Secure Software Education in Industry-oriented Software Talents Cultivation *

CHEN Dong-ming, ZHU Zhi-liang, GAO Xiao-xing
(Software College, Northeastern University, Shenyang 110004, China)

Abstract: The combination of attacks and defects means that today's security problems involving computer and software are frequent, widespread and serious. Thus, producers can suffer serious costs and consequences, so the need for a workforce more skilled in software security is clear. It is very important for us to teach students issues about secure software engineering. This paper provides some ideas from several aspects: security concepts, software security properties, secure software development activities, secure software project management and secure software sustainment which can be a guide for cultivating software talents.

Key words: secure software; emergent properties; software project management; software sustainment
CLC number: TP31 Document code: A Article ID: 0529-6579 (2007) S2-0112-04

1 Security Concepts

Security is a composite of the three attributes—confidentiality, integrity and availability. Security often requires the simultaneous existence of① availability for authorized actions only, ② confidentiality, and③ integrity with "improper meaning" "unauthorized". Security is not as simple as these sounds. Neither confidentiality nor integrity can be achieved unless identities can be firmly established— authenticated— and only allowable actions are permitted or possible— access control or in some situations encryption.

Security properties are emergent systems properties of the whole system. While necessary, security functionality does not make a secure system. To provide security for an item, individual pieces of security functionality such as password verifiers must be impossible to bypass to gain improper access. Preservation of properties that specify who is allowed to do what requires good identification of the actors. Authentication is the verification of an identity. Authentication may involve something one knows or possesses, a context or location, or an inherent characteristic. Authentication verifies an initial identification possibly done by the entity claiming an identity, by inference, or by recognition.

When attacks occur, the software may also be required to detect them and alert users, continue service, confine damage, rapidly recover, and aid in repair to prevent future problems. In addition to a software system's preservation of its required security properties within its digital domain, it can contribute to other systems, organizational or societal security goals including: ① Establishing the authenticity of users and data. ② Establishing accountability of users. ③ Providing usability, including transparency, to users to gain acceptance and resulting security. ④ Providing the abilities to: Deter and mislead attackers; Force attackers to overcome multiple layers of defense; Support investigations to identify and convict attackers.

Privacy needs are one of the key reasons for security. Privacy is a motivation for confidentiality, anonymity, and not retaining data. Avoiding falsehoods that could damage reputations requires data accuracy and integrity. Thus, software can help address security concerns at a number of levels. Figure 1 shows the relationships among a number of terms. Threatening entities, agents, or attackers may possess capabilities, resources, and intentions creating threats. To perform their attacks, attackers use their capabilities to exploit vulnerabilities in the system. These adversaries use specific kinds of attacks or "exploits" to take advantage of particular vulnerabilities in a system. Systems may employ countermeasures to reduce certain vulnerabilities.

2 Issues of Software Security

2.1 Software Security Properties

The goal of software security is the preservation of security properties, including confidentiality, integrity and availability (CIA) and accountability if their pres-

*收稿日期: 2007-10-28
作者简介: 陈东明 (1968年生), 男, 讲师; E-mail: chendm@mail.neu.edu.cn

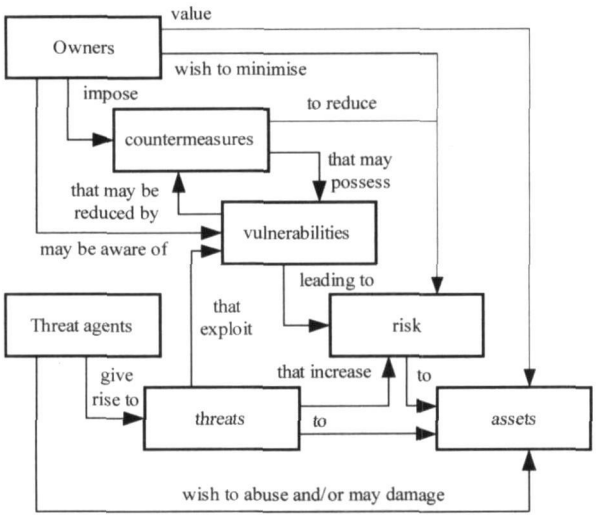

Fig. 1　Security concepts and relationships

ervation fails. Confidentiality, preventing unauthorized disclosure, and integrity, preventing unauthorized alteration, require a mechanism firmly establishing identities—authentication—and only allowing permitted actions—e.g., access control. Preserving availability includes preventing unauthorized destruction and ensuring adequate access or service. Other security properties include the ability to later reestablish all acts that occurred and their related actors—accountability—and causing relevant actors to be unable to effectively deny an act occurred—non-repudiation. Thus, software system security is a question of systems properties. Consequently, security is not just a question of the security mechanisms or functionality to aid in preserving them; the properties desired must be shown to hold wherever required throughout the system, e.g. the security functionality cannot be bypassed anywhere. In other words, security properties are emergent systems properties.

Security is an omnipresent issue throughout the software lifecycle. Likewise, potential security attacks or difficulties exist throughout the lifecycle of software systems and can result in a variety of security-related requirements for the software and its environment. Deciding upon the required security properties may be interwined with decisions on the extent of security-oriented development effort and functionality into an overall risk management decision.

Neither in the physical world nor in software can one absolutely guarantee security. Thus, when we speak of "secure software", the actual meaning is "highly secure software realizing—with justifiably high confidence but not guaranteeing absolutely—a substantial set of explicit security properties and functionality including all those required for its intended usage." One can also state this in a negative way as "justifiably high confidence that no software-based vulnerabilities exist that the system is not designed to tolerate". Notwithstanding these definitions' emphasis on high security, we should cover issues important not just for producing highly secure software but ones important to the many organizations under pressure to produce software merely more secure than the current version.

2.2　Secure Software Development Activities

Three things aid in reliably producing secure software: ① Outstanding software engineering performed by skilled people; ② A sound mastery of the relevant security expertise, practices, and technology; ③ The expert management required to ensure the resources, organization, motivation, and discipline for success. These implicitly imply an organizational culture that integrates concerns, activities, and rewards for secure software. Across these aspects, skilled people may be the highest leverage element. Achieving these skills, however, requires considerable knowledge beyond that already required for simply a good software engineering process.

Here we presume some people have knowledge of software engineering activities where security and safety are not concerns. For secure software development, however, a number of activities are new or significantly modified. Table 1 lists activities used by one or more secure software processes.

Tab. 1　Secure Software Engineering Activities

Adequately identify and characterize assets, and possible dangers or threats to them
Analyze dangers or threats, and defenses or mitigations
Develop security policies that meet user, organizational, and societal requirements, and formalize them
Develop a formal top-level specification
Show top-level specification agrees with security policy
Develop a security test plan
Develop an implementation of the system based on the top-level specification providing assurance case with argument and evidence
Perform penetration testing and test security functionality
Provide a covert channel analysis
Perform ongoing monitoring of security threats, needs, and environment
Provide assurance case including arguments and evidence showing software system meets security requirements
Perform changes securely maintaining conformance to—possibly revised—security requirements

In practice, these activities vary depending on

whether a heavyweight development process is used, a lightweight one, or one especially for the problems of legacy software. Generally the kinds of activities done in heavyweight processes are a modest superset of lightweight processes but with many activities performed with considerably more rigor. The activities used remedially for legacy software are usually similar to a subset of the lightweight ones. We must note that legacy software may eventually require more serious rework such as major redesign. Some activities on the table primarily are performed for systems with high security requirements. This table does not reflect the full impact of security on activities. In practice, concern for security affects performance of virtually every activity including, for example, consideration of security in all technical reviews. These activities and effects on other activities must be included in planning and tracking.

Furthermore, production needs to be performed in an appropriately secure work environment and a well-secured development system with the appropriate controls. Two aspects of producing secure software are the security of production and the security provided by the finished product. The former needs to be at least as good as the intentions for the latter. Thus, production and deployment need to be: ① Done in a secure environment operated securely, preferably providing counter-intelligence and forensic support; ② Possessed of an adequate set of trustworthy tools that have been received, installed, stored, updated, and executed securely; ③ Performed by trustworthy people; ④ Placed under appropriate controls against inside and outside mistakes, mishaps, and malicious actions.

In addition, adequate assurance must exist that these are true. One also needs to assess and improve security-related aspects, for example by: ① Collecting and analyzing security-related measurements; ② Improving security-related Processes, Personnel-related activities and skills, Artifact quality.

Merely a strong desire for suitable and correct products and services would be sufficient motivation for many of these activities. However, security needs provide even more motivations as well as new requirements for production and products. Because of this, some effective organizations establish processes across the organization that integrate security concerns and reuse them.

2.3 Secure Software Project Management

Secure Software Project Management is the "systematic, disciplined, and quantified" application of management activity to include the "planning, coordinating, measuring, monitoring, controlling, and reporting" that ensures the software being developed conforms to security policies and meets security requirements. Two issues cause differences in project management: High-assurance and Security. In running projects, managers have essentially four "variables" to control: ① Scope — the functions and facilities included in the system; ② Quality — the reliability, robustness, usability, etc., of the system; ③ Resources — the quantity and capability of the staff and computational tools used on the project; ④ Time — the schedule for the project.

For secure systems, where assurance and possibly certification or accreditation are essential goals, both scope and quality are constrained. The system will not be acceptable without certain non-bypassable security functionality, nor will it achieve certification or accreditation unless it is of sufficient quality. This reduces the options available to project managers. It is becoming widely accepted that incremental or evolutionary development is an appropriate way to address the problems of complex IT systems. The above analysis of options available to project managers suggests two problems with these approaches: ① The first releasable increment has to include all core security functions, hence it may be a "large" increment; ② Quality cannot be compromised, and the design must be such that the quality of the "first increment" cannot be undermined by later increments.

Thus managers need to define, as early as possible, the architecture for the system, identifying the key security properties and mechanisms and ensuring that they are designed and developed with an architecture such that they cannot be compromised by adding later increments. The project needs to be managed viewing the architecture and the quality of the software as non-negotiable plus gaining senior management acceptance that timescales and resources, hence costs, have to be variable (may need to increase) so as not to compromise security.

The competence of personnel is very important. Competencies include: Technical skills, especially knowledge of security; Knowledge of techniques for achieving low defect density; Domain knowledge, i.e., understanding of the application area; Communication skills; Personal attributes, such as integrity. There has been an attempt in the UK to codify such skills for safety of programmable electronic systems (IEE, 1999), but so far as we are aware no systematic approach exists in the area of security. There are few management techniques focused solely on low defect, high assurance software, although there has been some positive experience of using approaches such as the SEI's (Software Engineering Institute) Team Software

Process on projects in the UK. Perhaps the most crucial management technique is in process measurement of errors and defects, analysis of the root cause of such defects, and using this information to help manage the project and to improve processes.

If one has successfully managed a high-assurance project, say for safety, then a number of security issues remain, but much of the general project management approach can be the same. From a project management perspective, there is not anything different/additional with respect to any other high integrity development. When we managed the complex of large project, we didn't do anything differently because it was a security project-- simply applied the usual praxis planning and delivery approach (plus lessons learned from my previous projects, of course). The fact that it was a security project obviously affected the risk assessment, technical approach, quality planning, etc., but the project management approach was the same."

2.4 Secure Software Sustainment

Sustainment covers software assurance activities when software is put into an operational environment and the unique software assurance activities after initial deployment. It involves processes that continue to assure that software satisfies its intended purpose after initial deployment and during operations. Software is written to facilitate organizational goals and each goal has security requirements. If the software does not meet those requirements, the organization must realign the functioning of the software, or alter the goal to bring it into alignment. The security environment is continuously changing. Environmental factors possibly impacting security include: ① People connecting to the system's endpoints and the motivations of those people; ② Systems interconnected with the software; ③ The type of data flowing to, through, or from the software or system; ④ The way the organization does business, or the type of business that is conducted; ⑤ The rigor or extent of the security objectives; ⑥ The organizational risk model/risk tolerance.

Software must always satisfy security objectives. If that is not the case, then the organizations risk mitigation process must dictate the steps necessary to change the software, or change the objectives. See Figure 2.

Certain aspects of sustainment are also related to acquisition of secure software. The following is an outline of the contents: Background, Operational Assurance (Sensing), Analysis, Response Management (Responding), Infrastructure Assurance.

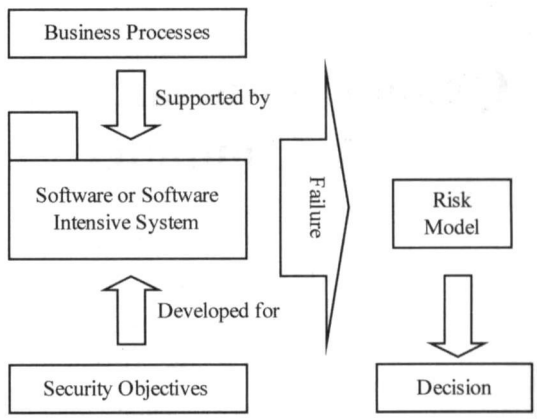

Fig. 2　The Environment of Software or Software Intensive Systems

3　Conclusion

Software is in danger all its life from malicious and non-malicious behaviors. We need to recognize and analyze these dangers and combine of all risks. At the same time, we must emphasize the development of secure software and put security into the practice of software engineering, especially to students who major in software engineering. Only by doing this can we get more secure software and promote the quality of software.

References:

[1] Clifford Berg, J. High-Assurance Design: Architecting Secure and Reliable Enterprise Applications[M]. Addison Wesley, 2005.

[2] McGraw Gary. Software Security: Building Security In[M]. Addison Wesley, 2006.

[3] Samuel Redwine, T. Secure Software Guide to the Common Body of Knowledge[M]. SEI, 2006.

[4] Viega John, McGraw Gary. Building Secure Software: How to Avoid Security Problems the Right Way[M]. Addison Wesley, 2003.

[5] Sommerville Ian. Software Engineering[M]. 8th Edition. China Machine Press, 2006.

[6] Maciaszek Leszek A, Liong Bruc Lee. Practical Software Engineering[M]. China Machine Press, 2006.

[7] Pressman Roger S. Software Engineering: A Practitioner's Approach[M]. 5th Edition. China Machine Press, 2003.

[8] Schach Stephen R. Object-Oriented and Classical Software Engineering[M]. Beijing: China Machine Press, 2003.

[9] Christensen Mark J, Thayer Richard H. The Project Manager's Guide to Software Engineering's Best Practices[M]. Publishing House of Electronics Industry, 2004.