웹 컴포넌트 및 소프트웨어 보안 설계에 대한 분석

김정태
목원대학교

Analyses of Design for Software Security and Web Component

Jung-Tae Kim
Mokwon University
E-mail : jtkim3050@mokwon.ac.kr

요 약

This paper explores how to characterise security properties of software components, and how to reason about their suitability for a trustworthy compositional contract. Our framework provides an explicit opportunity for software composers as well as software components to test a priori security properties of software components in a system composition. The proposed framework uses logic programming as a tool to represent security properties of atomic components and reason about their compositional matching with other components.

Ⅰ. Introduction

Software on networked computer systems must be free from security vulnerabilities. Security vulnerabilities in software arise from a number of development factors that can generally be traced to poor software development practices, new modes of attacks, mis-configurations, and unsecured links between system. An otherwise secure system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities. A software security assesment instrument can aid in providing a greater level of assurance that software is not exposed to vulnerabilities as a result of defective software requirements, design, code or exposures due to code complexity and integration with other applications that are network aware. A life cycle process that includes security assurance is needed for improving the overall security of software. Software on networked computer systems must be free from security vulnerabilities. Security vulnerabilities in software arise from a number of development factors that can generally be traced to poor software development practices, new modes of attacks, mis-configurations, and unsecured links between systems. An otherwise secure system can be compromised easily if the system or application software on it, or on a linked system, has vulnerabilities.

II. Security Mechanism

Software Safety Engineering has become a required business endeavour. It has been executed on aircraft, missiles, warheads, medical devices, trains, and should soon be making it's way into communications systems and even grocery stores. As fast as an analysis techniques were derived, applied, and corrected, technology took still another step forward, perhaps even on the same development project!

Meanwhile, we developed management plans which allowed us to integrate several design, development, analysis, and test techniques over an entire product life cycle. This has provided a solid baseline from which to extend and continuously improve. Software safety attempts to guide design through requirements and trade studies to infuse safety and defuse hazards from the start. The identifying and assessing of risk is done so that we can allocate resources against safety-critical code and determine remaining risk. An important service which helps lower risk is to bring the specific hazards to the attention of the appropriate designer or coder and to brief management and the customers. Once Safety identifies the initial hazards, requirements are levied against those requiring fixing, and in what priority. The key reason is to eliminate or control the hazard. Later analyses will statically ensure the hazard has indeed been controlled and no other hazards were inadvertently introduced by implementation. Dynamic testing can prove that a hazard actually was designed out as the requirement states. Without requirements, however, there probably will not be specific code addressing the hazard nor tests to illustrate the removal of the hazard. Unless specific contracting language includes it, separate safety testing will not occur. Imagine an Application Specific Integrated Circuit(ASIC) which is well into manufacturing layup when a safety hazard is discovered, which the customer demands be fixed. There will be perhaps hundreds of thousands in engineering, hours, manufacturing, test plans, and documentation reworked while the hazard is removed or mitigated. A special test point may have to be drawn out of the circuit to allow testing that device on-line or to insert a fault in functional qualification tests. The identity of each hazard must be known and the requirement against it must be testable (have some external lead perhaps). These two points need to be front-loaded (designed-in) or it will cost the company money to step back and fix it. Concurrent engineering will either live or die (economically) by the ability to find hazards early.

A vulnerability is the result of a software defect that an attacker can use to gain illegal access to－or negatively affect the security of－a computer system. A vulnerability's high-level schema includes a context description (platform, operating system, language, and so on), title, description, severity, vulnerability type, loss type, and reference. SDLC artifacts include code, software architecture, software design, penetration tests, and the fielded system. A principle is a statement of general security wisdom derived from experience. Although principles exist at the philosophical level, they stem from practitioners' real-world experience in building secure systems. Principles are useful for both diagnosing architectural flaws in software and practicing good security engineering. A sample high level schema for a principle includes title, definition (with a description, examples, and references), related guidelines, and related rules.

III. Infrastructure of Security Model

A. Vulnerability Matrix

Vulnerability task was initiated to develop a searchable database containing , a taxonomy of vulnerabilities and exposures and to catalog them into libraries of properties that can be used in conjunction with the property based testing and model based verification instruments to assess the security of software code to assure that the software is free from the specified vulnerabilities and exposures

B. Security Assessment Tools

The security assesment tools are free tools that have been developed and collected for use in testing and assuring the security of operating systems and software. This collecting is provided as a list on the web sited. A more complete description of the tools and a discussion of how to use each of the tools is currently being developed

C. Property Based Testing

The role of property based testing is to bridge the gap between formal verification and ad hoc verification. This provides a basis for analyzing and as hoc verification. This provides a basis for analyzing software without sacrificing usefulness for rigor, yet capturing the essential ides of formal verification. It also allows a security model to guide the testing for security problems. Property based testing is a technique for testing that programs meet given specifications.

D. Model Based Security Specification

Model based specification make use of discrete finite models to verify compliance of the model 세 desired properties; in this case, software properties. Network security properties often focus on characteristics that are manifested though the operation of multiple software applications and systems operating concurrently with an attacking process.

IV. Summary of requirements

Software security is about making software behave correctly in the presence of a malicious attack, even though software failures usually happen spontaneously in the real world – that is, without intentional mischief. Not surprisingly, standard software testing literature is concerned only with what happens when software fails, regardless of intent. The difference between software safety and software security is therefore the presence of an intelligent adversary bent on breaking the system. Security is always relative to the information and services being protected, the skills and resources of adversaries, and the costs of potential assurance remedies; security is an exercise in risk management. Risk analysis, especially at the design level, can help us identify potential security problems and their impact.1 Once identified and ranked, software risks can then help guide software security testing. The basic schema introduced in the main text describes a way to organize and interrelate software security knowledge. Figure 1 shows the seven distinct knowledge catalogs that we divide into three knowledge categories. The category prescriptive knowledge includes three knowledge catalogs: principles, guidelines, and rules. These sets span a continuum of abstraction from high-level architectural principles at the philosophical level to very specific and tactical code-level rules. Guidelines fall somewhere in the middle of this continuum. As a whole, the prescriptive knowledge category offers advice for what to do and what to avoid when building secure software. The diagnostic knowledge category includes three knowledge

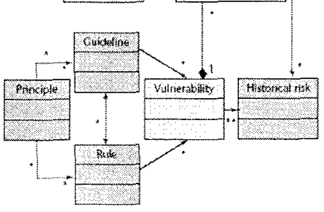

Fig. 1 Software security knowledge objects and a basic interrelating architecture.

V. Characteristics of Secure Software

Secure software refers to programs without security bugs or exploits vulnerable to abuse and malicious attack. These security exploits are defects inside programs that will lead to damages to users or the environment. Secure programs should be rigorously developed and analyzed. The characteristics of secure software include;

- Enforcement of confidentiality
- Enforcement of data integrity
- Minimized privilege
- Confinement or compartment
- Enforcement of mandatory access control
- Keeping code secure from unauthorized or malicious use

VI. Component Security Threats

Component based development offers great promise in reducing software production costs through software reuse. Capacity to acquire and absorb new features, new technologies and experimental knowledge make components an ideal medium for encapsulating intellectual assets.

6.1 Role of trust and reliability

Components can be software acquired from vendors and suppliers, open source software and, increasingly, freely downloadable software. In terms of security, this raise two important issues: trustworthiness of the supplier and trustworthiness of the product.

6.2 Overview of security models

Some knowledge of certain security models is essential for identifying the security features that are most relevant to component security. With respect to various security goals, confidentiality and integrity appear to be the most pertinent to component based design. These form the concerns of several widely known security models.

6.3 Modeling, analysis and testing

Though it goes against the definition of component, a detailed knowledge of component's internal structure undoubtedly helps "components analysis at a finer granularity, enabling a better control over component's behavior.

VII. Conclusion

Software vulnerability is second only to identity theft as the main security problem of the modern Internet. We propose an approach to reversing the trend that is inexpensive and consistent with existing and known successful programming practice.

References

[1] David Aucsmith, "Tamper Resistant Software: AnImplementation", Proceedings of the First International Workshop on Information Hiding, Pages: 317-33, 1996, LNCS 1174

[2] Toshio Ogiso ,Yusuke Sakabe, Masakazu Soshi,and Atsuko Miyaji, "Software Tamper Resistance Based on the Difficulty of Interprocedural Analysis", WISA 2002, Cheju Island, Korea, August 28-30, 2002

[3] G. Hoglund and G. McGraw, Exploiting Software: How to Break Code, Addison-Wesley, 2004.