Security Compliance as an Engineering Discipline

As a result of new initiatives and requirements like the Payment Card Industry Data Security Standard (PCI-DSS), many organizations are building comprehensive application security programs for the first time. To do so, a number of those concerns are looking to the proven success of the Microsoft Security Development Lifecycle (SDL). This can be a very smart business move, but it's important to understand how the engineering focus of the SDL makes it different from the typical security-compliance effort. This month I'm going to address some of the ways to harmonize compliance-focused programs with security engineering to improve your software development practices.

By integrating secure engineering practices into the entire software lifecycle, the SDL lets you achieve application security assurance using a holistic approach. The process requires direct engagement with software developers and testers, and unlike the typical compliance program, it's incremental, iterative and should be customized to every organization.

In this article, I'll focus on some of the strategies and best practices for deploying SDL and integrating it with security compliance regimes.

Compliance and the SDL

Building a software security program from scratch can seem daunting. Few organizations have a program or team dedicated specifically to secure application development. The portfolio of the typical security group includes networks and firewalls, policy and identity management, systems configuration, and general operations and monitoring. Securing the ever-growing set of custom-built, mission-critical applications is a new and challenging addition to these responsibilities.

While a mature software security practice will facilitate compliance efforts like PCI and the Common Criteria for Information Technology Security Evaluation, looking at the SDL as just another compliance effort will keep you from getting the best results from your program. Unlike compliance programs, which have externally defined goals and activities, the SDL recognizes that the business goals and abilities of attackers are constantly changing. Many organizations achieve compliance through large up-front efforts followed by minimal maintenance. In contrast, the SDL starts incrementally and focuses on using constant effort for continuous improvement. In addition, compliance often encourages separation of duties between implementation and enforcement. The SDL is built on close coordination with developers and testers. Given these differences, if your experience is only in managing compliance efforts, how should you approach an SDL implementation?

First, consider how your organization has managed other major software industry changes in the past decade, such as moving from procedural to object-oriented development, client-server to Web-based software, or waterfall to agile development methods. These changes weren't made across the entire organization with the stroke of a pen. They were introduced incrementally and grew organically. A similar approach, as outlined below, will work best for the SDL.

- Start small with one or two pilot projects.
- Even though you'll introduce the new way of doing things with a capable team, bring in outside help where needed.
- Study and learn from the results at the completion of a project. Adapt the standard practices to fit your particular organization. Get rid of the things that didn't work.
- Let coverage expand organically, nurturing internal experts and spreading the culture change at a pace the organization can manage without disrupting the business.

Second, recognize that being in compliance is just one goal for your program—the first step, not the finish line. More-secure and higher-quality software provides its own business and competitive value. Measure your program by how successful you ultimately are at delivering more trustworthy products and what that means for your bottom line.

Adding Value with Secure Engineering

Often a team needs help optimizing its software security program because the project is in the early stages and the implementers require guidance and encouragement. But sometimes even established programs somehow fail to gain traction. Although the organization may have recognized the value of security and implemented a wide variety of activities and best practices—training, design review, static analysis, dynamic scanning, code review and manual penetration testing—the results end up reflecting poorly on the large investment made.

The compliance-oriented approaches of traditional IT security frequently bring with them biases that prevent these programs from being truly successful in changing software quality. By reorienting programs to avoid some common compliance pitfalls and by including the development organization, thus integrating engineering approaches into compliance efforts, organizations can often identify simple changes that help their security effort reach its full potential.

In the following sections I'll highlight four of the most common pitfalls and explain how to avoid or address them.

Don't Do Everything—Do a Few Things Well

Compliance efforts and their results tend to be well-defined. Individual requirements are compartmentalized and have standard measures, and overall compliance status is a matter of having enough of the right boxes checked. From the abstract goal of becoming compliant, the correct management strategy is to use the available resources to cover as many areas as possible, and to spend as little as necessary to become compliant with any individual requirement.

Quality software engineering is much less amenable to this sort of measurement, except at the very finest granularity. Spreading resources too thin is a common pitfall. What's worth doing is worth doing well. A cursory or unskilled penetration test may give a false sense of security—overconfidence in the capabilities of a code-scanning tool may lead to under-allocating resources to manual code review, and limited training in areas such as cryptography may give developers just enough knowledge to be dangerous. It's better to do a few things really well than everything poorly.

Consider one example from the code review process used in security effort my company consulted on. A security code review by a dedicated team was a mandatory part of the development process for all Internet-facing applications—a rarely seen practice and capability that's usually indicative of a very strong program.

Code review by humans can be highly effective at finding difficult-to-spot and important classes of security flaws, but what did this organization's process look like? Because it was mandatory, the reviewers had a great deal of work to do and little time to complete it. Almost no time was allocated for the development team to collaborate in the process, so the code reviewers were working with little business context. Additionally, they had no access to previous bug reports from tools or manual testing, nor did they have access to a live instance of the system against which to prove their findings.

What were the results? Findings from code review were delivered to the development team mere days before scheduled go-live dates, and false-positive rates were as high as 50 percent. It was rare that these bugs could

even be triaged, let alone fixed, during the same release cycle in which they were found. Furthermore, identification of logic flaws—perhaps the most important benefit of manual analysis—was unattainable due to the lack of business context and collaboration.

A better approach—one unlikely to be heard of often in the world of compliance—is do less, and do it less often. Instead of devoting two days each to code review and black-box penetration testing on 10 releases a quarter, eliminate code review as a distinct activity. Instead, perform eight days of source-code-informed, white-box penetration testing on five releases per quarter. The reduced number of targets and broader context would allow enough time to find more flaws and to actually fix them before release.

Target Projects, Not Policies

Generally, when a compliance program is undertaken, its requirements are mandated for the entire organization and activities often are structured to have few dependencies. This usually means that compliance efforts proceed horizontally, with new policies made effective for everyone in an organization simultaneously. While some SDL activities are amenable to this approach, it makes much more sense to pick a few teams or projects to target with a vertical rollout.

To understand this, consider an analogy to another methodology change that many software organizations have undertaken recently: the use of agile development methods. No company approaches getting all its teams on agile methods by eliminating formal specifications for all its projects in Q1, increasing iteration speeds in Q3, and starting to write unit tests in Q2 of the following year. The practices of agile development enhance and support each other and should be rolled out together, one team and one project at a time.

Approach the SDL similarly: Don't exhaust your initial hard-won resources by implementing a universal mandate for threat modeling without follow-up that uses the threat models, measures their success, and iteratively improves. No SDL activity reaches its potential in isolation from the others, and there's no surer path to a security initiative's failure than a stack of expensive security documentation with no actionable next steps.

This isn't to say you should do everything at once. Pick a few activities to start with, such as threat modeling, code review, and automated scanning and roll them out together for a limited set of projects. This approach allows you to gather data on how accurate and useful your code scanner was in reducing vulnerabilities. It also lets you see what kinds of bugs the scanner missed, but that manual testing was able to find or that threat modeling was able to prevent. This is important information that helps you fine-tune each process, understand the assurance levels you're achieving, and guide the effective allocation of future resources.

Always Measure Effectiveness

Too often when it comes to compliance, the thinking is, to paraphrase Tennyson, "Ours is not to reason why. Ours is but to do and die." If a requirement is on the list, you won't be compliant without it. With such an iron-clad mandate, all incentive to measure the benefit of the activity is lost. So, for example, while most organizations have password rotation policies, few quantify how much security benefit such guidelines provide. Not many concerns attempt a cost-benefit analysis to see if changing passwords on 15- or 45-day cycles would be better than 30, because there's no reason to question the standard.

Avoid making this mistake when quality is your real goal. Typical training regimens provide a great example of this error. Often the only two measurements we see for evaluating instructional programs are: did everyone get security training and did the training mention the major vulnerability classes? If your business goal is more-secure software, though, have those two hard-earned (and probably expensive) checked boxes actually

improved your software quality?

Measuring effectiveness comes back to the core idea of relating software security directly to business goals. Before you start any activity, understand what you're trying to achieve and find a way to measure whether it works. Are developers and testers retaining what they learn? Can they apply it to the real problems they have to solve? Are teams that have been trained producing software with fewer bugs or eliminating bugs earlier in the lifecycle than teams that haven't been trained?

Metrics don't have to be expensive to collect—a simple survey might be enough to tune a training course—but they have to be there. Developers are savvy enough to tell apart meaningful activities tuned to produce results and hoops to be jumped through. The difference between successful and unsuccessful security programs is often one of mistaking the hoops for the results.

Setting principled goals, measuring the program's progress against those goals, and improving the process at each step will help motivate developers and change the internal culture of quality. Showing progress against meaningful metrics also provides a good justification for the effort and will support the procurement of additional resources required for the maintenance and growth of the program.

Just Because Everyone Else Is Doing It ...

Just as there are no silver bullets for the general software-development process, there can be none for developing secure software. Be wary of anyone telling you they have the one universal solution for all your security problems. Start small, adopt practices that are appropriate to your organization, and learn from the experiences of others with similar security needs and resources.

But how do you know where to start? The list of things to do seems huge.

A recent survey of the world's best software security programs observed that, essentially, all of them performed the same set of nine activities. At last: best practices! Shouldn't you set as your target the procedures the top 10 programs in the world agree on?

If you look at the world's top 10 navies, you'll find that they all deploy aircraft carriers and submarines, and their admirals would be quick to tell you that these technologies are the most effective way to project force at sea. Of course, as you keep looking, you'll find that nobody *but* the top 10 navies has submarines and aircraft carriers. Such fleets require vast expenditures, support structures and technical capabilities that are only within the reach of a major power. Many smaller countries would find little use for equivalent resources and might very well be bankrupted trying to acquire and maintain such systems.

Similarly, a small software-development organization may well bankrupt itself following practices and trying to manage tools whose purpose is scaling secure engineering to thousands of developers managing many millions of lines of code.

The big-ticket, silver-bullet solutions for security compliance tend to be better at killing budgets and credibility than bugs—unless an organization is well-prepared for the introduction of such products. The most successful organizations aren't those that check all the big boxes, but those that start small and that grow sustainable and manageable security programs closely related to the goals of the business.

Before you buy that expensive software package, build your expertise with the growing arsenal of free and built-in security tools available on the major enterprise software platforms. Get comfortable triaging and fixing all the bugs FxCop and C++ /analyze (or FindBugs if your organization uses Java) can turn up, and get your code to compile cleanly against the SDL banned API list before you spend a penny doing anything more

complex.

Once you're used to managing and running cleanly with these tools, you're much more likely to succeed with something more comprehensive. You may find that the free tools are just what you need, or they may leave you eager for bigger and better. You haven't lost anything by starting with free.

Harmonizing Quality and Compliance

While it may be starting to sound like the SDL is antithetical to compliance, that's definitely not the case. The two target the same basic needs, and harmonizing and integrating them can produce significant cost savings. When extending a compliance program to improve quality through the SDL, you can avoid common pitfalls and maximize your success with the four strategies discussed.

Don't try to do everything at once. Start with what you can do well. Increment your efforts project by project, not policy by policy. Always measure the effectiveness of your efforts for your own business bottom line. Grow into your program, don't buy into it.

Most importantly, get started now! There's no minimum increment for quality. Start by taking advantage of the free tools and guidance at the Microsoft SDL Web site (http://www.microsoft.com/security/sdl/default.aspx). Download and try out the SDL Optimization Model to evaluate the current state of your organization's capabilities, and use the professional services and training available through the Microsoft SDL Pro Network (http://www.microsoft.com/security/sdl/getstarted/pronetwork.aspx) to start improving quality and delivering more secure software while you meet your audit and compliance goals.

Brad Hill *is the Director of SDL Services at iSEC Partners, a full-service security consulting firm that provides penetration testing, secure systems development, security education and software design verification. Visit isecpartners.com for more information.*

Thanks to the following technical expert for reviewing this article: Michael Howard