Editor: John Steven, jsteven@cigital.com
Gunnar Peterson, gunnar@arctecgroup.net

Putting the Tools to Work: How to Succeed with Source Code Analysis

Pravir Chandra
Secure Software

Brian Chess
Fortify Software

John Steven
Cigital

Code analysis tools can play an essential role in creating secure software. They can help catch common coding mistakes such as buffer overflow, cross-site scripting, SQL injection, and a variety of race conditions. With a certain amount of customization, they can also provide for deeper, application-specific inspection as well as a general audit against custom coding standards.

But organizations often find that introducing a new breed of tool into the development process is easier said than done. To be successful, the new tool must fit smoothly into the existing process—it has to make a difference but not cause such a disruption that it's perceived as a source of busy work rather than the solution to a thorny set of problems. It's no small matter to create a good strategy for tool adoption.

Adoption anxiety

You're reading a magazine about security, so we won't waste words trying to convince you how important security is or that defects in your code can compromise your software. (If you still need convincing, we recommend *Security Engineering*[1] for a general look at building security systems and *Building Secure Software*[2] for a dive into software-specific security issues.) Chances are good you're already looking for ways to improve the security of the code your organization produces and that you know you're balancing the risk of shipping bad code against all the other risks in your environment.

Lately, source code analysis has become a popular option for catching bugs before they turn into security headaches. Most articles about source code analysis (including one that appeared in this department two years ago[3]) focus on how static analysis tools work—that is, the algorithms and heuristics that enable a tool to wring bugs out of code. But here, we're interested in a different problem: how to successfully integrate a source code analysis tool into a big and chaotic software development organization. (All the big software development organizations we've ever seen are at least a little bit chaotic.)

At first blush, this might not seem like much of a problem at all. Get the tool, run the tool, fix the problems, and you're done, right? Wrong. Changing habits is hard, and many software developers are in the habit of writing code without thinking about security. It's unrealistic to expect atti-

tudes about security to change just because you drop off a new tool. Adoption is not as easy as leaving a screaming baby on the doorstep.

Of course, getting software security right involves a lot more than just tools. Process augmentation, education, and training are essential, as is a well-grounded approach to risk management.

Three questions

In our experience, three big questions must be answered before adopting a tool. An organization's size along with the style and maturity of its development processes all play heavily into the answers to these questions. None of them has a one-size-fits-all answer, so let's consider the range of likely answers for each.

Who runs the tool?

Two different actors typically control the tools in an organization.

The central security team. You'll need to ensure that your security team has the right skill set—in short, you want security folks with software development chops. Even if you plan to target developers as the main consumers of the information generated by the tool, having the security team participate is a huge asset. They bring risk management experience to the table and can often offer big-picture security concerns as well. In the end, remember that the security team didn't write the code initially and won't have as much insight into it as the development team, so launching with just the security team on board is tougher. It helps a lot if you already have a process in place for the security team to give code-level feedback to developers.

The developers. Developers possess the best application-specific knowledge, which is important in the battle to keep remediation resource-efficient. Combine this with the vulnerability details the tool provides, and you have a solid case for letting development run the operation. On the flip side, developers are always under pressure to build a product on a deadline. It's also likely they won't have the same level of security knowledge or expertise as members of the security team. Thus, a crucial question is how much time developers will carve out to use the tool—both up-front and recurring. To that end, training your development staff to be more security savvy is also critical and goes hand-in-hand with their ownership of the tool.

When is the tool run?

Now that you've determined who will run the tool, the next step is figuring out when.

While the code is being written. Studies too numerous to mention have shown that the cost of fixing a bug increases over time, so it makes sense to check new code promptly. The first way to accomplish this is by integrating the source analysis tool into the developer's desktop so that developers can run on-demand analysis and gain expertise with the tool over time. Another method is to integrate scanning into the code check-in process, thus centralizing control of the analysis run. It costs the developers in terms of analysis freedom, but it's useful when desktop integration just isn't feasible.

At build time. For most organizations, software projects have a well-defined build process, usually with regularly scheduled builds. It gives code reviewers a reliable report to use for direct remediation as well as a baseline for further manual code inspection. Also, by using builds as a timeline for source analysis, you create a recurring, consistent measure of the entire software project, which gives wonderful opportunities for metrics to ensure that the project is moving along in the right direction.

At major milestones. Organizations relying on heavier-weight processes have checkpoints at project milestones, generally near the end of a development cycle or at some large interval. These checkpoints sometimes even include security-related tasks such as a design review or a penetration test. Logically extending this concept, checkpoints can be an excellent place to use a code analysis tool. One caveat: although they can help make an already existing practice more efficient, they can be less than ideal because they make security only an occasional focal point.

What happens after the tool runs?

You know who will run the tool and when—so now what?

Output feeds a release gate. Let's say your security analysts

Enterprise Software Security Framework

In the last issue, we introduced the Enterprise Software Security Framework (ESSF) concept. Moving forward, articles will include a breadcrumb trail that anchors topics into the ESSF via one of the five pursuits outlined in the last issue. To reinforce ESSF best practices, articles also assign responsibility for each concept. Here's this article's trail:

ESSF → software development life cycle integration → adopt a code analysis tool

Code analysis tool adoption, for example, promises immediate, consistent, and scalable feedback to developers about their code's security. Be sure to read the next issue for insight into how to measure aspects of your code's security and the maturation of your organization's security initiatives, in the ESSF pursuit of governance.

Five keys to success

It sounds easy enough on paper, but how exactly can you lead your organization to successfully adopt analysis tools?

- *Start small.* Work closely with one group to sort out any kinks in the adoption process, and once this pilot group succeeds, move on to replicate that success with a wider audience.
- *Go for the throat.* Rather than trying to stomp out every hint of a security problem on the first day, pick a handful of things that go wrong most often and go after them first. This approach often nets a big impact without overwhelming developers.
- *Appoint a champion.* Find a developer who knows a little bit about every part of the system and likes to mentor other developers. Put your new champion in charge of making the tool adoption work.
- *Measure the outcome.* Tools spit out quantifiable results, so use them to track projects over time. Code analysis results can indicate where trouble is brewing, but it can also tell you whether your education and training efforts are paying off.
- *Make it your own.* Use the tool to enforce your own standards and guidelines. Spend some time investigating customization capabilities to achieve deeper inspection and maximized accuracy for your environment.

process and prioritize the tool's output manually as part of a checkpoint at a project milestone. The development team receives the prioritized results and then makes decisions about which problems to fix and which will fall into the "accepted risks" category. This is the same approach often taken with the results from a penetration test, and it can lead to the results being used to impede the application's release in a high-profile, out-of-band fashion. Because this type of review can block a project from reaching a milestone, the most important factor is to make sure teams aren't simply accepting all the identified problems to avoid remediation time and reach their milestones faster. This approach can work well, but only if the release gate has real teeth. If developers can simply ignore the results, then they have no motivation to change the way they code.

A central authority doles out individual results. Code analysis tools scan for many different kinds of problems and can allow for directed inspection of particular pieces of code. Thus, a core group of tool users can look at the reported problems for one or more projects and then pick the highest priority items to send to the people responsible for the code in question. In such cases, the code analysis tools are set to maximum verbosity; the objective is to catch everything. False positives are less of a concern because a skilled analyst processes the results prior to the final report. With this model for remediation, the core group of analysts becomes skilled with the tools in short order, which leads to greater overall audit capacity.

A central authority sets pinpoint focus. Because of the large number of projects that might exist in an organization, a central distribution approach to results management can become unwieldy rather quickly. Additionally, most of the acute security pain could cluster tightly around just one or two types of issues. With this approach, the project team will limit the tool to a small handful of specific problem types, which can grow over time according to the risk each type poses to the organization. Ultimately, this set of in-scope problem types works well as a centrally managed policy, standard, or set of guidelines. It should change only as fast as your team can adapt and account for all the problems already in scope. On the whole, this approach gives people the opportunity to become experts incrementally through hands-on experience with the tool over time.

Building secure systems takes a lot of effort, especially for organizations that aren't used to paying much attention to security. Code analysis tools can help you codify best practices, catch common mistakes, and generally make the security process more efficient and consistent. But to achieve these benefits, an organization must have a well-defined plan for tool adoption that lays out who will run the tool, when they'll run it, and what will happen to the results. The most important point to remember when making tough decisions about tool use and priorities is that the real question is usually "now versus later," not "yes versus no." Get started with a process that will produce tangible results in the near term and refine the process over time. If you've got stories you'd like to share about your success or failure with tool adoption, please email John Steven (jsteven@cigital.com). □

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: How to Avoid Security Problems the Right Way*, Addison-Wesley, 2001.
3. B. Chess and G. McGraw, "Static Analysis for Security," *IEEE Security & Privacy*, vol. 2, no. 6, 2004, pp. 76–79.

***Pravir Chandra** is chief security architect at Secure Software. His passion is fusing*

cutting-edge security ideas with the real world through the application of common sense. He now works on code analysis research and security process refinement, but his past experience includes management of enterprise security infrastructure. Chandra also wrote Network Security with OpenSSL, *a reference book for the world's most popular open-source cryptographic toolkit. Contact him at chandra@securesoftware.com.*

***Brian Chess** is chief scientist at Fortify Software, where his research focuses on practical methods for creating secure systems. Chess has a PhD in computer engineering from the University of California at Santa Cruz, where he applied his background in integrated circuit test and verification to the problem of identifying security-relevant defects in software. Contact him at brian@fortifysoftware.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 in Cleveland, Ohio. Contact him at jsteven@cigital.com.*

Tools for code quality

Static analysis is good for more than just security. Style checkers and bug finders have a useful roll to play, and although their findings might not have a direct impact on security, they can still help.

Style checkers such as PMD (http://pmd.sourceforge.net) turn up poorly named variables, duplicated code, and many other deviations from coding conventions. If you perform manual source code review (and you should), a style checker can help ensure that the manual review team won't have to dive into a whole new kind of rat's nest every time they open a file. Style checking is a good thing, but it can be hard to adopt midstream, especially after the developers have been allowed to use any style they like. It's easiest to introduce style checking at a new project's outset.

Tools like FindBugs (http://findbugs.sourceforge.net) spot common reliability problems such as dereferencing null pointers, infinite recursive loops, and broken idioms that don't achieve what the author intended. Bug detectors are much easier to integrate into an existing project, and detecting common coding errors, even if they're not directly relevant to security, can help get developers in the habit of using a checker.

New department editors

The more observant among you will have noticed by now that two new department editors have assumed Gary McGraw's mantle. We'd like to introduce ourselves and describe what you can expect in future issues. John Steven's bio appears on this page in the main text; Gunnar Peterson is a software security architect and founding partner of Arctec Group, a consulting firm that focuses on security and risk management in distributed systems architecture.