Using Software Security Analysis to Verify the Secure Socket Layer (SSL) Protocol

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

Abstract

The National Aeronautics and Space Administration (NASA) have tens of thousands of networked computer systems and applications. Software Security vulnerabilities present risks such as lost or corrupted data, information theft, and unavailability of critical systems. These risks represent potentially enormous costs to NASA. The NASA Code Q research initiative "Reducing Software Security Risk (RSSR) Trough an Integrated Approach" offers, among its capabilities, formal verification of software security properties, through the use of model based verification (MBV) to address software security risks. [1,2,3,4,5,6] MBV is a formal approach to software assurance that combines analysis of software, via abstract models, with technology, such as model checkers, that provide automation of the mechanical portions of the analysis process. This paper will discuss:

- *The need for formal analysis to assure software systems with respect to software and why testing alone cannot provide it.*
- *The means by which MBV with a Flexible Modeling Framework (FMF) accomplishes the necessary analysis task.*
- *An example of FMF style MBV in the verification of properties over the Secure Socket Layer (SSL) communication protocol as a demonstration.*

1. Introduction

Software security attacks are increasing, not only in number, but in sophistication as well. Further, the risk of the threat of security break-ins as a means for conducting asymmetrical warfare (i.e. cyber-warfare) has been identified as a probable scenario. With the advent of cyber-warfare threats, the nature of the attacker expands from individuals or small groups to large, sophisticated, well funded organizations of paid professionals whose sole job is to defeat security measures and damage, or render useless, critical software systems.

There are two fundamental conditions that must be considered when applying assurance activities to the problems associated with software security.

- First, a system that resides in a networked environment is an open-ended system that has limited control, at best, over the systems with which it interacts and how that interaction takes place.
- Second, when a system experiences a break-in, it is due to a purposeful and intelligent entity or adversary engineering malicious events as opposed to an *unmotivated environmental event.*

In this paper, the term *motivated environmental event* is defined as an event directed at a system from the environment that is purposely initiated by an intelligent attacker to intentionally damage the system or interfere with its intended operation. Conversely, an *unmotivated environmental event* is an event that naturally exists in the system's environment and occurs with some probability (frequent or rare) without regard for the system or the damage it may or may not inflict.

2. The Need for Analysis

Unmotivated environmental events can be regarded as having some probability of occurrence that is less than 100%. A software system is built and tested against those harmful environmental events that are known, with priority given to those that are closer to 100% in probability of occurrence. Some unmotivated events with very low probability of occurrence are even ignored. Since unmotivated events in a software system's environment are not actively seeking out the system, the risk from events with a low likelihood of occurrence may be assumed as opposed to defended against. However, a motivated event is the result of an attacker actively seeking out harmful environmental event sequences and purposefully initiating them to harm vulnerable software systems. These harmful

sequences, if known by an attacker, must be regarded as having an occurrence probability of 100% because they are purposefully set in motion. The only barrier between the software system's security measures and the motivated environmental event is a lack of knowledge on the part of attackers that the successfully harmful event sequence(s) exist. Malicious adversaries discover harmful event sequences, not through testing but through a process of analysis of known parts of a software system that includes:

- Probing the system
- Collecting Data
- Analyzing the data to discover vulnerabilities.
- Formulating exploits of the vulnerabilities.

The key point that is made clear by this process is that the simple testing of previously known cases of system exploitation is not the driving force that facilitates dangerous new break-ins. This is due in large part to the fact that most critical "trophy systems" that interest attackers most are maintained by system security professionals that have effectively guarded the system against previously known exploitations in response to past attacks. New exploits are derived from analysis of system behavior in multiple contexts to discover an interaction that has not yet been considered by an organization's security professionals.

Testing alone cannot provide assurance for software security. Reliance solely on testing for improving software security is a primary reason for the "attack and patch" cycle in which the industry currently finds itself. Since potential attackers employ analysis of a software system, and its security defenses, to discover a system weakness, organizations must make similar types of system analysis a regular part of their system security practices.

Testing, by an attacker, can only be performed after the harmful event sequences are identified. The only time attacker testing takes place is in the form attacks perpetrated against vulnerable systems. Thus, by the time a test case is available for testing by the victim organization the attack has already occurred. When an attack is successful, patches are subsequently devised by reverse engineering the attackers analysis from information generated by the attack. The patches are then tested against the test case produced by the attacker's analysis. However, by this time, attackers' consistent focus on analysis has devised an entirely new class of attacks.

The only way to increase assurance of a software system's security defenses is through analysis with testing as a follow-on activity. Many software systems and their security defenses are already under analysis. Unfortunately, the current paradigm often involves attackers performing the analysis. The system owners, as opposed to attackers, must begin to perform analysis of software systems and their defenses. This is the only way to:

- Provide assurance of software security.
- Reduce reliance on the "attack and patch" cycle.
- Achieve any anticipatory advantage over future classes of attacks.

Technologies such as MBV and other formal methods offer a means to perform these analyses.

3. Model Based Verification

MBV, as it is used in this research, makes use of discrete finite models to verify critical system properties. The FMF is a generic approach to modeling and verification. However, the specific MBV and FMF properties addressed in this paper focus on software security pertaining to the SSL protocol.

Network security properties often focus on characteristics that are manifested though the operation of multiple software components operating concurrently. The concurrent nature of the systems results in an operational space that is too large to verify by traditional testing techniques. MBV with the FMF offers a method of verification of critical system security properties early in the development lifecycle before an implementation exists. This makes MBV valuable because software security vulnerabilities introduced in the early lifecycle phases are costly to remove in later phases. A vulnerability that goes undetected until after system deployment results in the addition of cumbersome "patches" to mitigate the vulnerability. These "patches" may introduce new vulnerabilities in addition to mitigating the ones being corrected.

3.1. Model Checking

MBV with the FMF uses Model Checking (MC) as a core technology. MC verifications, based on discrete finite models, can be used to verify and check compliance to desired security properties. Many security properties cannot be verified by test activity alone. However, verification through analyses and modeling at the design stage can increase the confidence that the specification provides a sound base for developing a secure application, system or communication protocol. The analysis and modeling process can begin early in the software development life cycle and continue into implementation. Modeling tools and languages used together provide a machine-readable model that facilitates automated verification

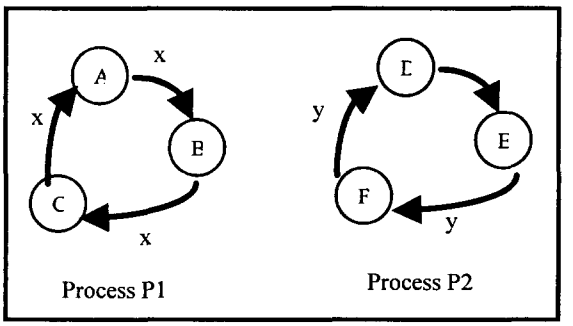

Figure 1: Concurrent Processes

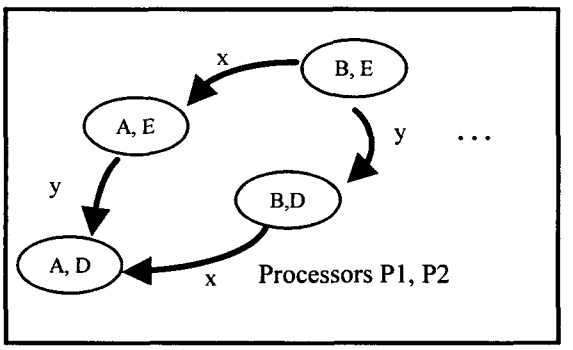

Figure 2: Interleaving of Processes

of system properties. Models must be updated and re-verified periodically, as requirements and designs become more mature. Analysis of up-to-date models can contribute to verification by testing programming code through test case generation from the Model Checking analyses. [8, 9]

Software model checkers automatically explore all paths from a start state in a computational tree (See Figures 1 & 2). 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 an abstract 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 such as formal proofs (and theorem provers) 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 an abstraction of the actual system

The MBV techniques, using MC as a core technology, exhaustively explores a system's finite operational state space. The objective is to verify system properties over all possible system scenarios. MC also provides counter examples when properties are violated, which are then used as traces for test case generation. [7,8,9]

MBV techniques, such as MC, are not without drawbacks. These include:

- MBV's resistance to fast adaptation of system models. This hinders MC's ability to evolve a system model in a timely manner when the system definition is volatile.
- The state space explosion problem inherent in model checking. [10] The operational state space that a model checker must search to verify properties grows at an exponential rate as the model becomes more detailed in response to system's that are large and/or complex.

3.2. The Flexible Modeling Framework

The FMF is offered as a means to bring software security issues under formal control while mitigating the drawbacks of MC discussed above. The FMF seeks to achieve this by a divide and conquer approach. As such, the FMF is a: 1) System for building models in a component based manner to cope with system evolution in a timely manner, 2) Compositional verification approach to delay the effects of state space explosion for larger and/or complex system models.

Modeling in a component-based manner involves the building of a series of small sub-models. Then, these components can be combined and verified over system properties of interest in a compositional manner.

The compositional verification approach 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:

- Infer verification results over systems that are otherwise too large for MC from the results of strategic overlapping subsets of the system in the form of model component combinations.
- Retain verification results from individual components and component combinations to increase the efficiency of subsequent verifications.

4. Model Based Verification of the SSL Protocol

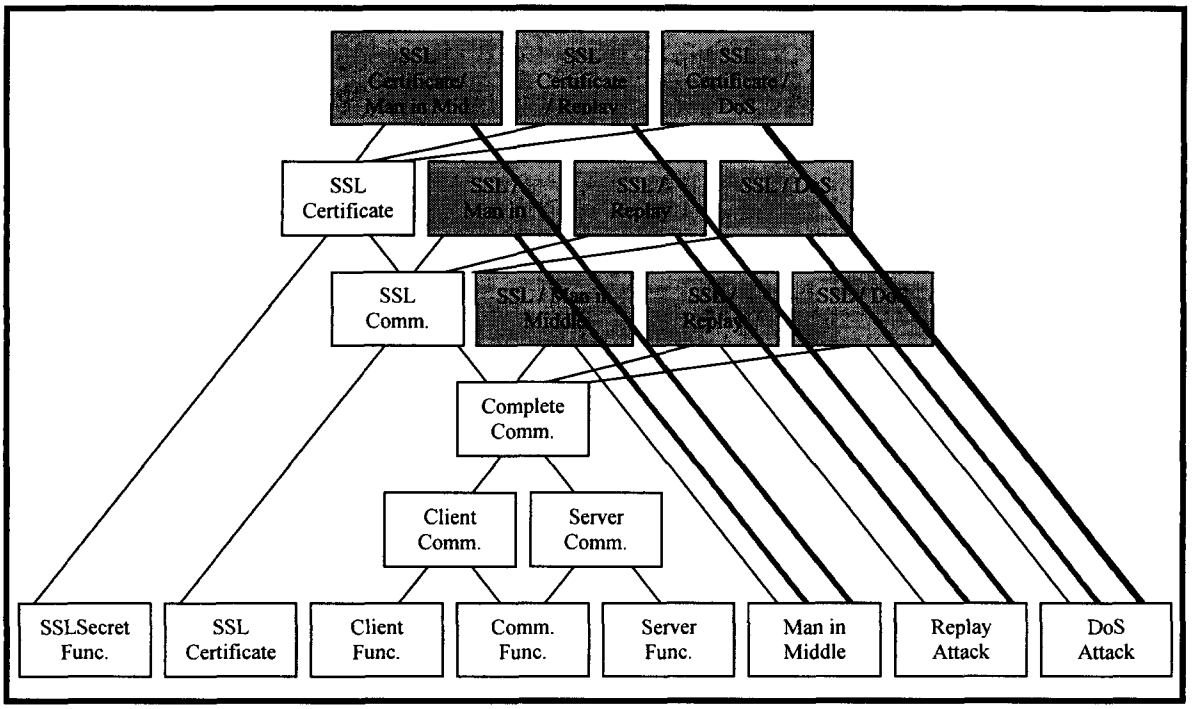

Figure 3: SSL Communication Protocol Model Component

As an example of a verification activity performed at JPL using MBV with the FMF, the SSL protocol was examined. The SSL protocol was modeled along with three potential classes of attacks in a component based manner. Properties of interest focused on the SSL protocol's ability to avoid falling prey to each class of attacks by recognizing the onset an attack taking appropriate action. The application of SSL protocol against the attack classes at varying degrees of rigor allowed by the SSL specification was analyzed by using the FMF. It is important to note that the verification of the SSL protocols resilience under attack did not involve testing each individual known instantiation of attacks but analysis of entire classes of attacks simultaneously. Further, analysis of the varying degrees of SSL rigor was a built-in side effect of the FMF verification methodology of verifying multiple overlapping component combination subsets during the verification of SSL as a whole. This not only produced results of SSL handling attacks but also identified the portions of the SSL specification that are essential in defending against each attack class.

Figure 3 shows how SSL model components can be mixed and matched within the FMF to verify correctness properties over multiple variations of SSL behavior. Development of a single model containing all possible behaviors can be counter-productive. Combining behaviors that do not reasonably co-exist in a system produces many false property violations. False violations under those conditions would flood the analyst with so much data to review that the timeliness of verification results would be compromised. Further, upon finding a valid violation of a system property in the environment of an overall model, the counterexample will often be convoluted by irrelevant interim model transitions. Thus, isolating and recommending corrective action in that environment becomes a long and tedious analysis task. When the model is separated into variations through the use of FMF, valid verification knowledge can be easily extracted from the pattern of violations and non-violations over the model variations. The FMF approach is a means for determining critical system functionality with regard to software security properties thereby isolating vulnerable areas for corrective actions. Finally, in an open system, such as the SSL protocol and its environment, an all-encompassing model will unduly stress the limits of the test platform's memory constraints due to excessive state space explosion without the use of the FMF.

Four SSL correctness properties were verified over the FMF model components. They are

1. The SSL secure communication shall initialize eventually unless less an attack has successfully inserted itself in such a manner that the resulting secure communication will be compromised

Each communicating entity will eventually achieve and execute the exchange of secure communication	No Attack	Man in the Middle Attack	Replay Attack	DoS attack
Signed SSL Entities (Certificates)	No Violation	Violation	Violation	Violation
Unsigned SSL Entities (No Certificates)	No Violation	No Violation	Violation	Violation
Non-SSL Client Server Entities	No Violation	No Violation	Violation	Violation

Table 1: Verification Results Summary

2. Once secure communication is establishhed secure contacts and responses will always be reached
3. A secure message that has been intercepted shall be detected and not accepted by the SSL recipient of the secure message
4. Under the rules for attacks, an attack may only read unsecured messages or secured messages if the SSL secret has previously been captured.
5. Securely communicating entities shall not reveal their secret even during the handshake initialization.

The significant verification results shown in Table 1 indicate that:

1. The absence of a violation in the "No Attack" indicates that SSL entities communicate correctly when no attack is present. This provides and early baseline that the model is reasonable.
2. Only the SSL entities using signed certificates recognized a Man-in-the-Middle Attack. The violation in the "Man in the Middle Attack" column corresponding to SSL entities with certificates indicates that communication was correctly aborted in before exposing secure communication in response the recognized attack.
3. The Replay Attack failed to access secure communication. The model did not distinguish between behaviors of SSL versus non-SSL entities in terms of the property verification. The violations resulted from the attack starving off effective communication between the entities. Therefore, while the attack was successful in disrupting intended system operations it did not specifically defeat SSL in that secure information was not accessed
4. The DoS Attack did deadlock the system but did not capture secure communication. The DoS attack had effects similar to the Replay attack in that SSL was not specifically defeated but the system was disrupted.

Further analysis shows that Replay attacks and DoS attacks are related at a very high level. In a hierarchical classification of attack types, the Replay and DoS attacks will likely fall into the same class of attacks at the middle to upper levels of the hierarchy. This information is valuable in making overall software security recommendations during the development lifecycle. At the design level, it is highly likely that one set of principles will address both types of attacks. This will simplify the problem in terms of building security into the software system prior to its implementation.

5. Conclusion

Testing is an important part of building security into software under development and responding to break-ins and vulnerabilities discovered after deployment. However, formal analysis must be employed along with testing if organizations are to provide continued assurance that critical networked systems are as secure as possible. Formal analysis compliments traditional testing by revealing the dangerous core catalysts of attacks. Further, these catalysts often show that many seemingly unrelated attacks could be treated as a single class of vulnerabilities for purposes of securing critical systems. MBV used in conjunction with the FMF offers a means to analyze vulnerabilities, such as those described in the SSL case study, early in the software development lifecycle. By addressing vulnerabilities and security issues before an implementation exists the cost of correction is decreased while the resilience of the system in the face of attacks is increased.

6. Acknowledgement

The work described in this abstract was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

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

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

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

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

[5] D. Gilliam, et. al., "Development of a Software Security Assessment Instrument to Reduce Software Security Risk" Proc. of the 10th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Boston, MA, pp144-149.

[6] D. Gilliam, et. al., "Reducing Software Security Risk Through an Integrated Approach", IEEE Goddard 26th Software Engineering Workshop

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

[8] J. Callahan, et. al. "Generating Test Oracles via Model Checking," NASA/WVU Software Research Lab, Fairmont, WV, Tech. Rpt #NASA-IVV-98-15.

[9] 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.

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