CHECKMARX
Source Code Analysis Made Easy

The AppSec How-To: Achieving Security in DevOps

How do you integrate security within a Continuous Deployment (CD) environment - where every 5 minutes a feature, an enhancement, or a bug fix needs to be released?

Traditional application security tools which require lengthy periods of configuration, tuning and application learning have become irrelevant in these fast-pace environments. Yet, falling back only on the secure coding practices of the developer cannot be tolerated.

Secure coding requires a new approach where security tools become part of the development environment – and eliminate any unnecessary overhead. By collaborating with development teams, understanding their needs and requirements, you can pave the way to a secure deployment in minutes.

What is DevOps all about?

DevOps is a continuous development process where small features and bug fixes are frequently deployed within short periods of time. As a new development methodology, DevOps is not restricted anymore to young start-ups. Numerous large enterprises such as Facebook, Netflix, Etsy, LinkedIn and Twitter have already adopted DevOps. Amazon, which closely follows the DevOps model, is known to have more than 1000 deployments an hour.[1]

Tradition vs. Disruption: Web application controls in a DevOps environment

Can traditional Web application security controls fit in a disruptive DevOps environment? Let's take a look at the common Web application security toolbox:

- **Penetration Testing**. A most valuable method to test security, there is one inherent problem: it takes time. Whether penetration testing is performed internally, or by a third-party, it takes a few days to test the application and then some time to produce the findings. When findings are at last presented, it takes time to analyze the results, get the affected development groups together and prioritize the tasks. It's not rare for a big project to produce a 300 page findings report after undergoing a three week assessment cycle, two days of a follow-up analysis, and an additional two weeks just to start incorporating the fixes within the development process.

[1] http://www.slideshare.net/AmazonWebServices/advanced-topics-session-1-continuous-deploymentpracticesonaws

- **Web Application Firewall (WAF).** A WAF requires tuning and learning the application it protects. For applications that do not change much, configuration the WAF requires a few hours to a few days. But what happens when applications constantly change? The WAF in this case would require continuous configuration and is simply not a solution for such a dynamic process.
- **Code Analysis.** This method gained a bad reputation for simply being too slow. Whether it's the setup time, running time or analysis time – anything that takes more than a few seconds cannot truly be integrated within DevOps.

Required: A new secure Software Development Life Cycle (SDLC) approach

The solution is to incorporate security already from the start of the development process. Consider the project from a security standpoint and make security a default process within the SDLC.
These following steps can help you achieve this goal.

Step 1: Plan for Security

Research what technologies and processes you will run into throughout the development and deployment process. Accordingly, consider their security aspects:

1. Security in technology
   a. Identify non-secure components and frameworks. For example, some organizations analyze their entire code base to map all their non-secure patterns, frameworks and libraries.
   b. Choose a programming language which has built-in security patterns. Each new PHP release, for instance, deprecates non-secure patterns from previous versions. Similarly, almost all frameworks had security breaches and provide the required fixes for them.

2. Security in code development
   a. Map security sensitive code portions. Not all code is created equally. For example, security in your test library is definitely not as important as a password change mechanism, a user authentication mechanism or a credit-card processing mechanism.
   b. Place extra security care around sensitive code portions. Flag the sensitive code portions so that when changes are applied to those modules they trigger a code review, special testing, and a separate scan specifically for those modules.

3. Security in features
   a. Anticipate regulatory problems and plan for them. Eventually, you'll hit regulations. Not preparing for them in advance will cost you later due to product changes, add-ons and modifications to already structured code. Design the incorporation of regulation aspects into the code. Design compliance verification into the process testing.

Step 2: Engage the Developers. And Be Engaged.

DevOps places the developer at the center of the process. And it is the developer that is held responsible to a high code quality standard. How can security teams communicate also the seriousness and importance of security?

Various companies have found the following recommendations helpful to bridge the security-developers gap:

1. **Connect developers to security.**
   - Position a "security champion" in each development team. Share with the champion security articles on the threat landscape and hacking motivations. Go together to your local OWASP training.
   - Make security training valuable. Instruct developers on effective reading of vulnerability descriptions, communicate the risk of vulnerable patterns in the code, and discuss correct mitigation strategies. Practice through security development exercises which present developers with their common and repeating coding issues.
   - Share attack details. Relate developers to the actuality of security and hacking. Present the logs of hacking attempts to demonstrate how their secure coding practices prevented the attacks from succeeding.

2. **Setup an online collaboration platform**. For example, generate a discussion on any sharing and collaboration platform, such as Jive or Confluence, by post a security problem and presenting ways to solve or prevent the issue. Take this one step further and establish a collaboration platform just to share security issues.
3. **Have an open door approach.** Be there when developers come to ask questions. For example, work with developers on how to fix and prevent the lesser known coding flaws.

Step 3: Arm the Developers.

Provide the developers with the right tools to help them prevent and mitigate security vulnerabilities.

1. **Secure frameworks**
   Secure frameworks are your built-in tools for securing the code already at the base. Currently, there is a pretty nice range of secure frameworks to choose from. Examples include Spring Security, JAAS, Apache, Shiro, Java SE, Symfony2. Furthermore, Ruby on Rails has a very wide range of security solutions for input validations, authentication and session management. OWASP also provides an open-source security framework for various languages named ESAPI.

2. **Use source code analysis tools for security feedback on the pre-commit stages**
   Running a source code analysis tool is a seemingly contradiction to this article's preface which considered it to be too slow. As mentioned, any delay due to security scanning cannot be tolerated in a DevOps environment which requires delivery every few minutes. But as the development environment changed, so have different scanners adapted in order to provide the development team with a rapid response. How can developers take advantage of these new scanning features?

- **Run the scan on small chunks of code.** Only scan the change between the last scan and the current scan. In this way, the scanner can scan small code portions without requiring the whole project to be set up and scanned for hours.
- **Access the tool from within the development environment.** Developers are responsible for testing their own code within their chosen IDE environment. This should also include testing the code for security. Developers can either do this through a code review or by using SCA tools. Only when the developers are confident that their code is secure, then they can commit the code into the source code repository.

Step 4: Automate the Process

The building block of DevOps is automation. The same should go for security. Security should first fit into the standard automated continuous deployment process. As a second step, apply application security testing tools – whether static or dynamic – that are capable to produce results in a very short time.

1. **Integrate within your build (Jenkins, Bamboo, TeamCity, etc.) different application security tools such as Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST).**

   When the code is committed, the build – typically through tools such as Jenkins or Bamboo – should trigger the scan of both dynamic and static testing tools. The static testing tool performs a comprehensive scan in order to cover the case where several developers commit simultaneously. The dynamic testing tool works as a self-learning environment where it monitors the positive tests written for regular testing tools. The tool also runs inputs on negative tests to verify the catching of inputs not caught by the positive tests.

2. **Fail the build if it does not pass the bar.**
   We realize that at first you might be put off by the sound of this notion. But just like a high-priority bug that does not pass the development stage, security should be considered on the same rung of importance.

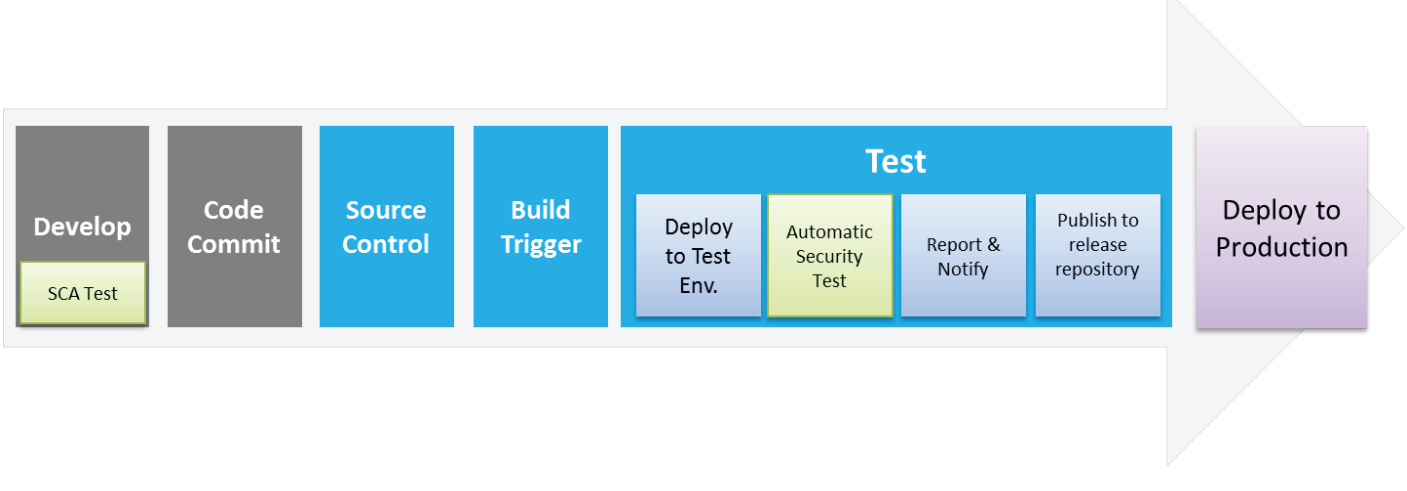

Diagram 1: Security within Continuous Deployment

Step 5: Use Old Tools Wisely

Don't start throwing away the old tools immediately. These still come in handy – but used in different ways:

- **Penetration Testing.**
  Ensure that your systems are military-grade by ordering a penetration testing on a periodic level, say every six months. At this stage, findings will be minor if non-existent – but these can act as a reassurance to your system.
  Additionally, have your customers perform penetration testing on your systems. First, this might be a requirement since some customers are required to audit third-party systems to meet compliance. Second, a cloud environment relationship is based on the trust between the provider and customer. Allowing customers to perform penetration testing on your systems will raise this level of confidence. When security is ingrained into your system, you have that assurance of zero findings.

- **Web Application Firewall (WAF).**
  Use the WAF as a solution for the more stable parts of the Web App. Maintain the WAF by performing a fine-tuning every once in a while to ensure that the WAF still guards the main functions that do not change too often.

- **Code Review.**
  Perform a code review for security sensitive code portions. Use a code review, for example, to ensure the security of authentication modules and credit-card handling modules.

DevOps is Happening. Right Now. Last Word of Advice

Security can and should be an integral part of a continuous deployment process. But start small to avoid being overwhelmed and making the process too hard to implement. Start with those features that are more accessible and less critical, and build up the security process from one deployment to the next. Eventually, you'll achieve small successes as proved by the reduced amount of vulnerability feedback for those security-enhanced features. Go with these results to management and receive their support to start integrating security into each and every part of your development life cycle.