SEVEN PRACTICAL STEPS TO DELIVERING MORE SECURE SOFTWARE

HP Enterprise Security Business Whitepaper

ENTERPRISE SECURITY

Actions You Can Take Today

Abstract:
Software security is a serious problem, and it is garnering more and more attention. However, the processes that go into making an application more secure are relatively immature. Where do you start? This paper provides seven practical steps organizations can begin today – with the emphasis on practical.

1. Quickly evaluate the current state of software security and create a plan for dealing with it throughout the development life cycle.
2. Specify the risks and threats to the software so they can be eliminated before they are introduced.
3. Review the code for security vulnerabilities introduced during development.
4. Test and verify the code for vulnerabilities.
5. Build a gate to prevent applications with vulnerabilities from going into production.
6. Measure the success of the security plan so that the process can be continually improved.
7. Educate stakeholders about security so they can implement the security plan.

Any development organization can implement this security plan and begin to receive a return on their efforts within a minimal period of time. The key is to start now.

Delivering More Secure Code: The Seven Steps

No one currently working in IT can escape the carnage wreaked by hackers. Worms, private data hacks, and other exploits are increasingly designed to target specific vulnerabilities in software ranging from operating systems to business applications. For that reason, attention is increasingly focused on the application development community. The industry is starting to ask itself how it can build more secure software.

More and more organizations are making software security a priority. These organizations have found that the key to getting started is to select a few practical activities that produce artifacts that can begin an improvement cycle. These activities can be as simple as a single "gate" or a series of small tasks at each step in the software development life cycle (SDLC). However, even this approach is easier said than done. The inertia against change can be so great that it is easy to become paralyzed — which usually means security is not being addressed sufficiently at any step — not in design, not in development, not in testing, not in production.

As a way to help, this document proposes seven practical steps that development groups can take to deliver more secure software — with the emphasis on "practical." These are actions that everyone can take today. Although these steps may not provide a magic bullet that slays the beast with one shot, they will generate measurable results in a short amount of time. The key is to get started immediately.

Step 1: Quick Evaluation and Plan

The first step is to evaluate the current state of software security inside your organization and create a plan for what additional steps to take to address security. This evaluation and plan need not be a comprehensive, multi-month effort. The best way to start is by simply creating lists of activities currently undertaken to address security and activities you'd like to implement. For example, the simple table in Diagram 1 on the next page documents a project's overall software security preparedness, providing a scale to measure against.

A plan – no matter how brief or short-term – is critical for getting buy-in within the organizations. Realistically, your first plan might need to be a "proof-of- concept" that allows you to build a more thorough and aggressive plan after positive results have been obtained. Regardless of your approach, the plan should address three elements: (1) the software security infrastructure that surrounds each software development project; (2) specific security activities each project team chooses to undertake; and (3) how you will manage vulnerabilities that are found. The table in Diagram 2 provides example elements of a plan. The steps outlined in the rest of this paper are excellent components of any plan as well — so you may want to start there.

Diagram 1. Evaluation

Internal security experts exist and have been identified	1	2	3	[✓ 4]	5
Threat analysis completed for every project	[✓ 1]	2	3	4	5
Education programs for security occur regularly	1	2	[✓ 3]	4	5
Specific tools and resources have been acquired and allocated	1	[✓ 2]	3	4	5
Post-deployment security is fully integrated with overall process	1	2	[✓ 3]	4	5
Vulnerability response maximizes benefits and prevents repeats	1	2	3	4	[✓ 5]

Diagram 2. Elements for a Plan

Assigned security expert of team lead	Who: Assign individuals to tasks and roles	How issues will be tracked and reported
Automated tools that support steps	What: List steps and success criteria for each	Team processes to triage, prioritize, and take action on reported security defects
Process checklists and requirements	When: Include steps and activities in project timeline	

Step 2: Specify the Risk and Threats to the Software

Security is all about risk mitigation. Software applications that store customers' private information are more sensitive about risk than an internal application for scheduling conference rooms. So, similar to how you would list your group's capabilities to build more secure software, you should determine the risk associated with a piece of software and the threats to its safety.

Risk analysis is a field unto itself and can be found in commercial solutions such as Microsoft's STRIDE and standards-based approaches such as NIST's ASSET. Although varied in their implementation, approaches like these typically have many detailed steps and involve a significant investment of time.

A simpler technique than a full-blown risk analysis is a threat analysis, which considers the threats posed to an application. Threat analysis helps you avoid security mistakes in your design and focuses code reviews and security testing on the most vulnerable components of the application. For that reason, you should consider this step as one of the most important you can undertake.

A simple threat analysis can be divided into two phases. Phase one identifies the assets an application must protect and evaluates which assets are most important. This task can be tricky as some assets are more apparent than others and the nature of assets varies from application to application. Examples of assets include records of private information, such as credit card numbers, employee/customer records, or financial figures. Other examples include resources that an organization provides to others, such as e-commerce solutions, corporate Web sites, and e-mail support for customers. Still others are intangible resources, such as your company's reputation. For example, how would a security breach affect the credibility of the company?

Phase two of threat analysis consists of understanding the application itself and the dangers it faces from attackers. Organizations should develop a high-level model of the application's components and dataflow paths. The application's attack surface should be mapped, identifying interfaces that accept input from users or interact with other systems.

Teams should note any points on the attack surface that allow an exploit to compromise the integrity, availability, or confidentiality of an asset. Finally, rank the threats based on importance of the asset affected and the likelihood of exploit.

This threat analysis exercise, while simpler than a full-blown risk analysis, still may require a high level of expertise about the application and how attackers work. However, it does not need to be precise, and the return can be substantial in that it focuses efforts on the most important areas without a lot of waste. Of course, not even the most thorough threat analysis can prevent the introduction of security vulnerabilities during development, which leads us to Step 3.

Step 3: Review the Code

There is a simple fact in the software world — the code that gets deployed is the true instantiation of any application. Consequently, organizations should review code throughout the implementation and testing stages for security vulnerabilities that may be introduced during development. In most every case, these reviews uncover numerous security vulnerabilities that would otherwise be deployed. And, coupled with threat analysis, an organization can use the review as a verification that the software does not leave open vulnerabilities to the threats they most care about.

Many groups today rely upon manual code reviews to perform this step. Manual audits, however, require rich security expertise and tremendous investments in time. Fortunately, there are consistent and well-researched patterns of how developers introduce security vulnerabilities into applications, providing a basis for accurate, automated security analysis tools. The best source code analysis tools can evaluate multiple tiers and track the flow of data within an application. They can work on bodies of code that range from small to large, and effectively present and manage their own results so human auditors can quickly identify and prioritize potential security flaws.

Security source code analysis embedded in developer environments, such as Microsoft Visual DevStudio and Eclipse, can also serve as basic and ongoing educational tools for developers. These tools provide feedback at the point the error is introduced, allowing for a less costly fix and more concrete learning experience.

Below is a list of key capabilities required for effective automated security source code tools:

- Comprehensive identification of numerous vulnerability types
- Ability to perform varied analysis, such as global data flow, control flow and configuration file analysis, to reduce false positives
- Ability to analyze across application tier boundaries in order to find vulnerabilities that manual reviews would never discover
- Multiple filtering, querying, and sorting options on analysis results
- Support for all the languages used by your development teams
- Extensible to enforce your particular secure coding polices
- Support for analysis at the developer desktop via Integrated Development Environment plug-ins

Step 4: Test and Verify the Code

Organizations should test code for security flaws in addition to features and correctness. Testing is a complimentary technique to code reviews, a final exam for the secure software development life-cycle, and something that can also be aided by automation. The warning here is that traditional functionality testing is not effective at finding security vulnerabilities. Software quality tests are traditionally focused on verifying a set of features as defined by some reasonably well specified requirements or expected user actions. Security testing requires a different mindset and approach — for it is the absence of security that testers must find.

A common approach is to conduct application penetration testing. The purpose of the penetration test is to simulate attacks against the software to discover anomalous behavior. As with code reviews, these tests can be performed manually or with an automated tool. Similar to source code analysis, human penetration testing is capable of uncovering complex problems that cannot be found any other way. However, they require skills, expertise, and time unavailable in most organizations

Automated tools can provide a knowledge base of known attacks and rapidly fire those attacks at an application, but they often miss the subtle flaws or input fields human hackers exploit. In addition, penetration testing coverage — the amount of a running application actually exercised in a penetration test — is often very small with respect to an application's true attack surface. One solution is to use results from source code analysis to feed into the penetration test so that (1) the testing takes into account all input sources and ignores areas where vulnerabilities do not exist and (2) the remediation task is greatly enhanced.

There is no easy answer to testing for security. The most practical approach, however, is NOT to rely upon testing to find the vulnerabilities. Testing should primarily verify that the vulnerabilities found in code review have been eliminated.

Step 5: Build a Gate Code

The most fundamental and arguably the best way to get the process started and achieve tangible results is to construct a security "gate." Many organizations begin by requiring a single gate as the software leaves testing and before it is released to operations or production. This "final check" gate has the advantage that it is simple to understand and raises the visibility of security issues before release. A downside is that a milestone so late in the development cycle does not provide adequate time to adjust for security defects found in the code.

Obviously, a key question is what criteria will be used to judge whether the software is fit to pass through a gate milestone. Since the base artifact is the software itself, a practical criterion would be an audit of the code. A code review gate – whether it is required during active development, testing, or as a last check before releasing the software to production – typically discovers significant numbers of security vulnerabilities.

As the organization's secure development processes mature, the criteria can expand to require specific steps completed in detail and discovered issues remedied; i.e., threat analysis was performed, code reviews were conducted and issues uncovered were mitigated, security testing was done, no other serious vulnerabilities were found, and minor vulnerabilities were corrected. This more stringent gate may also contain a specialized security audit via independent security experts before declaring the application "production ready."

Step 6: Measure

Organizations should measure the success of their security activities so that the process can be improved to meet changing requirements. Software security is still an emerging field — as are the metrics that judge the effectiveness of activities within that field. However, there are many activities that can and should be measured to provide insight into the state of software security for an organization or project. Organizations can measure adherence to the security process as it was designed, or measure the success of the security process as it is implemented. For example, Microsoft measures the effectiveness of their SDLC by counting the number of security bulletins within the first 12 months following a release (see Diagram 3). While the Microsoft metric is a trailing metric, measures taken during development can provide sufficient time to allow remediation of vulnerabilities pre-deployment. These metrics include tracking measures such as vulnerabilities found per category, by severity, and over time. For example, one project may introduce buffer overflow or SQL injection errors more frequently than another — signaling a candidate for additional education. Audit coverage and history, vulnerability aging, and even composite risk measures are possible.

The key is to begin collecting data early on to create a baseline during development.

Diagram 3.

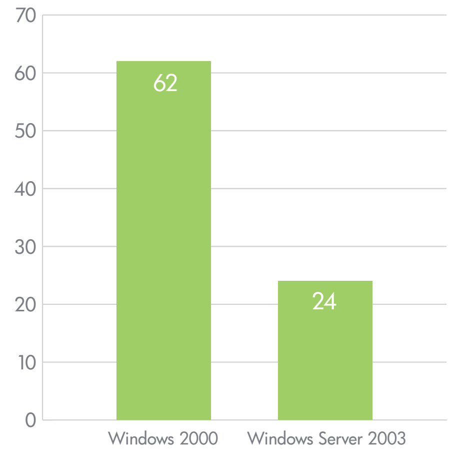


Diagram 4. Vulnerability Severities for Project Pluto

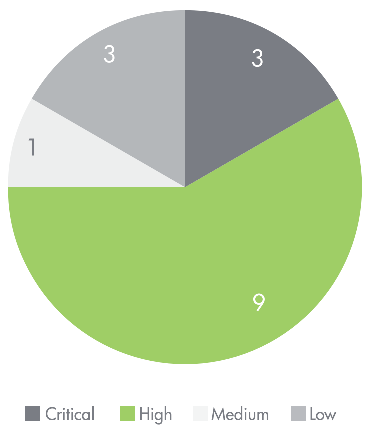
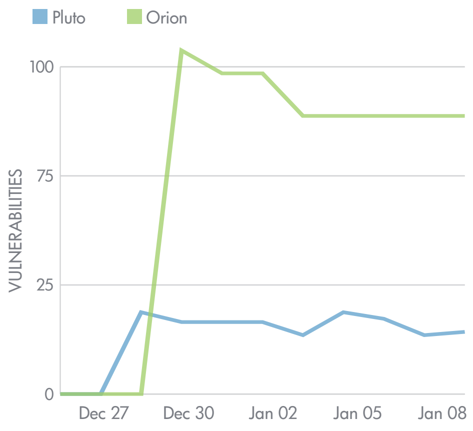


Diagram 5. Vulnerability Trends per Project



Step 7: Educate

After deciding what security measures the plan will implement, the most crucial issue is to educate the key stakeholders so they can effectively implement the security activities. Even with training, developers are not likely to become security experts because they have other pressing responsibilities. Typically, some group or selected individuals must be assigned to the role of security "leads." They may be individuals who already have an interest in security, individuals who are specifically trained in security, or individuals explicitly hired to fill that role. These experts will be responsible for implementing the security plan from beginning to end and are critical to its success.

The presence of security experts, however, does not relieve others of the responsibility for security. Security must be the responsibility of every member of the organization, even when starting small. Without full participation, no security plan is likely to succeed. However, because individuals cannot be responsible for what they don't understand, education is key.

To understand security requires knowing a lot of specific details. Developers must understand how hackers think, they need to know about vulnerable functions, and how many bits a session ID should be for a site that receives 7 million hits a day — and on and on. Organizations cannot make every member of every architecture, development, quality assurance, and operations team a security expert, so how can security be improved?

The security plan for responding to vulnerabilities suggests developing internal best practices. These are excellent security training resources for new and existing engineers. What better source of critical mistakes to avoid than those that have already been made? Leveraging internally developed best practices as educational tools provides targeted security knowledge with proven relevancy.

Consider sending key individuals in the development life-cycle to training sessions or bringing a security expert in-house to provide on-site training. Money and time invested in training are well spent, especially within the context of the previous six steps.

Ultimately, the solution is to make security a mindset that pervades the development group. Everyone should be thinking about what could go wrong each time they design a feature, produce a build, or write a line of code. Each should ask "How could an attacker take advantage of what I am doing?" If, at every step, everyone is thinking about what could go wrong, security will improve.

Conclusion

Software security is a serious problem. To mitigate the risks software development organizations must start thinking about security as a part of every step in the total software development life-cycle. Development teams can utilize a practical, seven-step plan to deliver more secure software:

1. Evaluate the current state of software security and create a plan for dealing with it throughout the development life cycle
2. Specify the risks and threats to the software so they can be eliminated before they are introduced
3. Build a gate to prevent applications with vulnerabilities from going into production
4. Review the code for security vulnerabilities introduced during development
5. Test code for vulnerabilities
6. Measure the success of the security plan so that the process can be continually improved
7. Educate stakeholders about security so they can implement the security plan Any development organization can implement this security plan and begin to receive a return on their efforts within a minimal period of time. The key is to start now.

Any development organization can implement this security plan and begin to receive a return on their efforts within a minimal period of time. The key is to start now.

About Fortify

HP Fortify's Software Security Assurance products and services protect organizations from the threats posed by security flaws in business- and mission-critical software applications. Our software security assurance suite, HP Fortify Software Security Center, drives down costs and security risks by automating key processes of developing and deploying secure applications.

About HP Enterprise Security

HP is a leading provider of security and compliance solutions for modern enterprises that want to mitigate risk in their hybrid environments and defend against advanced threats. Based on market leading products from ArcSight, Fortify, and TippingPoint, the HP Security Intelligence and Risk Management (SIRM) Platform uniquely delivers the advanced correlation, application protection, and network defense technology to protect today's applications and IT infrastructures from sophisticated cyber threats. Visit HP Enterprise Security at: **www.hpenterprisesecurity.com**

ESP-BWP00X-MMDDYY-XX, Created Month 2011