

Component Based Approach to Modeling for Model Checking

John D. Powell
Caltech, Jet Propulsion Laboratory
John.Powell@jpl.nasa.gov

David P. Gilliam
Caltech, Jet Propulsion Laboratory
david.p.Gilliam@jpl.nasa.gov

Abstract

Vulnerabilities in concurrent software systems and software applications render an otherwise safe and network secure environment unsafe and insecure. Any software system or application added to a safe/secure environment that has exploitable security vulnerabilities affects the security and safety of the whole environment. Thus, a system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities. Therefore, it is critical that software be free from such vulnerabilities.

Vulnerabilities in software arise from a number of development factors; but these vulnerabilities can generally be traced to poor software development practices, new modes of network attacks, mis-configurations, and dangerous interaction between systems.

A formal software assessment methodology can aid in providing a greater level of assurance that software is not exposed to vulnerabilities as a result of defective software requirements and designs or exposures due to complexity and integration with other applications that are developed in parallel or subsequently added to the system.

This paper presents a portion of an overall research project on the generation of a software security assessment instrument to aid developers in assessing and assuring the security of software in the development and maintenance lifecycles. This portion focuses on modeling requirements and early lifecycle designs to discover vulnerabilities that result from interaction between system components that are either under development in a new system or proposed as additions to an existing system or environment. There are early indications that a new approach, the Flexible Modeling Framework (FMF) has promise in the areas of network security as well as other critical areas such as system safety. Information about the overall research effort regarding network security is available at: <http://security.jpl.nasa.gov/rssr>.

Keywords

Model Checking, Security, Safety, Formal Verification, Security Toolset

1. Introduction

The National Aeronautics and Space Administration (NASA) has funded the Jet Propulsion Lab to develop a software security assessment for use in the software development and maintenance life cycle. One major goal of the effort is the use of a formal analytical approach, such as Model Checking (MC), for integrating security into existing and emerging practices for developing high quality software and computer systems.

Software on networked computer systems must be free from security vulnerabilities. Vulnerabilities in software arise from a number of development factors that can generally be traced to poor software development practices, new modes of attacks in the network security arena, mis-configurations, and unsafe interaction between systems and/or their components. An otherwise safe and secure system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities. This presents a verification problem for networked systems because the builders of a system often have little or no knowledge/control over systems that will be linked to it if it makes use of network connectivity. The most extreme case is when a system is connected to the Internet. MC offers a means for examining component interaction in relation to critical system properties such as safety and security. [1,9,10,11,12]

Currently, the use of MC as means of verification to mitigate these vulnerabilities during the software development and maintenance life cycle suffers from some practical limitations. Among these limitations are:

For example, to reconstruct the knowledge contained in Figure 4 in light of a modification to C_1 only C_1 itself and the combination of C_1 and C_2 (And_1) need to be re-verified. The results of re-verifying the remaining components/combination would only replicate the results of the previous set of verifications. This represent a savings in that 6 verifications ($C_1, C_2, C_3, C_4, And_1, And_2$) were needed to originally capture the knowledge in Figure 4 but only 2 re-verifications were needed to reconstruct the knowledge in light of the modification.

This will result in a decreased cycle time for verification of model updates thus improving the timeliness of the formal verification results. Further, as more is learned about the system's specific manner of accomplishing its task(s) the affected model components can be:

- Modified to reflect the more detailed approaches developed during the design phase to maintain model fidelity in a timely manner.
- Segmented into its own series of components when the complexity of the high level component begins to exhibit state space explosion problems. This allows logical grouping of related components while still allowing incrementally inclusion of parts of complex logical system entity in combinations where the state space is reaching MC thresholds.

There are numerous instances in which one must view a system or set of systems at a very abstract level before examining one or more parts in greater detail. In the security arena one will view a large network system from an extremely high level where protocols must be understood and systems within it are arbitrary connected entities. When one is building a system that will interact with a network the focus on that particular system entity becomes more detailed but the remainder of the network is still viewed very abstractly. The levels of detail in which a component/entity is modeled and examined is referred to as its *resolution* throughout this paper. As specific interaction with other systems are defined the resolution of those systems necessarily becomes somewhat more detailed to deal with issues such as:

- How will that system handle transmission from the system being developed?
- In what way will that system respond to the system under development?

Further, the system under development will be decomposed at a high level in to various network aware and non-network aware component, which will be subsequently view with varying levels of resolution. (See Section 2.3) Consider a system where network aware components such as application and login functions are interacting with non-network aware components such as routines on a local printer. At this level there are network aware components on the computer system, components making up the network environment, and components making up the local printer. (See Figure 5) At this level it is appears reasonable to believe that the computer system is interacting with the network and thus should be responsible for security. The printer is local (i.e. non-network aware) thus need not concern itself with network security and may rely wholly on the computer system in that regard. In the next section we will examine variable resolution of components and continue the example to illustrate that the above suppositions may not be true.

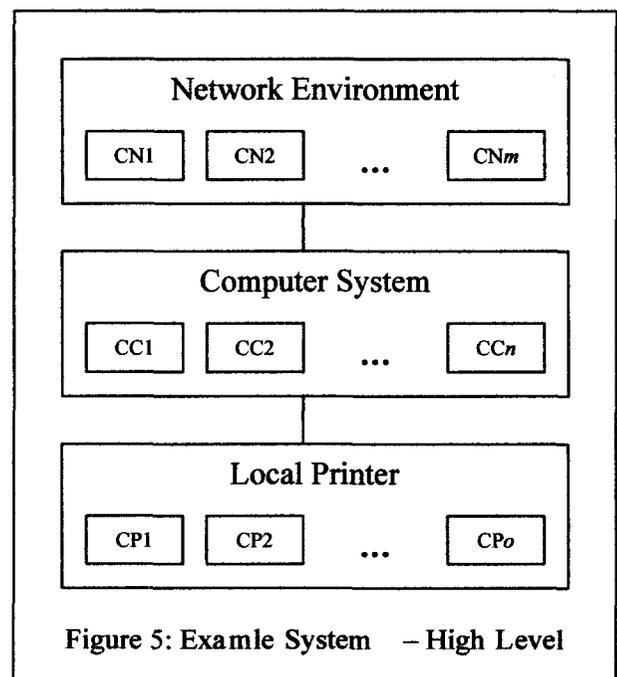


Figure 5: Examlle System – High Level

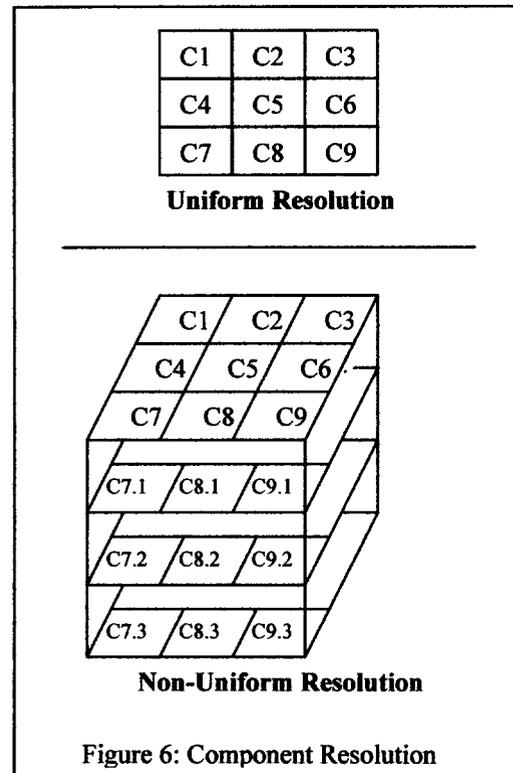
2.3. Non-Uniform Component Resolution

In the FMF approach components are viewed not only as a matrix of combinable components (See Figure 6) created at a given level of detail (resolution) but also as a 3-dimensional space of models where by component versions of different resolutions may be selected to manage state space explosion. Therefore, the tolerable state space maybe spread across several components in varying amounts, which together from a state space that is feasible for MC. The approach of building model components, as opposed to a single model, allows localized modification and enhancement of model behavior and detail in order to examine subsets of the system at various non-uniform levels of resolution. The FMF component methodology provides the ability to make tradeoffs in resolution between components while maintain the size of the state space within tolerable limits. The process of enhancing model components as more is learned about the system results in a series of component versions. When archived for later use they provide a readily usable facility for producing component combinations with non-uniform resolutions. This is done by selecting the components for the combination and then specifying what version (resolution) of the component to use. For example (See Figure 6) assume that:

- C_n is a component at its lowest resolution (least detail)
- $C_{n.3}$ is a component model with the highest resolution (greatest detail).
- All 9 components taken together at the $C_{n.1}$ level of resolution is just under the state space feasibility limit.

With the above assumptions, the ability to investigate one component in detail ($C_{n.3}$ level) is facilitated by accepting lower levels of resolution (C_n level) in other components. Raising the resolution in one component increases the state space. Conversely, lowering resolution decreases the state space. Therefore, to gain maximum benefits from the available memory resources the resolution is increased until the threshold is almost reached. By remaining just below 100% available memory usage, property verification over model component versions whose average resolution is the highest feasible. The analyst may then continue to make tradeoffs to probe various parts of the system

in greater detail by increasing resolution on one component and decreasing it on others.

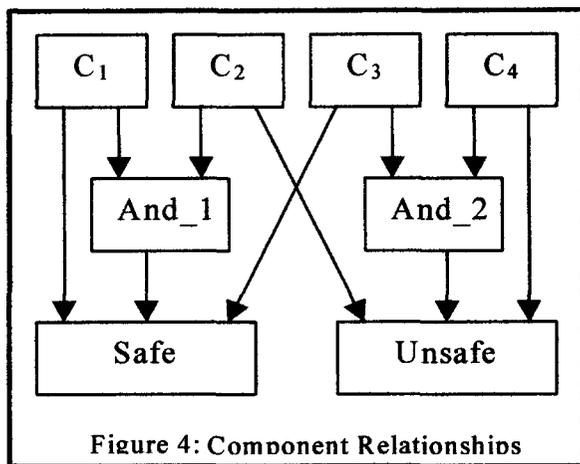


Recall the example from Figure 5 where a printer with no network capability need not concern itself with security issues. To gain a higher confidence that this is the case the components within each system (Network, Computer, Printer) must be examined at a higher resolution. It is apparent that examination of all components at a high level of resolution is an infeasible task for MC with reasonable memory constraints. Thus a process is undertaken to gradually:

- Increase the “resolution” on the computer system and the printer
- Lower the “resolution” on the network environment.
- Lower resolution or rule out some of the non-interacting components within the computer and printer.

This process can reveal useful connections to between the printer components and those components on the computer system that are security critical. (See Figure 7) While printer components like CP1, do not directly interact with the network it now may be possible for them to

later be strategically combined for system verification purposes. This 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 model checking 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.

The FMF verification process begins determining which model components are safe and unsafe with respect to the property in question. Then, the strategic combination process seeks to build up relationships between components. Figure 4 shows an example where the components C_1 and C_3 are safe with respect to some security property while the states C_2 and C_4 are unsafe. Relationships between C_1 and C_2 as well as C_3 and C_4 are shown. Since C_2

is individually unsafe, C_1 is individually safe and the combination C_1 and C_2 is safe, C_1 is said to *mitigate* C_2 with respect to the property in question. Conversely C_3 is safe and C_4 is unsafe and the combination of the two components is unsafe. In this case C_4 is said to *undermine* C_3 .

Network security professionals and builders of network systems are faced with an overwhelming task of either maintaining or defending against many systems to which their system is or will be linked.

These linked systems are often heterogeneous with respect to the software application sets that are running on them. Many will run various combinations and subsets of common software applications along with less common specialized applications. The FMF seeks to retain information about dangerous combinations of software. Further, the framework provides a means by which new combinations encountered by network professionals can be formed from existing models and quickly evaluated for their potential effects on linked systems.

It bears noting that two components that are labeled individually safe may produce an unsafe security condition when combined and vice versa. Similar to testing of an implementation, two modules that have undergone unit testing and passed may later produce problems during integration testing. These problems may be traced to such phenomena as timing issues resulting in race conditions or unexpected interactions due to the absence of reasonableness checks on input values for one or both modules. The FMF allows investigation of such possibilities at an earlier point in the life cycle before an implementation exists.

Maintaining the network of relationships for each property will allow future verifications of the property to be accomplished by noting the relationships that were used to make earlier verification inferences and only re-verifying the relationships affected by a component change or addition as the system evolves. When changes are made to the model only the affected components need be modified and re-verified. By retaining knowledge from previous verifications, the effort of re-verifying properties may be reduced significantly due to the fact that only the changed components, and combinations including one or more of them, need be re-verified.

errors are found in the early lifecycle sample test specification can be preserved for use by the PBT to provide traceability verification.

Model based verification techniques, such as Model Checking, are not without drawbacks. Among them is the inability to model a system with a high degree of fidelity in a timely manner while the system evolves. This is particularly problematic in the earliest stage of development such as concept, requirements and high-level design when the system definition is most volatile. MC's lack of agility limits an analyst's ability to maintain an up to date model and minimize the latency between the introduction of errors and their discovery.

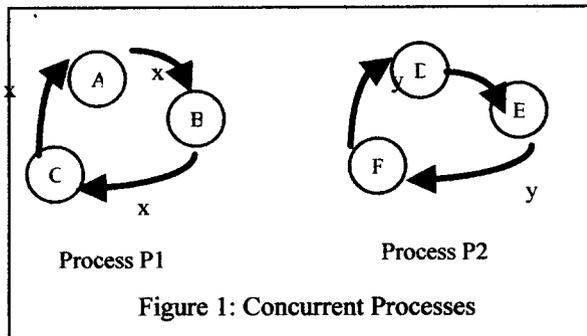
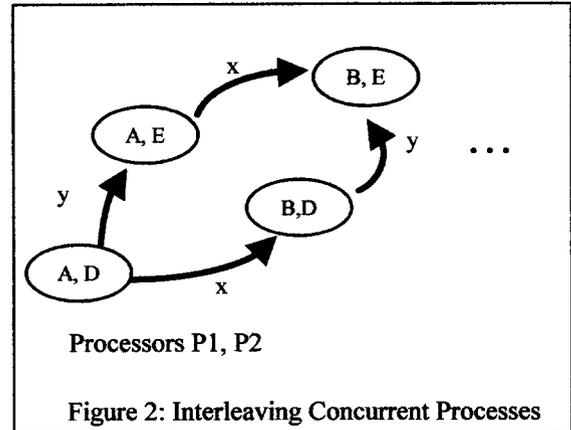


Figure 1: Concurrent Processes

A limitation specific to model checking is the state space explosion problem. [10] Similar to the growth of the operational space mentioned above, the state space that a model checker must search to verify properties grows at an exponential rate as the model becomes more detailed. As shown in figures 1 through 3 the state space grows at a rate of m^n where m is the range of possible values a variable may assume and n is the number of variables in the model. Despite the use of modeling techniques such as abstraction and homomorphic reduction it is infeasible to verify many software systems in their entirety though model checking beyond those that are either complex and very small or moderate in size and very simplistic.



Processors P1, P2

Figure 2: Interleaving Concurrent Processes

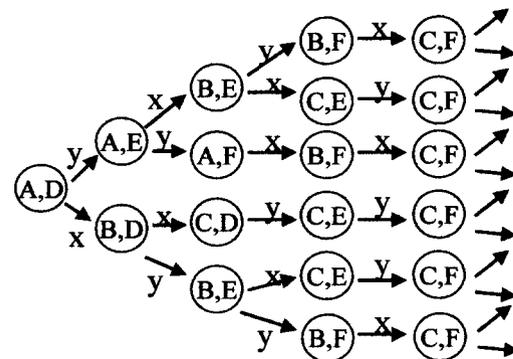


Figure 3: State Space

2.2. The Flexible Modeling Framework

An innovative verification approach that employs model checking as its core technology is offered as a means to bring software security issues under formal control early in the life cycle while mitigating the drawbacks discussed above. The *Flexible Modeling Framework* (FMF) seeks to address the problem formal verification of larger system by a divide and conquer approach. First 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 a:

- System for building models in a component based manner to cope with system evolution in a timely manner
- Compositional verification approach to delay the effects of state space explosion and allow property verification results to be examined with respect to larger, complex models.

Modeling in a component-based manner involves the building of a series of small models, which will

- Limits on the size and complexity of systems that may benefit from MC given reasonable computer memory resources.
- Difficulty in rapid development, modification and verification of models in a timely manner during the early life cycle when systems tend to change and evolve quickly.

The MC approach discussed in this paper seeks to develop a tool-supported methodology to support the model-based verification of software systems in an agile manner that can cope with memory resource constraints to a reasonable degree. Specifically, the *Flexible Modeling Framework* (FMF) offers a formal MC approach for engineering safety and/or network security into software systems and application throughout the software development and maintenance life cycles.

Model based verification uses precise abstractions. It offers the ability to verify security properties over system models early in the life cycle – before an implementation exists. MC can effectively identify security anomalies that have not been discovered as a result of a known network security attack. These new anomalies may then be added to a stored Vulnerability Matrix (Vmatrix). (See Section 3.2) Anomalies that are found in early lifecycle phases through the examination of abstractions (models) can be preserved and later passed on to additional technologies such as the Property Based Tester (PBT) [1,3,4,5] for verification at the code level. (See Section 3.2)

Assessments of high profile NASA systems believed to be vulnerable to network security attacks will provide a metric to determine the effectiveness of these activities and prototypes. The security assessment instrument will be verified on a JPL/NASA Class A Flight Project to assess the approach and the viability of the FMF for assuring the security of software on critical networked systems.

2. Model-Based Security Specification and Verification

Model based specification and verification make use of discrete finite models to verify compliance of the model to desired properties. In the cases

mentioned here, software safety and network security properties. Network security and/or safety properties often focus on characteristics that are manifested through the operation of multiple software components and systems operating concurrently with or without an attacking process. The concurrent nature of the systems results in an operational space that is too large to verify system properties effectively through traditional testing of the implementation. Further, vulnerabilities introduced in the early phases of the development lifecycle are costly to remove in later phases when an implementation is being tested. This results in the addition of cumbersome workarounds and “patches” to repair the software system which themselves could introduce new vulnerabilities. Model Based Specification offers the opportunity to perform verification of properties early in the life cycle, providing a clearer understanding of the vulnerability issues within the system before an implementation exists.

The FMF approach is currently under development and shows promise for early life cycle detection of security vulnerabilities. The approach may be generalized and/or tailored in future work for applicability to non-security domains such as safety.

2.1. Model Checking

Model checkers automatically explore all paths in a finite state space from a given start state in a computational tree. The objective is to verify system properties over all possible scenarios within a model. Model Checkers differ from more traditional heavyweight formal techniques in that:

- Model checkers are operational as opposed to deductive. Deductive approaches, while offering a higher level of completeness and are more resilient in the face of larger systems, are difficult to apply and require a great deal of expertise.
- Model checkers provide counter examples when properties are violated. The counter examples may be used to determine the cause of the property violation and as representative traces for test case generation. [6,7]
- Their goal is oriented toward finding errors as opposed to proving correctness since the model is an abstraction of the actual system. Where

interact with otherwise secure computer system components in a manner that renders them unsecure.

An attack similar to this scenario above has been seen before. The attack managed to exploit interactions between computer system components to send a message to the printer with additional data embedded in it. At a higher level it appears that a component on a computer system is sending a routine message to a local (non-networked) printer – a harmless interaction. Thus, the message is not scrutinized by the computer system because it is a non-network aware activity. Therefore the message is allowed to proceed to a seemingly non-critical area (the local printer). However because the printer did not address network security concerns either it failed to identify that its normal responses back to the computer system were actually giving an unauthorized outside party root access to the computer system. From there the attacker had access to the entire network.

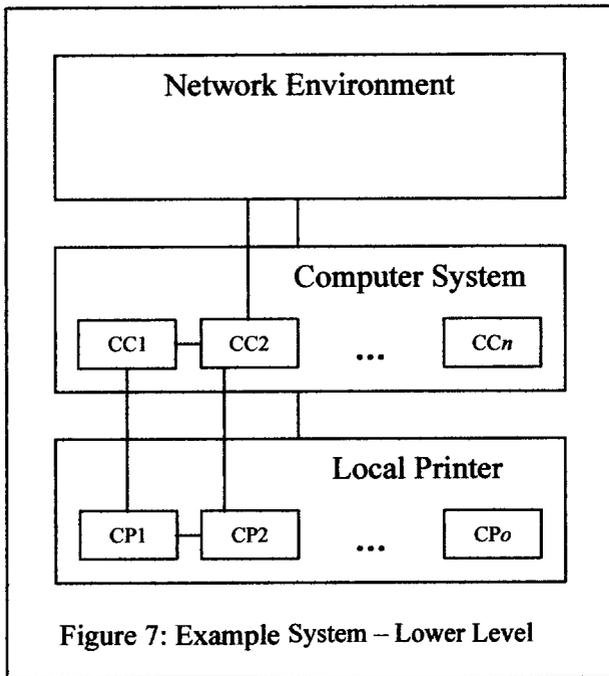


Figure 7: Example System – Lower Level

3. FMF Integration with other Technologies

The FMF is intended as a verification technology that can stand-alone but may also be combined with other technologies to afford maximum benefit and usability. [11,12]

3.1. State chart Representations and Tools

FMF makes use of an internal state chart representation as its means of representing models. This representation currently encompasses a subset of possible state notations. Elements initially supported by the approach are those that are generally interpreted in a consistent manner in most state chart representations. The currently supported notation features are:

- Conditional and unconditional transition between states.
- State definitions that include abstract assignment of system variable state values.
- Finite arithmetic assignment to a state's variable values. For example, $(a = (a+1) \text{ mod } 10)$ is finite while $(a = a + 1)$ is not because the number of visits to the state in a possible execution is generally unknown at the time of verification.

The use of state chart representations, as opposed to modeling languages such as Promela [10], provide the opportunity to employ existing robust state chart specification systems as an interface to FMF tool support capabilities.

3.2. Other Verification Systems

In the network security arena an integrated approach, which includes the FMF as a model-based verification element, for assessing security vulnerabilities is being explored. The other parts of the Security Assessment Instrument are PBT and the VMatrix.

PBT is an approach that allows the analyst to systematically test an implementation for adherence to various properties by making use of a support tool called the Tester's Assistant (TA). [1,3,11,12] First, the property is expressed in a form that the TA accepts as input. Then, an analyst uses the PBT to systematically insert assertions that are pertinent to the property into an implementation and exercises the implementation. The objective is to discover traces through the implementation that produce a non-conforming scenario.

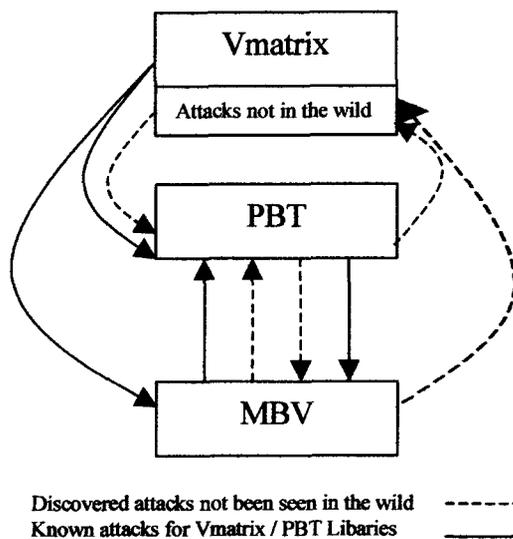


Figure 8: Technology Integration

The Vmatrix, examines vulnerabilities, exposures and the methods used to exploit them. Vulnerabilities and exposures are listed along with their Common Vulnerabilities and Exposures (CVE) listing. [2] The VMatrix includes:

- A brief summary and a description of the vulnerability or exposure.
- The affected software or operating system.
- The means necessary to detect the vulnerability or exposure and the fix or method for protecting against the exploit.
- Catalogue information, keywords, and other related information as available, regarding the vulnerability or exposure.

The individual parts of the Security Assessment Instrument can be used separately or in combination (See Figure 8) to provide the additional benefits of:

- Reduced rework to identify security properties.
- Increased confidence in the system through verification at multiple times during the development and maintenance lifecycle
- One tool is capable of verifying the input and output of other tools in the network security instrument.
- Finding additional network security attacks yet to be seen in the wild (attacks that have not yet been seen outside of a laboratory environment) and test for their viability and severity.

In addition to developing an abstract model of a system and performing MC verification, properties of interest must be defined. The identification of specific system critical properties that warrant formal verification is a non-trivial task. Integration of the FMF with the VMatrix addresses this problem for the network security arena. The VMatrix [1,2] provides a searchable knowledge base from which properties may be extrapolated for use with the FMF in the role of a MC function within the instrument. This knowledge base can also accommodate the discovery of new network security attacks not yet seen in the wild that may be discovered through MC techniques.

The network security instrument also provides a Property based testing tool [1,2,4] that verifies properties against the actual implementation of a software system. These properties are also extracted from the VMatrix. Used with the FMF, PBT can provide verification of a system implementation's fidelity to the model(s) of early lifecycle artifacts (Requirements and Designs).

4. Conclusion

Reducing the number of vulnerabilities in software systems is critical in computer systems that perform safety critical functions and/or make use of network connectivity. The use of formal approaches such as MC enhances the ability of developers and analysts to discover vulnerabilities arising through unsafe interactions between systems and/or otherwise safe software components. The FMF approach provides the benefits derived from MC while mitigating limitations posed by system size and complexity as well as requirements and design volatility during the early lifecycle phases. The FMF attempts to capitalize on the benefits of existing technologies in a manner that maximizes usability and minimizes duplication of effort between approaches.

Integrating software security and safety into existing and emerging practices is critical for developing high quality software. The *Flexible Modeling Framework* (FMF) offers a formal approach achieving such integration throughout the software development and maintenance life cycles. The approach seeks to maximize these benefits by attempting to integrate with, as opposed to

replacing, existing verification technologies. While work on this research effort is ongoing, the approach has shown considerable promise and garnered interest from JPL projects as a means of increasing confidence in the safety and security of software during development.

5. 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, and the University of California at Davis under a subcontract with the Jet Propulsion Laboratory, California Institute of Technology.

6. References

[1] D. Gilliam, J. Kelly, M. Bishop, "Reducing Software Security Risk Through an Integrated Approach," Proc. of the Ninth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (June, 2000), Gaithersburg, MD, pp.141-146.

[2] Published and maintained by Mitre. The CVE listing can be found at: <http://cve.mitre.org/>

[3] G. Fink, M. Bishop, "Property Based Testing: A New Approach to Testing for Assurance," ACM SIGSOFT Software Engineering Notes 22(4) (July 1997).

[4] M. Bishop, "Vulnerabilities Analysis," Proceedings of the Recent Advances in Intrusion Detection (Sep. 1999).

[5] J. Dodson, "Specification and Classification of Generic Security Flaws for the Tester's Assistant Library," M.S. Thesis, Department of Computer Science, University of California at Davis, Davis CA (June 1996).

[6] J. R. Callahan, S. M. Easterbrook and T. L. Montgomery, "Generating Test Oracles via Model Checking," NASA/WVU Software Research Lab, Fairmont, WV, Technical Report # NASA-IVV-98-015, 1998.

[7] P. E. Ammann, P. E. Black and W. Majurski. "Using Model Checking to Generate Test Specifications," 2nd International Conference on Formal Engineering Methods (1998) pp. 46-54.

[8] G. Lowe. Breaking and Fixing the Needham-Schroeder Public Key Protocol Using CSP and FDR. In TACAS96, 1996.

[9] W. Wen and F Mizoguchi. Model checking Security Protocols: A Case Study Using SPIN, IMC Technical Report, November, 1998.

[10] G. Holzmann. Design and Validation of Computer Protocols. Prentice Hall 1990; ISBN: 0135399254 .

[11] 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.

[12] D. Gilliam, J. Powell, J. Kelly, M. Bishop, "Reducing Software Security Risk Through an Integrated Approach", IEEE Goddard 26th Annual Software Engineering Workshop.