Challenges and Future Directions of Software Technology: Secure Software Development*

(Invited Paper)

Bhavani Thuraisingham and Kevin W. Hamlen
Computer Science Department
The University of Texas at Dallas
Richardson, Texas, USA
{*bhavani.thuraisingham,hamlen*}*@utdallas.edu*

***Abstract*—Developing large scale software systems has major security challenges. This paper describes the issues involved and then addresses two topics: formal methods for emerging secure sytems and secure services modeling.**

I. Introduction

Large scale software development is one of the biggest challenges faced by corporations. Incorporating security into the software development process is even more challenging. Some of the systems need end-to-end security while some others have to carry out the mission securely even if the components may be compromised. Secure software development involves many aspects, from security policy definition, formal modeling, developing security architecture and software models, testing verification and validation, and finally evaluation, certification, and accreditation. It does not end here. The process continues throughout the life cycle of the software system.

Today we are developing new types of software architectures including service oriented architectures and object management systems. These systems use new paradigms for computing and need novel security features. There is not a lot of work on incorporating security into object systems as well as service oriented systems. However, there are few efforts on developing approaches for incorporating security into the modeling process for such systems.

In this paper we discuss two security challenges that have to be considered in building evolvable and flexible secure systems. One is on applying formal methods needed—especially those that need high assurance; the other is secure services modeling of systems. In Section II we discuss security for overall software development. We discuss formal methods for software development in Section III. We discuss secure service modeling in Section IV. The paper is concluded in Section V.

II. Secure Software Development

The first step in developing a secure system is to develop the security policy. The policy has to be consistent with the organization's security policy that is developed by the executive management. The security professionals of the corporation have to work with the secure systems research and development professionals in coming up with the security policy. The next step is to develop a security model. This model could be based on formal principles if the level of assurance has to be high. Once the modeling is completed, the system is designed. This is a crucial stage as it is during this stage that the security-critical components have to be identified. If the system is composed of many components (or services) and the components come from different organizations, the security professionals have to take into consideration this aspect and ensure that the overall system has sufficient assurance.

The next step is to develop the system followed by security testing. This testing may include penetration testing carried out by an independent organization. In addition, unit testing and integrated testing are also carried out. Once the system is designed, developed, and tested, it is then evaluated by agencies such as the National Computer Security Center. Many of these agencies now use the Common Criteria to evaluate the system. After the evaluation, the corporations that use the system have to conduct certification. This ensures that the system operates securely. Finally, the management will accredit the system. There are several challenges that need to be addressed throughout the process. We will explore two aspects. One is applying formal methods for emerging secure systems in the verification and validation process, and the other is security modeling of the applications based on the services paradigm.

III. Formal Methods for Secure Systems Development

The task of reliably enforcing and verifying high-level security policies at a (relatively low-level) software level is a notoriously difficult problem whose roots can be traced back to the original incomputability results of Gödel and Turing. For example, even simple policies, such as access control policies with only one principal, one object, and one permission, turn out to be program properties that are in general undecidable [1]. Traditional formal methods that attempt

*This research was supported by Young Investigator Award FA9550-08-1-0044 provided by the U.S. Air Force Office of Scientific Research.

to prove software policy-adherence purely statically have therefore historically faced an array of daunting tractability challenges, such as state-space and prove-size explosion issues for model-checkers and theorem-provers when they are applied to large, real-world systems and complex, real-world policies.

An exciting, emerging alternative to traditional formal methods involves *provably correct runtime monitoring* [2], [3]. Here, some of the tractability burden of purely static verification is shifted to runtime. The result is a dramatic increase in verification tractability at the expense of a marginal increase in runtime overhead for the verified code. The increase in runtime overhead is minimized through the use of *certified in-lined reference monitors* [3], [4], which instrument the untrusted code with runtime security checks that provably suffice to enforce the policy.

As an example, consider a classic confidentiality policy that prohibits untrusted applications from performing network-send operations after they have read from a confidential file. Formally verifying a large software system for adherence to this policy traditionally requires numerous heavy-weight code analyses, such as interprocedural (and even inter-module) control-flow and dataflow analyses that attempt to trace all possible flows from file-read to network-send operations in the untrusted code. The policy is in general undecidable, so any purely static, sound analysis is subject to false positives, wherein the verifier cannot prove that certain flows are safe. Eliminating these false positives typically requires human experts to then assist the verifier by formulating appropriate preconditions, post-conditions, and loop invariants. In total the effort can easily escalate to tens or even hundreds of thousands of man-hours for a large system.

In contrast, a certifying in-lined reference monitor (IRM) framework simply replaces operations that are not provably safe with guarded operations that detect and prevent policy violations at runtime. In the case of the policy above, the IRM might inject a new *state variable* into the untrusted code that tracks whether a read of a confidential file has yet occurred. The new guard operations set the variable immediately after each confidential file-read and consult the variable before any network-send. The result is *self-monitoring code* that is guaranteed to satisfy the security policy when executed.

Key to this approach is the insight that IRM-instrumented code is typically far easier to formally verify that the original code. In particular, a verifier need only prove that each inserted guard suffices to prevent a local policy violation, and that the guards are not circumventable by the surrounding untrusted code. This greatly reduced verification facilitates far more light-weight, automated verification of IRM code. Past work has implemented certified IRM systems for Java bytecode, .NET bytecode, and ActionScript bytecode architectures using simple, light-weight type-checking and model-checking technologies for fully automated formal verification [3], [5]–[7]. Our current work involves developing a certifying IRM system for x86 binary code applications.

An emerging challenge in this research therefore involves the development of hybrid systems that span the gap between purely static and purely dynamic policy enforcement. This introduces a useful tradeoff between verification tractability and the runtime overhead imposed by security. That is, stronger static verification reduces runtime overhead by provably eliminated unnecessary runtime security checks, whereas more efficient runtime enforcement of software security policies reduces the static verification burden by simplifying the verification problem. Further development of such systems will therefore lead to a range of options whereby verification and runtime efficiency may be balanced to form high-assurance yet practical security for large, real-world software systems.

IV. Secure Services Modeling

Security has been incorporated into the software engineering life cycle and more recently on the object-oriented life cycle. For example, secure software design and development includes defining security policies, incorporating security into the design of the system, developing security architectures, and then security testing and maintenance. In the case of object-oriented system life cycles, security considerations will include defining the security policies on objects and their activities as well as incorporating society into the design of the object system and the security testing and maintenance. Similarly, in the case of secure service-oriented life cycles, we need to determine the security policies, the security levels of the services, and the interactions between the services, including the composition of the services, incorporating security into the design and development of the services, and subsequently testing the secure services.

In his book on SOA, Thomas Erle explained the service life cycle [8]. He stated three ways to develop services: one is the top-down approach, the second is the bottom-up approach, and the third is what he called the agile approach. Security cannot be an afterthought in the design of services; one has to consider security throughout all three approaches. In the top-down approach, one has to conduct analysis, then design the services, develop the services, test the services, integrate the services, and then maintain the services. Here, security policies have to guide throughout the process. For example, when two services are composed, what is the resulting policy on the composed service? In the bottom-up approach, services are designed and developed as needed. Therefore, as services are designed, security has to be considered. For example, when a new service is designed, it should not violate the security policies specified for the prior services. In the agile approach, an integrated approach is used. That is, the application is analyzed and the services are identified. However, one does not have to wait until

all the services are identified. Security impact on this agile approach is yet to be investigated.

Another aspect when considering security is dynamic policies. That is, security policies enforced on the services and service compositions may change with time. The challenge is to ensure that there is no security violation when accommodating changing policies and security levels. This is also a major challenge in designing secure service-oriented systems.

In developing secure service-oriented analysis and design approaches, the first step is to analyze the application and determine the services that describe the applications. The logic encapsulated by each service, the reuse of the logic encapsulated by the service, and the interfaces to the service have to be identified. From a security policy view, in defining the services we have to consider the security policies. What is the security level of the service? What are the policies enforced on the service? Who can have access to the service? When we decompose the service into smaller services, how can we ensure that security is not violated? For example, Service A may not have access to Service B. However, Service B may be decomposed into Services C and D wherein A has access to C but not to D. Now, if A has access to both C and D then the policy that prohibits A from accessing B might be violated.

The next step is for the relationship between the services, including the composition of services, to be identified. In a top-down strategy, one has to identify all the services and their relationships before conducting the detailed design and development of the services. For large application design, this may not be feasible. In the case of bottom-up desgin, one has to identify services and start developing them. In the agile design both strategies are integrated. From a security policy view, there may be policies that define the relationship between the services. The example we gave earlier regarding services A, B, C, and D shows that while A may have access to C, A may not have access to D if we are to enforce the policy that prohibits A from accessing B. Here, access means invoking a particular service.

Note that business logic in applications could be modeled as services. Furthermore, such an approach sets the stage for orchestration-based service-oriented architectures. Orchestration essentially implements workflow logic that enables different applications to interoperate with each other. Also, we have stated orchestrations themselves may be implemented as services. Therefore, the orchestration service may be invoked for different applications that are also implemented as services in order to interoperate with each other. Business services also promote reuse. From a security point of view, we have yet to determine who can involve the buisness logic and orchestration services. A lot of work has gone into security for workflow systems including the BFA model [9]. Therefore, we needed to examine the principles in this work for business logic and orchestration services. When a service is reused, what happens if there are conflicting policies on reuse? Also, we have to make sure that there is no security violation through reuse.

The main question is, "How do you define a service?" At the highest level, an entire application, such as order management, can be one service. However, this is not desirable. At the other extreme, a business process can be broken into several steps and each step can be a service. The challenge is to group steps that carry out some specific task into a service. However, when security is given consideration, then not only do we have to group steps that carry out some specific task into services, we also have to group steps that can be meaningfully executed. If security is based on mutilevel security, then we may want to assign a security level for each service. In this way, the service can be executed by someone cleared at an appropriate level. Therefore, the challenge is to group steps not only meaningful from a task point of view but also meaningful from a security point of view.

Next we must examine the service candidates and determine the relationships between them. One service may call other services. Two services may be composed to create a composite service. This would mean identifying the boundaries and interface, and making the composition and separations as clear as possible. Dependencies may result in complex service designs. The service operations could be simple operations such as performing calculations, or complex operations such as invoking multiple services. Here again, security may impact the relationships between the services. If two services have some relationships between them then both services should be accessible to a group of users or to users cleared at a particular level. For example, if Service A and Service B are tightly integrated, it may not make sense for any service to have access to A and not to B. If A is about making a hotel reservation and B is about making a rental car reservation, then an airline reservation service C should be able to involve both services A and B.

Once the candidate services and the service operations are identified, the next step is to refine the candidates and state the design of the services and the service operations. Therefore, from a security point of view, we have to refine the services and service operations that are not only meaningful but also secure. Mapping of the candidate service to the actual service has to be carried out according to the policies. Next we will briefly examine secure service oriented modeling approaches. More details can be found in [10].

Secure SOMA: IBM's SOMA implements service-oriented analysis and design (SOAD) through the identification, specification, and realization of services, components that realize the service components, and flows that can be used to compose services. With secure SOMA, we need to identify the policies enforced on the services and the various components. For multilevel secure web services, we also need to assign security levels of services. In addition, the

execution level of services should also be defined.

Secure SOMF: SOMF (the Service-Oriented Modeling Framework) is a service-oriented development life cycle methodology and offers a number of modeling practices and disciplines that contribute to successful service-oriented life cycle management and modeling. The security impact on this framework needs to be examined.

Secure UML for Services: Secure UML for services essentially developed secure UML for service-oriented analysis and modeling. Several efforts on applying UML and other object-oriented analysis and design approaches for secure applications have been proposed. We need to extend these approaches to secure SOAD. We also need to examine the security impact on service-oriented discovery and analyss modeling, service-oriented business integration modeling, service-oriented logical design modeling, service-oriented conceptual architecture modeling, and service-oriented logical architecture modeling.

V. Summary and Directions

This paper has provided a brief overview for some of the security challenges in building software systems. We have focused on two aspects: one is formal methods for secure systems and the other is secure services modeling. We started with a discussion of secure OOAD. Then we discussed the concept of secure service-oriented life cycles. This was followed by a discussion of secure SOAD and secure services modeling. Finally, approaches to secure SOAD were discussed. We discussed the challenges in reliably enforcing and verifying high-level security policies at a software level. We argued that traditional, purely static, formal methods often face significant tractability challenges, but emerging hybrid, static-dynamic solutions, such as certifying in-lined reference monitors, offer great promise for safely alleviating much of the verification burden. In secure services modeling we discuss some high-level concepts and argued the need to examine the various approaches and examine security.

There is still a lot of work to do in these fields as well as in other areas, including software testing, evaluation, and certification, especially for secure object systems and secure services. Finally, one of the major challenges today is to build secure systems and applications even if the components may be compromised. For example, the components of the supply chain may come from different corporations. One cannot guarantee assurance for these systems. Therefore, system developers have to ensure that the overall system is secure even if the parts might be insecure.

References

[1] K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Computability classes for enforcement mechanisms," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 1, pp. 175–205, January 2006.

[2] I. Aktug, M. Dam, and D. Gurov, "Provably correct runtime monitoring," in *Proc. 15th International Symposium on Formal Methods*, May 2008.

[3] K. W. Hamlen, G. Morrisett, and F. B. Schneider, "Certified in-lined reference monitoring on .NET," in *Proc. ACM Workshop on Programming Languages and Analysis for Security*, June 2006, pp. 7–16.

[4] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, February 2000.

[5] M. Jones and K. W. Hamlen, "Disambiguating aspect-oriented security policies," in *Proc. 9th International Conference on Aspect-Oriented Software Development*, March 2010, pp. 193–204.

[6] K. W. Hamlen and M. Jones, "Aspect-oriented in-lined reference monitors," in *Proc. ACM Workshop on Programming Languages and Analysis for Security*, June 2008, pp. 11–20.

[7] M. Sridhar and K. W. Hamlen, "Model-checking in-lined reference monitors," in *Proc. 11th International Conference on Verification, Model Checking, and Abstract Interpretation*, January 2010, pp. 312–327.

[8] T. Erl, *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. New Jersey: Prentice Hall, 2005.

[9] E. Bertino, E. Ferrari, and V. Atluri, "The specification and enforcement of authorization constraints in workflow management systems," *ACM Transactions on Information and System Security*, vol. 2, no. 1, pp. 65–104, 1999.

[10] B. Thuraisingham, *Secure Semantic Service Oriented Information Systems*. CRC Press, 2010.