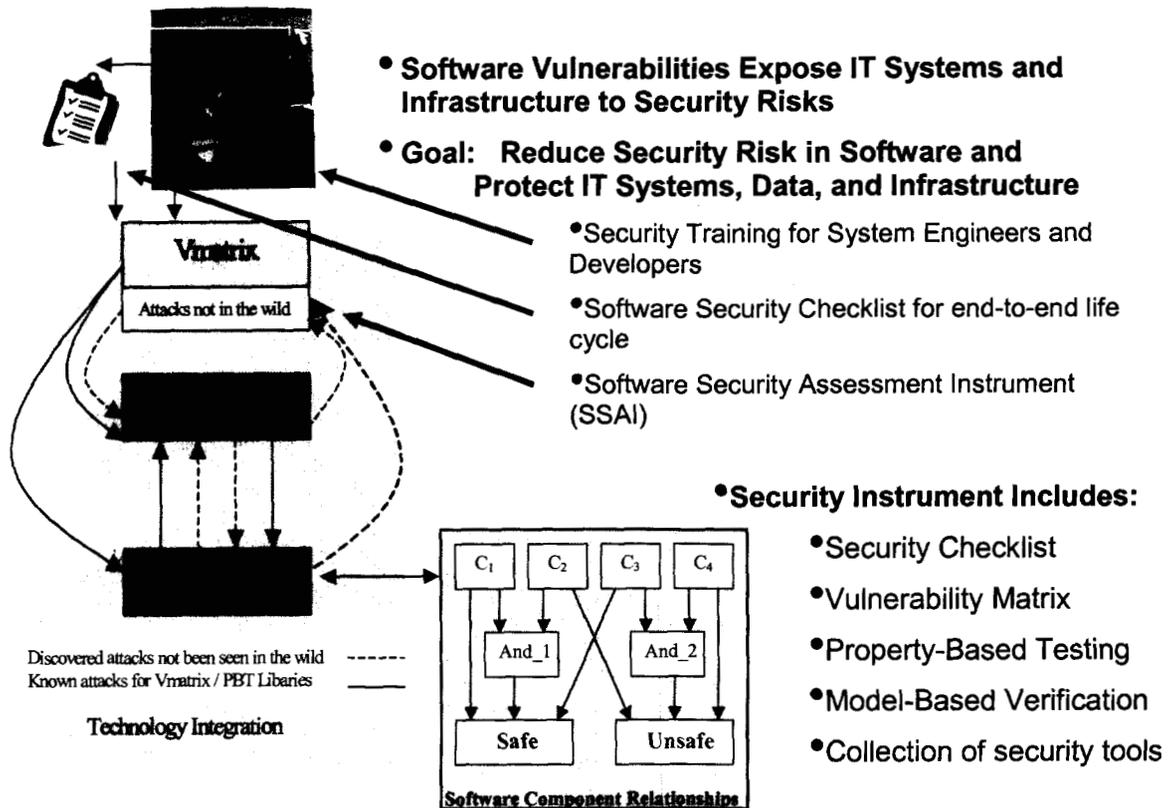


Figure 2: Integrated Use of the Security Assessment Instrument

On another front, the SSAI is being included as an instrument for use in the JPL Project Life Cycle. In response to NPG 7120.5B, "NASA Program and Project Management Processes and Requirements,"[6] and NPG 2810.1, "Security of Information Technology,"[7] an effort is underway to integrate security into the project life cycle. The SSAI along with experts in the use of these instruments and tools will be made available to projects as part of a security risk assessment for the project life cycle.

Training courses for project managers and software developers are currently being developed to integrate security and elements of the SSAI into the project life cycle. In addition, an approach is being pursued to have available at the various NASA Centers domain experts to assist projects with software and system security and methods to identify and mitigate risk in the project life cycle. It is hoped through these efforts that NASA will be able to produce software that has a higher level of assurance that security defects and unwanted exposures are not present in the final product.

#### REFERENCES

- [1] C. Mann, "Why Software Is so Bad," Technology Review (July/August 2002).
- [2] D. Gilliam, J. Kelly, J. Powell, M. Bishop, "Development of a Software Security Assessment Instrument to Reduce Software Security Risk" Proc. of the Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Boston, MA, pp 144-149.
- [3] D. Gilliam, J. Powell, J. Kelly, M. Bishop, "Reducing Software Security Risk Through an Integrated Approach", IEEE Goddard 26th Annual Software Engineering Workshop.
- [4] Component Based Model Checking, J. Powell, D. Gilliam, Proceedings of the 6th World Conference on Integrated Design and Process Technology, June 23-28, Pasadena CA, p66 & CD
- [5] NPG 2210, "NASA Procedures and Guidelines: External Release of NASA Software," January, 2002, August 26, 1999, [http://nodis3.gsfc.nasa.gov/library/lib\\_docs.cfm?range=2](http://nodis3.gsfc.nasa.gov/library/lib_docs.cfm?range=2)
- [6] NPG 7120.5B, "NASA Procedures and Guidelines: NASA Program and Project Management Processes and Requirements," November, 2002, [http://nodis3.gsfc.nasa.gov/npg\\_img/N\\_PG\\_7120\\_005B/N\\_PG\\_7120\\_005B.pdf](http://nodis3.gsfc.nasa.gov/npg_img/N_PG_7120_005B/N_PG_7120_005B.pdf)
- [7] NPG 2810.1, "NASA Procedures and Guidelines: Security of Information Technology," August, 1999, [http://nodis3.gsfc.nasa.gov/library/displayDir.cfm?Internal\\_ID=N\\_PG\\_2810\\_0001&page\\_name=main](http://nodis3.gsfc.nasa.gov/library/displayDir.cfm?Internal_ID=N_PG_2810_0001&page_name=main)