Chapter VI
Novel Methods of Incorporating Security Requirements Engineering into Software Engineering Courses and Curricula

Nancy R. Mead
Software Engineering Institute, USA

Dan Shoemaker
University of Detroit Mercy, USA

ABSTRACT

This chapter describes methods of incorporating security requirements engineering into software engineering courses and curricula. The chapter discusses the importance of security requirements engineering and the relationship of security knowledge to general computing knowledge by comparing a security body of knowledge to standard computing curricula. Then security requirements is related to standard computing curricula and educational initiatives in security requirements engineering are described, with their results. An expanded discussion of the SQUARE method in security requirements engineering case studies is included, as well as future plans in the area. Future plans include the development and teaching of academic course materials in security requirements engineering, which will then be made available to educators. The authors hope that more educators will be motivated to teach security requirements engineering in their software engineering courses and to incorporate it in their curricula.

INTRODUCTION

Exploitable defects in software pose a threat to both our national security and our way of life. That is because our critical infrastructure is enabled by information technology (PITAC, 2005). Nevertheless, even though software plays a pivotal role in ensuring every sector of our economy, the President's Information Technology Advisory Council (PITAC) found that "commonly used software engineering practices permit dangerous defects, which let attackers compromise millions of computers every year" (PITAC, 2005, p. 39).

Most defects are the result of programming or design errors (Jones, 2005). And such defects do not have to be identified or actively exploited in order to be a threat (Redwine, 2006). Yet, given that unfortunate fact, PITAC still found that "current commercial software engineering lacks the rigorous controls needed to [ensure defect free] products at acceptable cost" (PITAC, 2005, p. 39). And even worse, "In the future, the nation may face even more challenging problems as adversaries—both foreign and domestic—become increasingly sophisticated in their ability to insert malicious code into critical software" (Redwine, 2006, p. xiv).

In fiscal terms, the exploitation of defects costs the U.S. economy an average of $60 billion dollars per year (Newman, 2002). However, the real concern lies in the fact that the exploitation of a flaw in the software that underlies basic infrastructure services like power and communication could cause a significant national disaster. The Critical Infrastructure Taskforce sums up that likelihood in a single statement: "The nation's economy is increasingly dependent on cyberspace. This has introduced unknown interdependencies and single points of failure. A digital disaster strikes some enterprise every day, [and] infrastructure disruptions have cascading impacts, multiplying their cyber and physical effects" (Clark, 2002, p. 6).

The generally acknowledged solution to the problem of exploitable defects is more secure practice in every aspect of the acquisition, development, and sustainment of software and software artifacts. Nonetheless, "informed consumers have growing concerns about the scarcity of practitioners with requisite competencies to build secure software" (Redwine, 2006, p. xiii).

Because of the key importance of capable practitioners and the general lack of proper preparation, The National Strategy to Secure Cyberspace – Action/ Recommendation 2-14 has mandated the Department of Homeland Security (DHS) to "promulgate best practices and methodologies that promote integrity, security, and reliability in software code development, including processes and procedures that diminish the possibilities of erroneous code, malicious code, or trap doors that could be introduced during development" (NIAC, 2003, p. 35).

It would seem to be a simple task to "identify the necessary workforce competencies, leverage sound practices, and guide curriculum development for education and training relevant to software assurance" (Redwine, 2006, p. xiv.). However, the problem is that security is not a mature field, and so the teaching of security topics is done in a number of disjointed places within higher education. That includes "software engineering, systems engineering, information systems security engineering, safety, security, testing, information assurance, and project management" (Redwine, 2006, p. xiv).

Coherent knowledge about "software assurance processes and practices has yet to be integrated into the body of knowledge of the contributing disciplines" (Redwine, 2006, p. xiv). Too often, the result of this lack of integration is the graduation of a software engineering student who develops buggy code with weak security measures.

It is both impractical and impossible to simply drop the whole body of software assurance knowledge into a traditional computer curriculum. Therefore it is necessary to adopt a focused strategy and a clear starting point. One of the logical places to start the integration process is in an area that is vital to good security practice, but which is also well established and important to general development. That is security requirements engineering.