03 Feb 2023 |Dexter McGriff

How To Secure Your Software For Government Agencies!

What is Secure Coding? Traditional methods of cybersecurity reactively catch vulnerabilities after software is already deployed to production, but secure coding changes traditional cybersecurity to a proactive approach by encouraging developers to validate software before it's published to the public. Secure coding follows the "shift left" approach to cybersecurity, where software is scanned for vulnerabilities before deploying to production, but it changes the software development lifecycle and must be carefully integrated into your developer workflows. **Elements of Secure Coding** You might already have some elements of secure coding in place, but missing any critical best practices in your software development lifecycle could result in oversights and eventual compromises. Developers often don't understand the elements of secure coding, so a third party helps build cybersecurity into their workflows. Good secure coding practices do not inhibit the velocity of deployments, which means that many cybersecurity elements will be automated seamlessly into development lifecycles. **A few secure coding best practices include:** • Monitor code repositories for hard-coded secrets, passwords, and keys. Developers often put passwords, usernames, secrets, and keys for remote API (application programming interface) access and other sensitive data in configuration files. These configuration files are stored in a code repository with public access, leaving secrets openly disclosed to anyone. Monitoring tools can be used to scan code repositories for any stored secrets and alert developers to remove them. • Avoid saving secrets in application logs. Usually, logging setups involve agencies and developers. Logs should never contain passwords and secrets. For example, if a user authenticates into your application, do not write the successful password to logs. Instead, write an event that indicates the user login was successful. • Validate configurations. When software is deployed, agencies set configurations on infrastructure to host it. For some services, configurations are written into deployment files. Misconfigurations can be avoided by validating configurations before and after deployments. • Run vulnerability scans on software prior to deployment. A static application security tool (SAST) scans code while it's being written to catch common vulnerabilities early in the software development lifecycle, and it will alert the developer. This strategy should not replace vulnerability scans after deployment, but it adds penetration testing to your "shift left" security workflows. • Educate developers and agencies on common vulnerabilities in software. For developers to write more secure code, they should understand the way hackers exploit vulnerabilities. Educating them on vulnerabilities helps them think ahead during the development process. This list isn't exhaustive, but it's a good starting point. You still must secure your environment from common vulnerabilities and malware. Developers and government agencies are targets for phishing and social engineering attacks. They must know what both attacks look like and know what to do when they believe they are a target. Email security helps stop phishing emails and messages containing malware, but developers and government workers must know not to fall victim to social engineering. For example, after an attacker gains access to a developer password, the attacker might use social engineering to trick the developer into divulging their two-factor identification number or accepting a two-factor authentication request. **What Vulnerabilities Result from Insecure Coding?** Any number of exploits can affect vulnerable software, but some

common attacks target software code mistakes. For example, buffer overflows are common in software where code reviews aren't carried out. Buffer overflows happen when input isn't validated and sanitized and allowed to pass to application methods. In a buffer overflow, malicious input allows an attacker to run arbitrary code on a server or on a target machine. The infamous Heartbleed attack was an unknown overflow bug in code that existed for years before it was detected. The biggest issue with web-based vulnerabilities and insecure coding practices is code injection. Code injection is an umbrella term for vulnerabilities in software that allow an attacker to send malicious code as input and run it in another section of the application or on another user account. Malicious code can steal session tokens, execute arbitrary code on a user browser, or possibly modify database content. Developers can't stop distributed denial-of-service (DDoS) attacks, but a standard denial-of-service attack (DoS) is a potential issue. A standard DoS happens when malicious input causes the application to crash, and it's no longer available or can't function normally. A DoS could cause issues for agencies or internal end users that use the system, but the attack could greatly affect revenue, agency satisfaction, and end user productivity. **Getting Started with Secure Coding** If you don't have security staff onsite, you can still foster good cybersecurity and secure coding in your development lifecycle. Developers need tools to help them, so finding a good SAST is a good addition to the entire development lifecycle. A good start is to have a third party review your code for any possible vulnerabilities. A good penetration tester will also scan your software using blackbox testing, which means that security people scan software in the same way as an attacker to find vulnerabilities. To foster secure coding, developers need training and education in cybersecurity, the ways to create bug-free code, and what can be done to monitor and remediate vulnerabilities. Developers might need to work with third parties to set up monitoring and logging of their applications, but these methods are critical for detecting malicious activity after software is deployed to production. Both teams should work together to set up alerts and notify the right people to quickly remediate any vulnerabilities when they are found in production. The way you implement secure coding depends on your own development environment and the way developers deploy to staging and production. Some security controls can be automated, but any automation scripts should be fully tested before being trusted to catch vulnerabilities. **Conclusion** If your software is open to the public and you are chasing vulnerabilities, changing to a more "shift left" cybersecurity approach with secure coding greatly reduces the risks of a compromise. Developers educated in the many ways hackers compromise systems and their software will be empowered to write code that avoids exploits. You can't completely eliminate cybersecurity risks, but you can take the necessary steps to avoid critical data breaches. Secure coding is just one step in building bug-free software resistant to the latest exploits.