

Essential Factors for Successful Software Security Awareness Training

As organizations flesh out their enterprise software security framework (ESSF), they quickly spot an overwhelming gap between their current state of practice and their eventual goals for building secure applications. To address this gap, the application security

when secure coding issues arise.

Ultimately, development managers have the important decision-making role when it comes to security: they constantly make trade-offs between schedule, cost, quality, and functionality. Security issues might pop up on an organization's radar, but depending on the maturity of its security assurance activities (such as penetration testing or code review), management likely handles these "blips" in vexing fire drills. Here, the development manager defers to the lead developer. In an IT shop, the development manager often has to weigh cost and schedule under the additional pressure of the business owner and almost always has to buckle, leading to insecure software and a bevy of "exception forms" documenting the decision. Most organizations just don't have the framework for making objective quantitative trade-offs involving security.

Finally, we come to the security resources themselves. Organizations spread individuals too thin, preventing them from bolstering their knowledge with the latest vulnerability data or becoming familiar with a new open source package development uses. In the worst-case scenario, security folk don't know what to look for and have to rely on a security tool or an Internet "Top 10" vulnerability list instead.

The role of training

Even at an awareness level, training curriculum must not only pave the

KENNETH R. VAN WYK
KRVW
Associates

JOHN STEVEN
Cigital

group must bring ESSF plans to other parts of the organization—corporate, engineering, business, training, and IT groups, to name a few. Such socialization helps organizational participants understand their role in framework adoption and roll out, and it should cover what tools people will need, how they'll interact with each other, and what levels of effort they can expect to put forth.

Creating a training curriculum is one means of socialization. It addresses a broad spectrum of personnel and provides enough organization-specific details to be useful to software engineering practitioners. Socialization through training begins with defining a well-rounded ESSF endorsed by the organization's most senior management. The trainers who deliver this material must possess organizational savvy and speak credibly about new expectations and activities. They must also be prepared to answer questions about implementing the ESSF and reiterate senior management's buy-in to each audience member's satisfaction—regardless of which part of the organization an employee represents. Training extends beyond "software

security is coming, and here's what it will mean to you." The application security group must close any knowledge gaps that key organizational performers suffer from before ESSF roll-out plans stall.

State of affairs

Developers fulfill the single most important role in software security: they build the software. Yet, it's very likely that all but a handful of an organization's developers are completely blind to how the software they're building or maintaining today could be exploited tomorrow. A predictable consequence is that developers can't build security into their applications and often push back heavily on any security analyst who takes a pot shot at their code.

University IT degrees and even computer science programs don't equip graduates with the ability to program securely. Even experienced developers often don't possess essential software security knowledge—in fact, some become quite entrenched in a hard-to-break "deliver features" mentality. The trend toward outsourced development only adds to the finger-pointing

way for other software security efforts laid out in the ESSF but must truly change the way an organization's developers do their work. Therefore, awareness training must be customized to the structure, activities, and other salient aspects of the organization's ESSF. Training based on a generic secure development life cycle isn't satisfactory; although a life cycle outlines excellent best practices, it rarely maps to an organization's software development idiom right out of the box.

Moreover, awareness training must be customized to reflect the organization's platforms, technology paradigms, languages, and packages. An organization might have lines of business that evolve in profoundly different environments. Without customized training, developers will complain that the guidance is "too high-level."

Developers often find software security principles difficult to execute at first, even if they're well-presented. Training structured as a lecture forces them to try out software security techniques for the first time "in the wild," amid the pressure of schedules and feature delivery. Organizations must therefore commit enough time to awareness training to give examples and case studies that engage their audiences and help them apply and retain the information being presented.

Customizing training

After the ESSF is fleshed out and awareness training has introduced the organization to the basics, it's time to delve into specific activities. Because an ESSF is likely to vary significantly among development organizations, training can't be a one-size-fits-all sort of effort. Instead, the training at this point must be highly ESSF specific, similar to training for a particular tool.

The first step should be to design a curriculum that covers the personnel roles across the development organization as well as varying levels

of detail for the different tiers of software engineers. If the organization works with an outsourced training provider,

- The vendor should be highly familiar with the ESSF and the software development organization's needs.
- The vendor should be able to write custom courses that represent the detailed steps the ESSF describes.
- The courses should use the technologies deployed within the development organization.
- The vendor should have access to the company's own code base, "lessons learned" from past failures, and other (highly sensitive) organizational information to incorporate the most relevant examples and exercises possible within each class.

With these recommendations in mind, let's take a closer look at what might go into the curriculum itself.

Job roles

The curriculum should be useful and informative for each job role within the organization. This means that each role should receive basic awareness training across the ESSF as well as highly specialized training for the security activities in the ESSF that fall within their respective fields. Software designers and architects, for example, should spend a significant amount of training time learn-

- *Executive* training should teach the difference between software security and conventional network security approaches, particularly the pervasiveness of software security issues. It should also present and align business owners with the ESSF. Finally, it should establish short- and medium-term goals for software security capability, as well as set group owner expectations, means of measurement, risk management goals, and MBOs.
- *Management*-level awareness training should teach development managers how to make schedule, cost, functionality, and risk trade-offs via a risk-management framework. It should also demonstrate how software security touchpoints affect development methodology. Finally, it should equip managers with knowledge collateral and questionnaires to validate the probability and impact of security findings and mitigation strategies.
- Awareness training for *development and security* staff should seek to transform the way developers behave when they return to their development environments. It should further show how to execute software security touchpoints to build security in to the development life cycle. It's also vital that it clearly shows how attackers exploit software and how to resist attack. Finally, it should provide analysts with a toolkit of specific attacks and principles with which to interrogate systems.

Organizations must commit enough time to awareness training to give examples and case studies that engage their audiences and help them apply and retain the information.

ing how to secure the architectures used across the organization.

To this end, organizations need at least three flavors of awareness training:

These tiers should really just serve as a starting point. Organizations might well find it worthwhile to take them one step further—for

Prescriptive guidance

Prescriptive guidance shows design and code samples for authenticating Web requests that comply with the corporate standard (perhaps by using a Kerberos plug-in, for example). Furthermore, it mandates declarative authorization constraints, walking developers through how to structure their application's ActionController classes and associate URL-pattern and authorization-constraint tags in the deployment descriptor.

So how specific and detailed is "enough"? Jokingly, we suggest that there should be at least as much Courier font as there is Times New Roman—referring to the fact that authors often use Courier as a convention for showing code's syntax.

example, by providing a separate awareness training program for software architects and quality assurance testers.

Tiers

Without a doubt, a one-size-fits-all approach won't work. At a minimum, the training should have beginner, intermediate, and advanced classes for different needs.

Let's look at code-review training, for example: beginners can learn an approach, but they'll likely need a checklist or a code analysis tool's output to be effective. Intermediate students might use analysis tools, but their training should focus more on code navigation and understanding so that they can find more subtle bugs manually. Rather than step-by-step tool and checklist walkthroughs, this training could entail more case studies. Advanced training should introduce the most complex techniques—such as analyzing underlying technology frameworks for use in code reviews. This helps reviewers find hidden problems when solely considering source code. Teaching even intermediate students this type of technique is a disservice, though, because they won't be able to effectively apply it and could waste a tremendous amount of time trying.

Pitfalls to avoid

Security professionals often insist that developers think like attackers to write better code. Examples from the physical world, such as a bank or a castle, are demonstrative, but they don't make software security real. Indeed, "making it real" means presenting very concrete information about attacks and protective mechanisms in the language developers understand: design and code. Only when training gives prescriptive design and coding guidance of what to do to resist attack does it stand a chance of sticking in a developer's mind. Ultimately, guidance, examples, and case studies must be deeply steeped in the platforms, technology paradigms, languages, and packages that developers currently use (see the "Prescriptive guidance" sidebar for more information).

In following this advice, training content providers sometimes jump head-first into a deeply technical rabbit hole, and two common flaws can emerge—the *bug parade* or the *how-to cookbook*—and both should be avoided as sole solutions for training. The bug approach attempts to teach by merely describing software security defects, whereas the cookbook approach attempts to teach by citing example after example (with source code, quite often) of secure software. Although both are worthy topics, they should be woven into training in a way that uses them as examples of the core material being presented—such core material should stress the "how" and not (just) the "what."

Examples and exercise

It's largely accepted that one of the least effective ways for adults to learn new topics is to sit in a lecture-only training class for several days. An effective curriculum should include numerous case studies as examples as well as exercises in which students can realistically put to practice the

information provided in the lecture content. Moreover, software security training must present concepts as examples and immediately follow those examples with exercises or, even better, an interactive case study revisited and extended throughout the day.

Through extensive experience in delivering training to software developers, one of this article's authors (Van Wyk) has found it valuable to break audiences up into several groups and facilitate each group's exercise individually. Having each group share findings with the entire class lets the trainer explain findings, show emerging patterns, and tie specifics back to principles for reinforcement. Because practicing security means thinking of all the esoteric possibilities, there's an additional value in showing each individual group aspects of the problem they missed but that other groups caught.

In this context, we're discussing a goal curriculum, but it might be beneficial to consider implementing a target curriculum one step at a time. The junior, intermediate, and advanced training tiers, for example, could be implemented a couple of years after the initiative has started, based on the needs of the organization's developers or within the available budget's bounds.

Rolling out a training curriculum

Even in an ideal environment with an enormous training budget, organizations should take care when rolling out courses to employees. Each course would benefit from iterative improvement and refinement over time. Also, as each course is developed, it's worth conducting a test class or two to ensure that the content, exercises, and so forth are appropriate and that everything works smoothly. Let's look more closely at some concrete ideas for what your organization could do.

Pilots

Run one or two pilot classes to a representative audience, explaining to them in advance that this is a test run and that you're soliciting feedback. Consider inviting a few students from the target audience. Look in particular for people who are good at mentoring junior staff.

Instructors

Finding good training instructors is rarely easy. Perhaps this is because the best trainers are both highly technical and excellent communicators—character traits that are often at odds. Software security trainers must be sufficiently knowledgeable about software that they can talk to their audiences as peers, but they also need a deep understanding of software defects and how they're exploited. Professional IT security trainers rarely have the necessary background to do this effectively. Consider recruiting trainers from within the development group, looking particularly for people who can communicate effectively.

Exercises

As we already discussed, hands-on exercises are an excellent tool for training developers. To the extent feasible, they should include actual computer-based exercises in which students can run tools, exercise software defects, and so on. Naturally, some of these things must be done in a carefully controlled environment to ensure that they don't affect the rest of the organization. Consider virtualized training environments that can be run on small, closed networks and quickly restored to secure baseline configurations for each class.

Iterative refinement

Each class should be kept up to date with existing technologies as well as with the ESSF as it evolves. Iterative improvement helps ensure that classes meet developer needs.

Technologies

Consider using computer-based training (CBT) tools for at least some of the classes. They can be effective for certain aspects of the training and offer some enticing advantages to the traditional teacher-classroom environment. On one hand, they let developers participate at their own pace and time; on the other, they limit class interaction and exercise feasibility, so proceed with care. CBT could be just the ticket for highly technical topics such as how (or how not) to write secure code in specific languages. Just as with a technical book, they can serve as reference material for developers, especially if they have good search capabilities.

Designing and implementing a software security training curriculum that meets the needs of the enterprise is an essential element in rolling out an ESSF. Although some organizations start off with a basic awareness training program, it isn't enough. An effective training program must be about getting software developers to change their habits. To accomplish this, a broader training curriculum is needed that addresses the different roles and levels of experience within a software development organization. It must also guide them through the various minefields they'll face when trying to accomplish the ESSF's goals. □

Kenneth R. van Wyk is a principal consultant at KRvW Associates. His interests include software security and incident-handling. Van Wyk has a BS in mechanical engineering from Lehigh University. Contact him at ken@krvw.com.

John Steven is a technical director and software security principal at Cigital. His interests include J2EE security, and he works in partnership with companies large and small to help them build their own software security capabilities internally. Steven has an MS in computer science and a BS in computer engineering from Case Western University. Contact him at jsteven@cigital.com.

Here now from the IEEE Computer Society

IEEE ReadyNotes

Looking for accessible tutorials on software development, project management, and emerging technologies? Then have a look at ReadyNotes, another new product from the IEEE Computer Society. These guidebooks serve as quick-start references for busy computing professionals. Available as immediately downloadable PDFs (with a credit card purchase), ReadyNotes are here now at <http://computer.org/readynotes>.



IEEE
computer society
60TH anniversary