A Portal for Software Security

Nancy R. Mead
Software Engineering Institute

Gary McGraw
Cigital

One of the real challenges facing the emerging field of software security is the lack of an easily accessible common body of knowledge. Simply put, most software developers and architects—the very people who need to understand and practice software security—remain blithely unaware of their critical role. Without their direct participation, software security will languish. In this installment of Building Security In, we describe a software security portal that the US Department of Homeland Security (DHS) National Cyber Security Division (NCSD) is developing (along with the Carnegie Mellon Software Engineering Institute [SEI] and Cigital). The launch of this portal is scheduled for October 2005 as part of the US-CERT Web site. The portal aims to provide a common, accessible, well-organized set of information for practitioners wishing to do software security.

Software security: Strong philosophy, weak practice

Since 1999, several seminal books have helped define the software security field.[1–3] These books introduced the approach to building security in, which practitioners have since enhanced, expanded, and published in various technical articles, including the Building Security In series (see the sidebar).

The core philosophy underlying this approach is that security, like dependability and reliability, can't be added onto a system after the fact through the addition of sets of features, nor can it be tested into a system. Instead, security must be designed and built into a system from the ground up. More than 90 percent of reported security incidents are the result of exploits against defects in the design or code of software, according to the CERT Coordination Center (CERT/CC) of the SEI. Although traditional security efforts attempt to retroactively bolt on devices that make it more difficult for those defects to be exploited, such devices simply aren't effective.

Standard-issue software development lifecycle models—ranging from the process-heavy Capabilities Maturity Model (CMM) to the lightweight Extreme Programming (XP) approach—are not focused on creating secure systems. They all exhibit serious shortcomings when the goal is to develop systems with a high degree of assurance.[4] If they even address security issues at all, they're most often relegated to a separate thread of project activity focused on security features; as a result, security is treated as an add-on property. For example, although using applied cryptography (such as SSL) to protect message traffic is a useful security feature, the union of all such security features doesn't ensure secure software.

Any isolation of security considerations from primary system-development tasks results in an unfortunate and untenable separation of concerns. Security should be integrated and treated on a par with other system properties. The only way to develop systems with required functionality and performance that can also withstand malicious attacks is to design and implement them to be secure. Software security is thus a full lifecycle undertaking in which critical design decisions and trade-offs must be clearly and thoroughly understood. In addition, tools for supporting security engineering (for example, source code analysis tools) must be integrated into the software development environment. By treating software security risk explicitly throughout the software life cycle, we can properly identify and mitigate the consequences of security failure and successful security attack.

For each lifecycle activity, a team made up of security analysts and developers must address security goals and incorporate best practices to assure security. In some situations, existing development methods can be used to enhance security. Current research is also creating new methods that developers and analysts can apply as they build software; however, more research and experimentation are required before the goal of security can become a reality.[5] One way of illustrating a lifecycle approach that incorporates security into each basic phase of software develop-

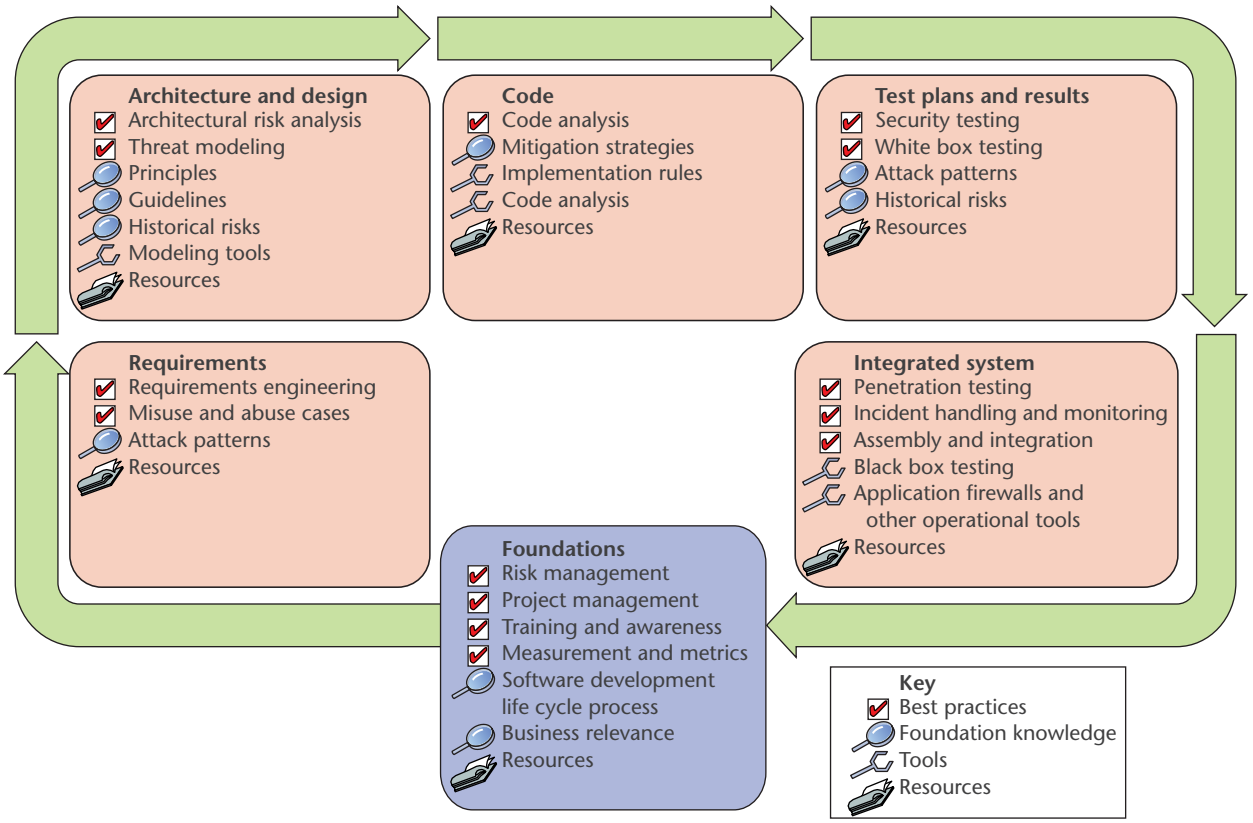

Figure 1. The organizing concept for the Build Security In (BSI) portal. The alignment of this view shows not only best practices (as Figure A in the sidebar does), but also knowledge and tools.

ment has been intentionally created to be process agnostic. That is, the best practices and methods described are applicable to any and all development approaches as long as they result in the creation of software artifacts. Given this approach, software development processes as diverse as the waterfall model, Rational Unified Process (RUP), XP, Agile, spiral development, and CMM involve creating a common set of software artifacts (the most common artifact being code). In this way, we can apply software security best practices and their associated knowledge catalogs regardless of exactly which "base" software process is followed. Figure 1 includes best practices (as does Figure A in the sidebar), knowledge, and tools, all organized according to software artifacts.

The BSI Software Assurance Initiative

The Build Security In (BSI) Software Assurance Initiative seeks to alter the way that software is developed so that it's less vulnerable to attack by building security in from the start. BSI is a project of the Strategic Initiatives Branch of the DHS's NCSD, which has sponsored the development and collection of software assurance and software security information that will help software developers and architects create secure systems. The effort is managed by Joe Jarzombek, the DHS director for software assurance.

As part of the initiative, a BSI content catalog will be made available as a Web portal in October. This portal is intended for software developers and software development organizations that want information and practical guidance on how to produce secure and reliable software. The catalog is based on the principle that software security is fundamentally a software engineering problem that we must address systematically throughout the software development life cycle. The catalog will contain links to a broad range of information about best practices, tools, and knowledge.

BSI content catalog

Figure 2 identifies aspects of software assurance covered in the catalog. Material will be divided into three major categories: processes, tools, and knowledge. This is an alternative way of organizing software security content without reference to artifacts.

The categorization is the result

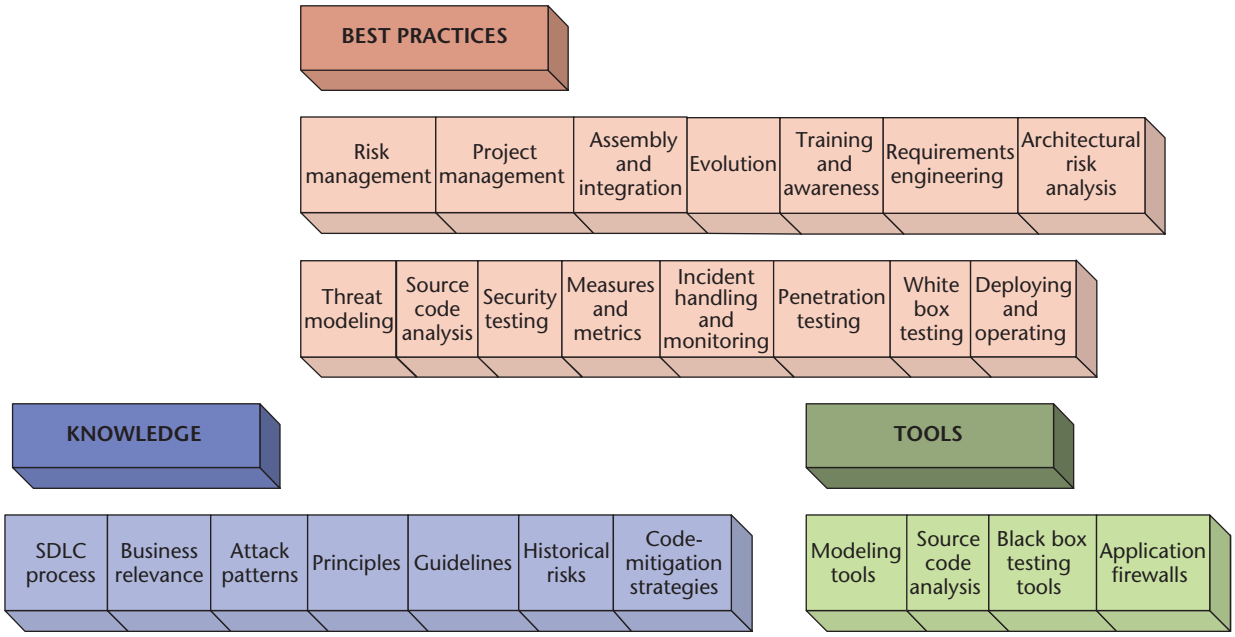

Figure 2. Components of the BSI content catalog. The three main content areas are best practices, knowledge, and tools, which are organized by software artifacts, as shown in Figure 1.

of merging an earlier collaboration framework with ideas presented in the lifecycle touchpoints diagram that accompanied each Building Security In article (see Figure A in the sidebar). The US National Cyber Security Taskforce's report also identified additional practices on processes to produce secure software (www.cyberpartnership.org/initsoft.html). Soon, the BSI portal will supplement the taskforce's practices with process models and references to appropriate tools, measurement, and other resources.

Although the team creating the portal won't achieve complete content coverage immediately, the DHS plans to launch the portal with some content in most of the areas shown in Figure 2. The BSI team will use feedback received on this initial content (as well as input from industry) to prioritize further work on the catalog.

Software assurance aspects

The portal will include several types of information, categorized for efficient search and utility.

Best practices. A significant portion of the BSI effort is devoted to best practices that can provide the biggest return considering the current best thinking, available technology, and industry practice. As more resources become available, more practices are proven, changes occur in the industry environment, and technology progresses, this list will grow. *IEEE Security & Privacy*'s Building Security In series has covered several of these best practices in some detail (keys BSI2, BSI3, BSI4, BSI5, and BSI6 in the sidebar).

Knowledge. Software defects with security ramifications—including implementation bugs such as buffer overflows and design flaws, including inconsistent error handling—promise to be with us for years. Recurring patterns of software defects leading to vulnerabilities have been identified by longtime software security practitioners, and the BSI team is documenting detailed instructions on how to produce software without these defects. This work shows up in Figure 2 as "guidelines" and "code-mitigation strategies."

The BSI team has also identified principles that provide high-level direction for avoiding security problems in design, such as the principle of least privilege and the principle of compartmentalization. We'll further describe and enhance them as the portal evolves. (For more on foundational knowledge for software security, see key BSI7 in the sidebar.) The BSI team is collaborating with the US National Institute of Standards and Technology (NIST), the International Organization for Standardization (ISO), and the IEEE on standards activities focused on developing safe and secure subsets of languages and software assurance style guides.

Tools. The BSI portal includes information about which tools developers and security analysts can use to detect and/or remove common vul-

Software security best practices BSI series

Over the past two years, the Building Security In series in this magazine has introduced and discussed several software security best practices that developers and security engineers can apply throughout the software development life cycle (see Figure A). The articles in this series, listed in the table below, provide an outline for a common body of software security knowledge.

The Build Security In (or BSI) portal will be a living repository for software security knowledge, best practices, and content based directly on the ideas introduced in this series. The portal will be officially launched in October.

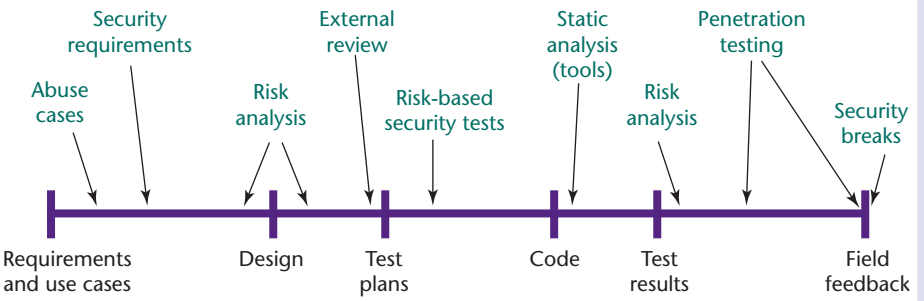

Figure A. Software security touchpoints. We can apply several software security best practices (the arrows) throughout the software development life cycle, given a set of common software artifacts (shown along the bottom).

Table 1. Building Security In articles.

TITLE	KEY	AUTHOR	*IEEE SECURITY & PRIVACY* CITATION	URL: WWW.CIGITAL.COM PAPERS/DOWNLOAD/
Software Security	BSI1	Gary McGraw	2(2):80–83	bsi1-swsec.pdf
Misuse and Abuse Cases: Getting Past the Positive	BSI2	Paco Hope, Gary McGraw, Annie Anton	2(3):32–34	bsi2-misuse.pdf
Risk Analysis in Software Design	BSI3	Denis Verdon, Gary McGraw	2(4):79–84	bsi3-risk.pdf
Software Security Testing	BSI4	Bruce Potter, Gary McGraw	2(5):81–85	bsi4-testing.pdf
Static Analysis for Security	BSI5	Brian Chess, Gary McGraw	2(6):76–79	bsi5-static.pdf
Software Penetration Testing	BSI6	Brad Arkin, Scott Stender, Gary McGraw	3(1):84–87	bsi6-pentest.pdf
Knowledge for Software Security	BSI7	Sean Barnum, Gary McGraw	3(2):74–78	bsi7-knowledge.pdf
Adopting a Software Security Improvement Program	BSI8	Dan Taylor, Gary McGraw	3(3):88–91	bsi8-program.pdf

nerabilities. Of particular interest are static analysis tools that help developers look for common security-critical problems in source code. The best current commercial tools support languages such as Java, CLR, C++, C, and PHP (see key BSI5 in the sidebar).

Business Case. Even with deep technical content, a business case is required to convince industry to adopt secure software development best practices and educate consumers about the need for software assurance. Therefore, each documented best practice addresses the business case for use of that practice. In addition, the portal will include an overall business case framework.

Dynamic navigation. The extent to which users will find the content accessible as well as useful will determine how this portal impacts real-world development practices and, thus, overall systems security. The BSI team is trying to make the content approachable in several different ways. For example, a software engineer might use the catalog to determine applicable security guidelines; an architect might use security principles to determine how to design an *n*-tier application in a secure fashion; and a development team leader might use the information to justify software assurance techniques to management by building a business case. Because the repository will be structured and designed to evolve

as well as support usage by a variety of user types, it will include a dynamic navigation interface.

Once practical guidance and reference materials are available for the day-to-day work most development organizations do, the BSI team plans to identify and organize content for practical guidance and reference materials for enterprise-level security concerns.

To help ensure that this software assurance initiative is accepted and supported by the community of software development organizations, the team is seeking involvement from representatives from industry, academia, and government. Toward this goal, working groups to guide the creation of the BSI software assurance portal have been formed. The Software Technical Working Group (STWG) is composed of respected individuals in the technical community whose primary function is to review the portal content's technical veracity and identify future content.

Although the portal is currently in a nascent stage, the BSI team welcomes feedback; prior to the site's launch, you can send it to Jan Philpot at the SEI (philpot@sei.cmu.edu). Community involvement and use is crucial to the portal's success, and we look forward to help from the community in improving software security worldwide. ▢

References

1. R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, 2001.
2. J. Viega and G. McGraw, *Building Secure Software*, Addison-Wesley, 2001; www.buildingsecuresoftware.com.
3. M. Howard and D. LeBlanc, *Writing Secure Code*, Microsoft Press, 2001.
4. A.B. Marmor–Squires and P.A. Rougeau, "Issues in Process Models and Integrated Environments for Trusted Systems Development," *Proc. 11th Nat'l Computer Security Conf.*, US Govt. Printing Office, 1998, pp. 109–113.
5. N.R. Mead et al., "Managing Software Development for Survivable Systems," *Annals Software Eng.*, vol. 11, no. 1, 2001, pp. 45–78.

Nancy Mead *is a senior member of the technical staff at the Software Engineering Institute (SEI). She is also a faculty member at Carnegie Mellon University. Her research interests are in the areas of software requirements engineering, software architectures, and software metrics. Mead has a PhD in mathematics from the Polytechnic Institute of New York, and a BA and an MS in mathematics from New York University. She is a senior member of the IEEE. Contact her at nrm@sei.cmu.edu.*

Gary McGraw *is chief technology officer of Cigital. His real-world experience is grounded in years of consulting with major corporations and software producers. McGraw is the coauthor of* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001),* Java Security *(John Wiley & Sons, 1996), and four other books. He has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.*