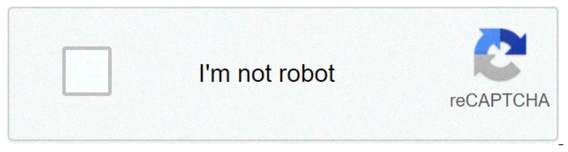

Continue

Secure software development life cycle

Technical Writer, Web Security Expert and Software consultant.Avoid making common security mistakes that make your software vulnerable to attack. As a developer, you’re certainly aware of the importance of following security best practices. But many times, we miss certain things, maybe because it hasn’t become second nature to us yet. Unfortunately, we realize only after a security issue has arisen, and it’s marked on our brains like a spot of indelible ink.Many instances of poor security practices come to the surface when it’s too late. Many mistakes are made even by large organizations and experienced developers. Here, we’d like to cover some common steps which will prove helpful in enhancing your software security.1. Instead of Using Encryption, Hash the PasswordsSometimes developers use encryption to store passwords. Due to this, there’s always a chance of someone finding the decryption algorithm or key. To avoid this, use hashing, as it doesn’t have a direct reverse. This means no one can reverse it unless they already have a mapped table from plain text to hash.2. Avoid Adding Secret Backdoors in SoftwareWhether you have a genuine reason or not, don’t add backdoors to access software. It’s best to avoid putting backdoor access into the system. Because sooner or later someone else may find it. Exploiting backdoors is a common tactic used by cyber attackers. Adding backdoors can damage the reputation of your organization and portray you as a bad guy who’s enabling stealing personal data of users, adding malware, or hijacking devices.3. Make Sure to Require User Authentication on Every PageSometimes it’s easy to accidentally skip essential steps for software security. One common issue is not requiring authentication on every page. For instance, a copied URL with confidential information (such as a confirmation page) can be opened into another browser without providing login details. This shouldn’t be the case—be sure to ask for login details instead of showing the page directly.4. Have a Plan For Security PatchingAs a software developer, it’s your responsibility to keep your software updated and free from any vulnerabilities by providing regular updates and patches for your software. Be sure that critical security issues are patched quickly, before attackers can take advantage.5. Test Before Publishing Your SoftwareOften security loopholes in software are found through testing. That’s why it’s important to follow proper testing practices before publishing any software—no matter whether it’s a simple or a complex application. More testing (such as checking the performance on different platforms, testing any input conditions, etc.) will help you in providing software security before problems occur.6. Use a Code Signing CertificateEven if your software is well developed and published, its reputation is what decides whether it will become successful or not. To increase the reputation, sign your software using a Code Signing Certificate from a trusted Certificate Authority. This will help you in many ways:avoid security warning signs for unsigned softwaredemonstrate the trustworthiness of your softwareprove the integrity of the softwareboost the confidence of your usersPreviously published at Hacker Noon Create your free account to unlock your custom reading experience. When security vulnerabilities in a vendor's software are exploited, significant costs are faced by the vendor and its software users. Software with security vulnerabilities harms an organization's reputation with customers, partners and investors. It increases costs as companies are forced to repair unreliable applications, and it delays other development efforts as limited resources are assigned to address current software deficiencies. With the increased scrutiny of internal processes and controls resulting from mandates such as the Sarbanes-Oxley Act, executives are demanding that IT improve the development process in order to create more secure and reliable software. Fix a flawed development process All software has bugs, and a large number of these bugs have security implications. It's not just buggy code that is an issue. Software behavior and coding practices that were considered safe at the time of writing may now be ripe for exploitation by malicious hackers. The problem for software development organizations is that they must simultaneously reduce software vulnerabilities while keeping operational costs in check. Plus, any new development strategy is expected to be applicable across geographically distributed teams -- including offshore service providers. Something has to change. Software quality, and specifically software security, must be improved, and the most effective means is to address the root causes of poor software -- the defects in the source code. But to improve software, the current flawed development process must be addressed. Start by assessing the situation Rather than throwing more money and resources into a flawed process, companies need a new plan of action. Before implementing new processes and investing in new tools, companies should consider these steps: Ensure information flow: A smart software development process ensures timely and effective information sharing. This enhanced knowledge improves communication between management and the development teams, allows developers to work with solid and secure architecture and coding practices, provides visibility into an application's context and its health at any point in the development life cycle, and lets IT manage software assets like other business assets. Know the goals: A key consideration for any software security initiative is whether the goal is to audit the current state of your software's security or to implement a change in current development practices. An audit is a one-time event, while an in-process deployment can improve the security of existing applications, as well as provide the necessary experience, tools and processes to extend the concept of secure development throughout the entire development organization. Determine strategies for new and existing code: Attempting to retrofit secure coding practices into existing code can lead to unforeseen issues. Often, the prudent course is to focus efforts on cleaning up critical, exploitable problems. In contrast, a new development environment offers the opportunity to implement secure coding practices right from the beginning. Understand the range of security issues: These issues include: security vulnerabilities (problems that can be exploited by an attacker) and security flaws (problems that may exhibit themselves at the design level); security symptoms (theoretical threats that indicate a potential security vulnerability); and security weaknesses (nonuniform distribution of security symptoms and vulnerabilities in a given software system). A checklist to produce high-quality, secure code To create more-secure code, organizations can implement a six-step strategy that combines straightforward processes supported by multiple technology options. The steps are as follows: Conduct a software audit. An audit will uncover and help prioritize existing security vulnerabilities and code quality issues. Implement a "stop the bleeding" plan. The plan, accomplished by deploying an extensible, rules-based "quality/security compiler," ensures that tactical issues are addressed and enforces proper coding practices in new development. Perform a more detailed in-process audit. The in-depth audit looks for design and architecture weaknesses and correlates them with known security vulnerabilities. The findings are used to help plan a strategic road map. Convert in-process audit findings into policy. The next step is to clean up "one-time fixes" and update a rules-based security compiler to include any new policy requirements. Measure improvement. To gauge the initiative's success, it is imperative to monitor the trends and results from implementing the new processes and tools. Manage iterations of improvements. This iterative process requires repeating Steps 3 through 5. The key is to streamline the change process based on priorities, starting with the highest priority weaknesses. Tools and techniques to support improved software development There are several process and technology options available to support the implementation of this checklist. Organizations should invest in a combination of them to achieve the best results. Use, but understand the limitations of black-box testing. When addressing software security, traditional testing approaches fall short. While black-box testing will continue to play an important role in identifying and removing traditional, functional quality issues, it should not be the only tool used. Judiciously employ manual code reviews. Rigorous code reviews can enable detection of implementation-level security vulnerabilities but they require a disciplined methodology in order to obtain results. While manual code reviews are helpful, they can be quite costly and prone to human error. Use automated defect detection tools. Static analysis tools analyze source code and identify coding errors that can lead to security vulnerabilities. These tools complement traditional testing and manual code reviews. Conclusion Traditional software development is failing in today's globally networked environment. Insecure software reduces the productivity and wastes the money of businesses, and, as a result, harms the reputations and bottom lines of software producers. While an improved development process entails investment, it can be incremental and highly automated. Just as an efficient manufacturing process can provide a competitive advantage, high-quality software development can help eliminate costly problems and improve profits. By addressing and investing in fixing the software security issue today, development companies can create a repeatable, sustainable approach for delivering consistently high-quality, secure code. Djenana Campara is the founder and chief technology officer of Burlington, Mass.-based Klocwork Inc., a provider of static analysis software that detects software security vulnerabilities and helps improve overall software quality. Before founding Klocwork, Campara spent 11 years at Nortel Networks, advancing from senior software designer and team leader to manager and system architect. At Nortel, Campara and her team developed the technology that ultimately would lead to the founding of Klocwork. She has been awarded three U.S. patents for her software development work in creating Klocwork inSight, and has two patents pending. Copyright © 2005 IDG Communications, Inc. The following article is part of a series of articles about our NerdWallet Internship program. Saswata Gupta shared their experience as an software engineer intern. If you are curious about joining NerdWallet as an intern or full-time employee, please apply for one of our open positions! What This Is About I’ve just finished up with my internship at NerdWallet as a security engineer and I couldn’t help but realize how different this internship was compared to my previous five internships. That’s not to say my previous experiences were all homogenous, but I had definitely become accustomed to a pattern of work that I didn’t experience during these past few months. For instance, there was a week I spent where I hadn’t written a single line of code which was crazy to me at the time. That’s not to say my time at NerdWallet was tarnished, but actually, the opposite since my goal for internships is to get a wide breadth of experiences. I wanted to understand why this experience felt so new to me. The obvious conclusion was that security engineering was the outlying factor, as my previous experiences were more software development related. Putting more thought into it led me to understand the stark differences between these two roles in the tech industry, but also how they are similar. In this post, I hope to give those of you curious about these domains an overview of both and how they compare. What They Are Security Engineer: Goal: Ensure that existing software systems cannot be exploited and private data cannot be accessed by attackers. Domain of Expertise: Methods of attack hackers can exploit and how to mitigate them. Major Types of Work: Exploratory work – combing through source code or documentation to better understand a system and thus its vulnerabilities. Collaboratory work – discussing with other teams / third-party vendors about how the system behaves and how it could be vulnerable. Design work – constructing a solution outline to patch up a vulnerability within a system considering all of its effects. Software Developer: Goal: Create new software systems and/or maintain existing systems to ensure they function as expected and are performant. Domain of Expertise: What an effective software system looks like and how to maintain that. Major Types of Work: Feature / Project work – creating a software system or updating one through programming. Design work – outlining how a software system should behave with all functional requirements in consideration. Collaboratory work – Discussion with regarding behaviour of a system or how multiple systems may interact. What’s Different What stood out to me while on the job as the biggest difference was the lack of programming, and in a broader sense, a lack of structure in the work being done. As a developer, it is much easier to know what is correct / what works and what isn’t / doesn’t. As a security engineer, the problems being solved are more vague in the sense that there is less of a definitive correct answer. An example of this is the main project I worked on during my internship: improve input validation within our backend code. There are so many ways input validation within code can be improved, just in terms of which libraries are used, or even using writing our own libraries. Aside from that, there are many other factors that must be considered, which only make the correct solution harder to identify such as the practicality of expecting developers to code the input validation correctly and how we could monitor the state of input validation to assess the situation and confirm our solution works. What Both Share Though the time spent on types of work done may be different between the roles, it wouldn’t be correct to say that any of the types of work listed solely belong to either role. I can confidently say that doing the work in one role will definitely improve the quality of work in the other, as the type of work and the domain specific knowledge helps towards both goals. For example, if a security engineer is well aware of how a developer writes code for a system, it is much easier to identify its behaviour and thus vulnerabilities as well. This goes in the other direction as well, as a developer aware of common security flaws can write more secure code. Why Both Are Valuable Broadly speaking, security engineers tend to have less structured work and place an emphasis on in-person and written communication, while developers are focused on programming and designing systems. Both are necessary for a successful product, and both have skills transferable to the other. The only conclusion I can state with complete confidence is that I gained many valuable skills during my internship that will be transferable to any future role I take in the tech field.