TechTarget | ComputerWeekly.com E-guide

CW+ Content

A guide to continuous software delivery

In this e-guide

In this e-guide:

Software empowers business strategy. In this e-guide we explore how to delivery new software-powered functionality for continuous business improvement

Business leaders and IT heads recognise the value of software in driving business innovation. This is even more important now that people have adapted to new ways of work and the customers transact with businesses.

A recent article published by management consultant, McKinsey, discussed how post-Covid business operations need to adapt to meet the changing demands of customers. Specifically, the authors of the paper covered rapid digitisation, which will force business leaders to reassess the workflow and processes that run through their organisations.

McKinsey is not alone. Spending on software is rising. With a worsening economic climate, software, under the guise of digitisation, is becoming a pillar of post-pandemic business strategies. Business leaders are turning to technology to enable their organisations to adapt quickly in order to drive business development and adapt to providing more personalised products and services. For IT leaders, this means there is a business imperative to optimise all of the steps involved in turning software innovations into the digital technologies that will drive the business forward.

The likes of Amazon, Facebook, Google and Netflix and other web giants are in a position to rollout out software updates on a continuous basis to provide increasingly sophisticated taylored services for customers. Such organisations tend to build their own systems to support their pace of software innovation. Other organisations can turn to integrated software delivery platforms to support their continuous software delivery ambitions.

Continuous delivery is a pipeline covering all aspects of a software-powered projects from development and quality assurance, testing and through to deployment as well as the feedback loop for continuous improvement. Generally, there is not one single system to cover the end-to-end process. In fact continuous delivery is an approach to optimising the software development business process, which relies heavily on automation and analytics to ensure every aspect of the software development process runs smoothly.

Analyst Forrester recently looked at the market for integrated software delivery platforms. Among the key recommendations in the Forrester Now Tech: Integrated Software Delivery Platforms, Q2 2022 report is that IT leaders should determine desired outcomes first prior to selecting a platform. "Many leaders make the mistake of choosing an ISDP solution first and then struggle to make it work within their organisation. The report's authors suggest that IT leaders should first start by having a good understanding of the desired outcomes they want to achieve. This, they say, may be tactical, such as faster release cycles, or strategic, such as transformation into product teams. The choice of ISDP then helps them achieve these outcomes.

In this e-guide

How to create and manage a rock-solid DevSecOps framework

Stephen Bigelow, Senior Technology Editor

Security has long been treated as an afterthought in software development. Developers work to create effective code but only consider software security in the testing and deployment stages of the development lifecycle. With accelerating intellectual property theft, malicious software exploits and severe business impacts of cybercrime, developers must change.

The shift-left movement in development puts security considerations as an essential part of every development iteration and sprint. Organizations are systematically incorporating security practices throughout their DevOps pipelines to form *DevSecOps*.

DevSecOps is not a single tool or technique. Successful DevSecOps involves a comprehensive framework of practices and tools that ensures organizations consider security in all phases of the development workflow -- nor is one person or department responsible for DevSecOps, which spans development, cybersecurity, QA testing, IT operations and support teams. This article outlines the advantages and challenges of adopting DevSecOps, the elements of a DevSecOps framework throughout the application lifecycle and commonly used tools for each stage of it.

DevSecOps basics

DevSecOps is the addition of security considerations and practices to an organization's CI/CD workflow. DevSecOps does not replace existing development paradigms. Rather, it expands and complements those paradigms by adding a comprehensive layer of security throughout the development cycle.

DevSecOps is a way to solve the problem of developers reserving security checks and testing for the later stages of a project -- often as it nears completion and deployment.

Late-stage security checks lead to two problems. Security flaws can go undetected and make it into the released software, and security issues detected at the end of the software development lifecycle demand considerably more time, resources and money to remediate than those identified early on.

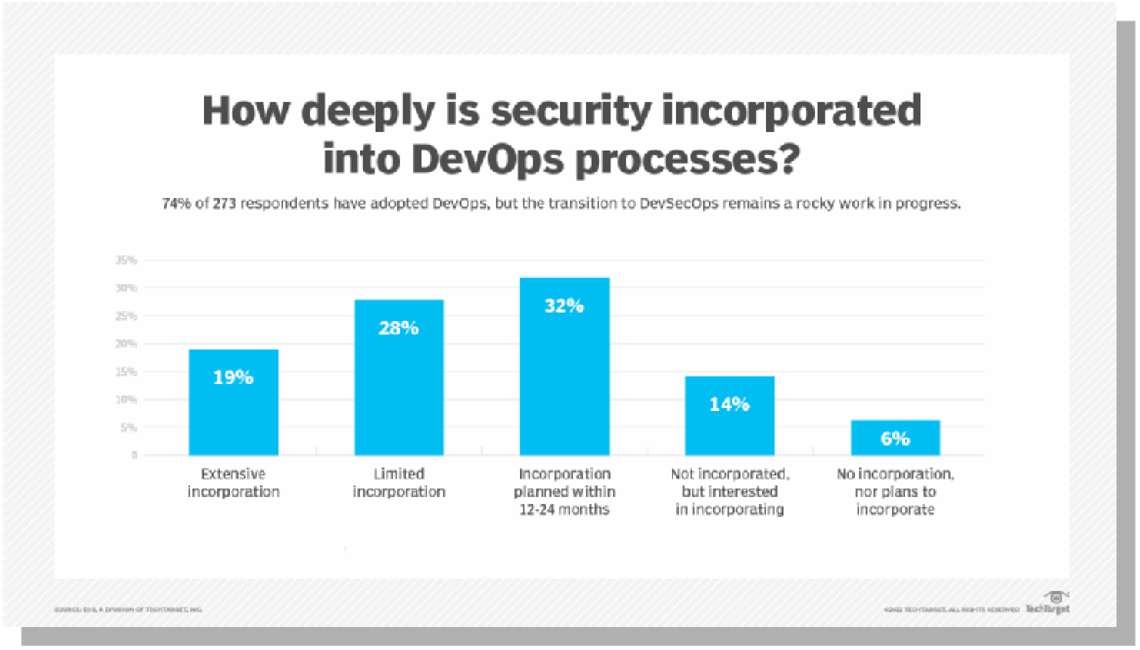
Together, ever-shorter development cycles and the need to manage project costs and mitigate business risks have resulted in a shift-left approach to software security. Shift left signifies moving security checks closer to the beginning of a project, imagining a timeline laid out left to right. The farther left security and other operational concerns move, the more incorporated they are into the design and creation of the product. The goal of DevSecOps is to design, build, test and deploy software in which developers assign as much importance and consideration to security as any other major software feature or functionality.

When implemented well, DevSecOps can yield the following benefits:

- reduced security risk to organizations, users and customers;
- detection of software flaws and vulnerabilities early in the product lifecycle;
- faster remediation of security issues;
- improved communication and collaboration across development, security and operations teams; and
- better overall code quality.

However, incorporating security into an existing Agile development process can pose several drawbacks:

- introduction of changes -- and possible bottlenecks -- to existing workflows;
- additional training and expertise necessary for both the development and security teams; and
- disrupted development workflows due to poor communication or initiatives that aim for zero vulnerabilities.



Create a modern DevSecOps framework

Since DevSecOps is an expansion or enhancement of existing software development practices, it's easiest to consider a DevSecOps framework in terms of the Agile approach to planning, coding, testing, deployment and ongoing operation.

Planning and design

DevSecOps starts early in the development lifecycle, typically in design or planning. For example, security concerns affect goals set during sprint planning. A DevSecOps framework brings security goals into the planning phase in the following ways:

- **Create coding standards and conduct peer reviews.** Security flaws can enter a product when developers write various sections of code in different ways. A team that establishes and enforces a consistent set of coding standards -- and evaluates a codebase against it -- can prevent mistakes that introduce vulnerabilities. Peer code reviews help ensure that software meets coding standards and that developers catch common programming errors.
- **Use security plugins for integrated development environments (IDEs).** IDEs are extensible platforms that often accept security plugins that check code for potential vulnerabilities, in the same way that the IDE would point out missing punctuation or syntax errors. IDE-based security checks offer developers static code analysis before they make any commits to a repository.
- **Conduct threat modeling.** Threat modeling encourages developers to approach an application as a hacker would. They should consider potential abuses of the application, ways to prevent those abuses and how to prioritize those preventative actions in the application design goals. Common threat modeling methodologies include STRIDE, DREAD and OWASP. Multiple threat models can be combined to strengthen design security.

Coding and code management

DevSecOps doesn't stop at security-minded coding standards and peer reviews. Put security at the center of codebase management and maintenance to avoid introducing vulnerabilities. Common techniques to ensure security in code commits and management include the following:

- **Manage security in dependencies.** External libraries, open source code and reused modules are common in software development. However, external code might not include the same security standards and precautions as internal work. Developers should review the security of all dependencies, verify that they are authentic and delivered securely, and ensure that the latest versions are used for each project.
- **Scan code and repositories.** Repository scanning tools can perform static analysis of code committed to repositories before build execution, checking for vulnerabilities, hardcoded credentials and other common oversights. Vulnerability testing and other static testing are vital for code security. Repository scanning adds safety for larger teams in which many developers access the same repository.
- **Secure the development pipeline.** Upon detecting a security issue, a comprehensive DevSecOps framework might adjust the development workflow to prevent code commits to default or trunk branches until the issue is remediated. An attacker could also compromise the pipeline itself by introducing malicious code or stealing credentials. Consequently, organizations should implement and review security controls within their development pipelines.

Testing

Testing should detect application flaws and security issues. Automation and orchestration in build, test and release pipelines should include running security tools when code is deployed for testing -- for example, checking for vulnerabilities during unit testing. Common test considerations for a DevSecOps framework include the following:

- **Integrate dynamic application security testing (DAST).** DAST and penetration testing have historically been the last step in the development cycle. DevSecOps adherents should add DAST, pen testing and other types of dynamic vulnerability testing to the build's test regimen within the pipeline. Full DAST and other dynamic vulnerability testing, such as security acceptance testing, can be time-consuming, but lighter test regimens are an option that can yield faster results, while identifying issues missed in static testing.
- **Secure the infrastructure.** DevSecOps security considerations should go beyond the application itself to include the deployment environment, whether local or cloud infrastructure. Consider implementing policy-driven VMs, containers and Kubernetes clusters. Tools such as Microsoft Azure Policy and AWS Organizations enforce security-driven policies on cloud infrastructures. Infrastructure as code is one way to build standard and known-good application environments.

Deployment and operation

Security concerns do not end with a successful build. DevSecOps practices carry forward into deployment and operational environments with the following:

- **Configuration management.** The infrastructure environment created to host the application must be stable. Any attempt to change an established infrastructure configuration could be malicious activity. Tools that monitor and enforce an infrastructure configuration should be a central element of an organization's DevSecOps framework. Cloud providers offer these tools as well, such as Microsoft Defender for Cloud and Microsoft Sentinel.
- **Intrusion detection and behavioral analytics.** Analytics tools, including intrusion detection and prevention systems, establish baselines in traffic patterns and performance and then look for anomalies indicative of suspicious or malicious activity in the workload or network. Such tools are well established and should be embraced by organizations adopting DevSecOps.
- **Ongoing security testing.** DAST, pen testing and other types of security testing should not end with the testing phase of the application's lifecycle. Conduct testing, such as port scanning and fuzz testing, routinely and whenever the team suspects the code has new weaknesses. For example, if a newly discovered bug appears in a processor's command set, the operations or security support administrators should act. When testing confirms potential vulnerabilities, they can develop and deploy new patches.
- **Alerting and reporting.** Security tools and policies must be paired with comprehensive alerting and reporting. Developers and project stakeholders

should receive actionable intelligence from operations in a way that supports prompt issue identification and remediation.

- **Post-mortems.** Even with the best security efforts, an organization is likely to eventually experience security issues in its application or infrastructure. When a security incident occurs, it's important to conduct blameless post-mortems. Teams should work to identify and remediate the issue and then use the experience to tune future development and operational efforts to prevent subsequent issues.

DevSecOps tools

Organizations shifting to DevSecOps should evaluate their toolchain -- and each tool within it -- for security tasks. Some organizations already use DevSecOps-ready tools, while others will need to update or replace tools. While the list of potential DevSecOps tools is too large to include in full here, the following are some common examples of suitable tools for each major phase of the DevSecOps pipeline:

Planning and design. Team discussions and collaboration on security considerations highlight potential oversights and risks for the project and workflow. Common tools used for planning include issue-tracking and management tools, like Atlassian Jira, and communication tools, like Slack.

Threat modeling is a major planning issue under DevSecOps. Specialized tools, like IriusRisk, CAIRIS, Kenna.VM, Microsoft Threat Modeling Tool, SD Elements, securiCAD, Tutamantic, Threagile, ThreatModeler and OWASP Threat Dragon, provide threat modeling capabilities.

Coding and code management. Developers prepare secure code using static code analysis, IDE plugins like pre-commit hooks and repository scans. Code review tools include PMD, Checkstyle, Gerrit, Phabricator, SpotBugs and Find Security Bugs. When choosing a code review tool, select one designed for the project's programming language and IDE or toolchain interoperability.

DevSecOps security practices in the build phase include software component analysis, static application software testing and unit testing that analyzes the new code, as well as any dependencies. Common tools for build analysis include SonarQube, SourceClear, OWASP Dependency-Check, Retire.js, Snyk and Checkmarx.

Testing. The test phase is focused on using DAST to identify vulnerabilities associated with application operations, such as user authentication, authorization, SQL injection and API endpoints. DAST tools include Invicti (formerly Netsparker), Astra Pentest, Acunetix, PortSwigger, Detectify, Rapid7, Mister Scanner, AppScan and AppCheck.

General DevSecOps testing tools that address attack proxies and network protocol fuzzing include Boofuzz, OWASP Zed Attack Proxy, BDD-Security, IBM AppScan, Gauntlt, JBroFuzz and Arachni. Choose tools based on their suitability for the application type and tests needed and their interoperability with other tools in use.

Deployment and operation. A DevSecOps release candidate should be securely coded, build-checked and thoroughly tested. Next, DevSecOps teams establish a hardened operating environment for the release. They set up access control, network firewall access and secrets management. Change and configuration management tools are central to a DevSecOps model at the deployment stage. Common configuration management tools include Red Hat Ansible, Chef, Puppet, Salt, HashiCorp Terraform and Docker.

DevSecOps teams then focus on situations in the live runtime environment. Differences between testing and production environments should be identified and studied carefully, as they are often a sign of security issues. Runtime performance tools include Osquery, Falco and Tripwire.

In addition, teams use chaos engineering tools, like Chaos Monkey and Gremlin, to evaluate a deployment for -- perhaps untested -- faults, such as server crashes, drive failures and network connectivity issues. The aim is for the deployment to either survive the disruption or fail gracefully.

Feedback loops are core to DevSecOps success. Tools are useless unless the results they produce are cycled back into the development process. Take advantage of reporting and analytics across the toolchain to evaluate the security status of the current release, and use that insight to improve the next development cycle.

Develop a DevOps branching strategy to promote efficiency

Stephen Bigelow, Senior Technology Editor

Version control is key to any DevOps pipeline. Software evolves quickly as numerous developers work on frequent iterations. Repositories demand careful management to track and manage code changes properly. Otherwise, the project spirals out of control as developers struggle to find the right components for a build.

But DevOps is hardly a straight-line process. The value of DevOps and other CI/CD paradigms lies in experimentation, rapid changes, trial and error, and efficient testing -- tasks that can take projects in exciting new directions. Each varied task presents developers with a new branch in the workflow -- and new complexity to repository management.

What is branching?

Just as a tree spreads into an array of individual branches, each leading to a unique leaf, a software project splits and proliferates into myriad different versions as it develops.

Branching strategies help developers manage varied tasks, keep projects on track and enable different IT teams to develop a single project in different directions simultaneously.

Some developers might work on bug fixes, others might work on adding new features, and some might experiment with creative new ideas for changes and enhancements. Each effort represents a potential software branch, which developers can then test, evaluate and merge into the main ongoing project -- or discontinue and discard.

Because branches enable different work to occur on the same project simultaneously, effective branching can bring efficiency to the DevOps process.

Because branches enable different work to occur on the same project simultaneously, effective branching can bring efficiency to the DevOps process.

How does branching work?

A *branch* is a point of divergence as well as a copy of the codebase at a given point in time. Developers or IT teams can work with branches to accomplish necessary tasks, such as bug fixes or new features, without affecting other branches or the main codebase. Developers can then merge and consolidate the branches to enable collaboration. This ensures the branches never diverge too far from the main codebase.

Organizations can establish branches for any purpose, but there are generally five major use cases for branching:

- **Ordinary development.** Typical code development occurs within a main or regular development branch. This is the day-to-day work of building code in accordance with requirements and aligning it with the established roadmap, such as integrating expected features, user stories and testing. Branching strategies that support ordinary development focus on developers, testing and deployment coordination.
- **Fixes.** Fixes can be addressed as ordinary development or as unique branches. This enables some developers to continue working while others fix and optimize the software or otherwise deal with trouble tickets or reported issues. Fix branches focus on merging and testing.
- **Unusual changes.** Changes occur for many reasons, such as user or stakeholder requests, changes in the competitive landscape, or new enabling technologies -- for example, the release of a new underlying platform or engine, such as Unity. These changes might be small or large, and often demand efforts outside the ordinary development cadence. A change branch and workflow cater to these events.
- **Experimentation.** Iterative development creates the opportunity to experiment without committing changes in the product roadmap. For example, developers or stakeholders can offer up creative new ideas, while developers test the performance and integrate new libraries or frameworks. While changes are sometimes folded into their own category, experiments can be used as unique branches for developers who want to try things that might not make it into production.

- **Emergencies.** This is a bug fix intended to address a serious incident or issue that cannot be dealt with through ordinary development or fixes. Branching strategies that accommodate emergencies must enable developers to move a change to release while paying attention to ordinary development.

Branching strategy considerations

A branching strategy is the way a development team creates and uses branches in conjunction with a version control system and repository. As a minimum, a sound branching strategy should do the following:

- Support collaboration by enabling parallel development so that multiple developers can work on different project tasks simultaneously.
- Support version control by complying with prevailing repository and version control systems.
- Support the existing workflow from production or release.
- Optimize the development workflow and enable faster release cycles.
- Integrate with existing development tools and practices.

Selecting a branching strategy is a crucial decision for a software project. There are numerous strategies and variations with unique tradeoffs, and different projects might adopt different branching strategies. When creating a strategy, project managers must consider factors such as the following:

- user and stakeholder requirements for updates and releases;
- the development method or paradigm;

- the development tool set; and
- the project's scale.

Ultimately, any branching strategy should align with the existing CI/CD pipeline -- not the other way around. A strategy that does not fit with the development pipeline can be difficult to implement, and can disrupt and delay development rather than enhance the effort.

Branching strategies

Businesses can adapt and tailor branching strategies to meet a project's unique needs. There are five generic strategy types:

1. **Trunk and release branching.** Release branching creates a branch for the desired release candidate. Developers create and maintain the main code line or trunk branch, and then create a branch to release or deploy to production. Release branching requires developers to apply other branches, such as changes or fixes, to both the release branch and the main code line.
2. **Feature branching.** Feature branching creates a branch to implement a new feature or user story in the project. When the feature branch merges to the project, it adds the new feature. Flags are often used with feature branches to enable or disable the feature while it's in development -- or it's disabled so that users don't see or use the feature until it's ready.
3. **Change and development branching.** Sometimes associated with feature branching, change or development branching represents a longer-lived branch

that contains changes and enhancements. Change and development branches are not the result of defects or the addition of new features, which have their own respective branches. Change and development branches can receive input from numerous developers and must undergo extensive testing and validation.

4. **Fix branching.** Fix branching implements bug fixes or optimizations. A fix branch represents a longer-lived, broader and more comprehensive fix for low- and medium-priority issues -- or a short-lived emergency hotfix for critical issues, such as unexpected server instability. An organization might use two different fix branching types for regular fixes and hotfixes.
5. **Task branching.** Task branching addresses development by envisioning every fix, change or feature as a task, which an issue tracking platform, such as Jira, can follow and reference. Each issue corresponds to its own branch. By coupling tasks with issues, developers and project managers can readily see which code addresses which issue.

These branching strategies can be implemented using several popular branching flow paradigms, such as the following:

GitFlow

GitFlow first appeared in 2010 to enable long-term trunk and development branches -- though the strategy supports every branching strategy type. Teams perform development work, such as new features or regular bug fixes, in development branches. Merges only occur when the developers are satisfied with the development branch. GitFlow also handles emergency bug fixes or hotfix branches.

While the GitFlow strategy proved popular, its support for large numbers of long-term branches makes branch maintenance and merges problematic. Today's GitFlow users must focus on short-term feature branches and frequent merges to reduce potential limitations. Many organizations have stepped away from GitFlow in favor of other strategies.

GitHub flow

GitHub flow, an alternative to GitFlow, addresses every change as a feature branch. Developers use GitHub flow to create new feature branches from the trunk, and can test and validate feature branches before or after a merge. Developers can then merge feature branches back into the trunk branch in a releasable form. The trunk can generate a release branch for additional release preparation, such as packaging into a container image file. Consequently, this is a simpler approach for branch maintenance and merges.

One crucial limitation with GitHub flow is trunk vulnerability -- a bad merge can leave the trunk undeployable. Trunk protection measures, such as backups prior to merge, are essential. Neither GitFlow nor GitHub flow support continuous integration as smoothly as development teams might demand.

GitLab flow

GitLab flow is a Git-based strategy that relies on clearly defined policies and practices to combine feature branching with an issue tracking system. Thus, every branch and its associated development work can be traced to a specific issue -- whether it's a bug, a new feature or an emergency issue -- with the issue tracking system.

GitLab flow merges all features and fixes to the trunk branch, which generates a stable branch and a production branch. When a trunk branch is ready for deployment, it can merge into the production and release branch. Established guidelines and best practices manage this process.

Trunk-based development

Trunk-based development (TBD) is a variation of GitHub flow where developers rely on a releasable trunk and draw branches from the trunk. TBD requires developers to commit and integrate changes daily. This keeps branches short-lived, means changes are small with less effect and emphasizes frequent collaboration between developers. TBD supports many kinds of branches, but generally aligns well with CI/CD requirements.

Longer-term efforts can use mechanisms such as feature flags to toggle work in progress on or off. In effect, the developer creates a switch in the code where in one condition, the software works as normal, and in the other, the state executes the new code and disables the old. This enables branches to merge daily with the new features disabled until the work is complete. Once tested and validated, IT teams can remove and clean up the old code.

Merging and validation

Unlike the tree metaphor, developers frequently select and merge parallel software development branches to form a project's main code line. This merge keeps a project on track as a single cohesive arc. However, merges can be difficult. Any branching strategy must consider the merge's speed and effectiveness. Larger and longer-lasting tasks can make merges more difficult and highlight the benefits of smaller, shorter-lived branches. Branching strategies and tools can play a huge role in a merge's success or failure.

Validation follows the merge and is critical to test the merged project's main line. This demands careful CI implementation and automated testing techniques. Branch testing validates each branch before a merge to reduce problems in the final merge, but there is no substitute for testing post-merge.

How CI/CD pipelines are putting enterprise networks at risk

Rob Wright, News Director

LAS VEGAS -- When it comes to the software supply chain, organizations should be concerned about more than just external threats like poisoned open source dependencies.

During a Black Hat USA 2022 session Wednesday, cybersecurity vendor NCC Group presented its findings from studying continuous integration (CI) and continuous delivery (CD) software pipelines at many organizations during the past five years. Enterprises have embraced CI/CD pipelines in recent years to increase speed and efficiency for software development.

While supply chain security has become a concern for enterprises amid growing reports of poisoned software libraries and compromised repositories, NCC Group researchers argued that the pipelines themselves pose significant risk to organizations.

During the Black Hat presentation, NCC Group's Iain Smart, principal security consultant, and Viktor Gazdag, managing security consultant, explained how they were able to compromise virtually every CI/CD pipeline they've tested in the last five years, with some rare exceptions. They described how gaining unauthorized access to pipelines, which are often not properly segmented and hardened, could allow threat actors to turn them into "remote code execution (RCE) as a service" threats that deliver signed malware instead of enterprise applications.

In an interview with SearchSecurity, Smart said CI/CD pipelines are a dangerous attack surface that has gone "unrealized" for a long time. But that might not last, as threat actors continue to probe for weak spots in enterprise environments, particularly around the software supply chain.

"I think a lot of this comes down to a lack of threat modeling, which is a formal way of saying you're just not thinking about what you're deploying," he said.

CI/CD pipeline attacks

Smart and Gazdag detailed several customer engagements where they were tasked with compromising a CI/CD pipeline and then breaking out of the pipeline to gain access to a company's primary infrastructure.

Smart described one engagement where the client granted NCC Group researchers a set of developer credentials, which lacked admin privileges, to see if they could break out of the pipeline. While they had access to just a single Bitbucket repository, they were also able to modify which dependencies were required for the pipeline to build an application.

Smart's team added its own malicious dependency to the mix, and when the pipeline connected to it, NCC Group was granted access to a Jenkins runner. While the runner had a limited environment, Smart said NCC Group discovered a private SSH key lurking in it for unknown reasons. With the key, the researchers were able to gain access to all the organization's Jenkins control nodes and obtain all secrets -- which NCC Group described as any nonpublic data, such as credentials and API keys -- across all the customer's Jenkins projects.

"In this case, it turned out we had full access to deploy anything we liked into the customer's production workload," including malware, Smart said.

In another red team engagement, NCC Group compromised a customer that Smart said was a "relatively hard target." The researchers weren't able to get into the customer's CI/CD pipelines through their usual methods, but they successfully phished a developer and used the developer's credentials to access a wide range of repositories.

Even though the customer had hardened its development environment and limited permissions and access, the red team was able to modify some of the pipelines' functions and grab secrets that had not been filtered out. Eventually, the researchers found a single pipeline that had domain admin access, which gave the red team significant control.

"This shouldn't need to be said in 2022 -- don't run everything as domain admin, please," Smart said. "It's just not a good idea."

Gazdag and Smart discussed other customer engagements where the research team was able to break out of a compromised pipeline into not just the full product environment, but also the customer's cloud and on-premises infrastructure. In one case involving a particularly well-defended customer with extremely limited developer access, Smart was still able to obtain the production API keys and modify the company's AWS environment.

Smart said the customer was initially in denial that such an attack was possible and was so confident that it allowed him to execute the attack. "Two redacted screenshots later, we were on an incident response call," he said.

He also said that code signing software in production is a good practice, but it's not a silver bullet to prevent malicious activity. In fact, it could be used by threat actors to create an RCE as a service.

"A lot of people think, 'Well, I sign everything at the end of my pipeline, so I'm secure,' and they're kind of correct," Smart said. "But if an attacker has compromised your pipeline before that stage, then congratulations -- you've just programmatically signed my malware, and now everyone is going to trust it more."

In this e-guide

Lessons learned

Gazdag emphasized that none of the CI/CD pipeline engagements NCC Group worked on involved any exotic exploits or novel techniques. "None of these problems are new, and we didn't use any zero days," he told the session audience.

In an interview with SearchSecurity, Gazdag said there were a few customer engagements where the research team couldn't break into the pipelines. "When we couldn't compromise a pipeline, it's because the company had the security controls that we either recommend or were previously deployed," he said.

Those controls include relatively simple steps like segmenting of production environments, applying principle of least privilege, using role-based access control, and setting up alerts and proper logging for the pipelines. Gazdag and Smart also encouraged organizations to limit pipeline access to external dependencies that have been validated.

Smart told SearchSecurity that many of the customers NCC Group worked with didn't fully grasp how big of an attack surface their CI/CD pipelines were -- and how much damage could be done if they were compromised.

"These pipelines are, by their very nature, privileged. The more automation you add in, the more privileges you need to give your automation," he said. "If you've got a box that can automatically do things for you, and someone compromises the box, then that attacker has all of the permissions that the box had."

Scrum vs. DevOps: Are they intertwined or redundant?

Clive Longbottom, Independent Commentator and ITC Industry Analyst

As software development becomes more complex and iterative, rather than monolithic, IT ops teams need a methodology to help manage CI/CD successfully.

Approaches such as Agile have been successful, but DevOps has become more popular over the past few years. One approach that has garnered a reasonable following, however, is Scrum, which is often associated with an Agile approach.

What is Scrum?

Named after a move in the British ball game Rugby, Scrum first appeared in a paper written by Hirotaka Takeuchi and Ikujiro Nonaka in 1986. The name choice emphasizes that Scrum is a team-focused approach, with Scrum team members called Scrum Masters.

Agile focuses on four core values:

1. Individuals and interactions -- with a focus on processes and tools
2. Working software -- in preference to comprehensive documentation
3. Customer collaboration -- rather than contract negotiation

4. Response to change -- rather than follow a restrictive plan

Scrum applies a framework around Agile's core values while also ensuring that other areas of an IT environment are not discarded completely.
The idea behind Scrum is to take a problem -- the customer's request -- and break it down into discrete, smaller tasks that individuals or small IT ops teams can solve. Business and customer problems make up a product backlog that is then analyzed through a rapid sprint planning session and broken down into a sprint backlog.

Scrum teams hold daily scrums to break backlogs down into tasks. At the next daily scrum, teams review the tasks to see if the required progress is complete. If so, the code becomes available as an incremental update and a sprint review carries the feedback to sprint and product backlogs. Decisions are then made on what new tasks go into the sprint backlog. As the sprint backlog depletes, the product backlog can bring in new tasks, or remain empty.

Daily scrum meetings should focus on progress and needs, rather than technical issues. Customers can -- and should -- be involved to provide direct and timely input for any requirement changes.

The DevOps approach

Much of this might sound familiar to DevOps practitioners, though. DevOps breaks down large requirements into smaller packages of work and moves them to provisioned operational software -- a few steps further than Scrum does.

However, many DevOps developers might not look at their approach with the same granularity that Scrum does. In many cases, DevOps is a process flow aid: It moves code through the development process, puts in place a provisioning capability to move code through to the operations environment and enables feedback loops between the operations and development environments.

DevOps also tends to pay less attention to the customer. Many DevOps practitioners request requirement statements from the customer, and then use DevOps to fulfill that, which is more similar to Waterfall-style approaches.

DevOps vs. Scrum

DevOps identifies automation opportunities to improve overall efficiencies in code development and provisioning through reducing friction in processes: It does not work as an end-to-end environment to manage everything from customer requirement to running code.

With Scrum, IT ops teams must learn a new way to approach problems, with a focus on project methodologies rather than code mechanics. The difference between Scrum and DevOps is Scrum defines how to solve problems, whereas DevOps provides the means to solve those problems.

Therefore, DevOps and Scrum can come together. Scrum focuses on development and provides a deep but granular approach to manage rapid development. It does not move code into operations -- it indicates that such code is ready for provisioning.

Meanwhile, DevOps spans the two environments but generally does not provide the task-level capabilities of Scrum.

Note the "generally" here. There is no reason why DevOps tools cannot come with Scrum capabilities built in. For example, Microsoft Azure Boards is an environment that brings together GitHub repositories with Agile approaches, such as Scrum or Kanban, to provide a console to monitor progress, known bugs, upcoming features and backlogs. In addition, Atlassian provides high levels of automation in DevOps environments through its Open DevOps and Jira product line that encompasses Agile approaches within a highly functional DevOps tool set.

Scrum -- or any Agile approach -- and DevOps are not mutually exclusive. Do not bring the two in and run them in separate environments. DevOps practitioners should look to how Scrum can be interwoven within DevOps tools. This leads to better feedback loops at the early stages of the DevOps process, maintains constant interactions with customers and ties downstream feedback loops from the DevOps system into Scrum meetings and discussions.

GitHub targets vulnerable open source components

Cliff Saran, Managing Editor

GitHub has introduced an automated alert mechanism to enable developers to address vulnerabilities in the open source components their code uses.

According to GitHub, the new feature, called Dependabot alert for vulnerable GitHub Actions, will make it easier for developers to stay up to date and fix security vulnerabilities using their Actions workflows.

Vulnerabilities such as Log4j have shone a spotlight on the weakness of open source security, and US president Joe Biden has made software security a national priority. His executive order on cyber security requires that only companies that use secure software development lifecycle practices and meet specific federal security guidance will be able to sell to the federal government.

The strength of open source code is that external code modules can be pulled into a project from a public repository such as GitHub. This makes it easy for developers to incorporate functionality without having to write all the code themselves. The open source modules are maintained by third-party developers.

However, as Computer Weekly has previously reported, if a security risk is discovered in the open source module, projects that depend on this module are also at risk. In many cases, developers whose code requires such modules may not be aware that the open source code they have incorporated into their own project has a security risk.

This is the situation GitHub hopes to address with Dependabot alerts for vulnerable GitHub Actions.

In a blog post discussing Dependabot alerts for vulnerable GitHub Actions, Kate Catlin, senior product manager at GitHub, and Brittany O'Shea, an author on the GitHub blog, said the Alerts will be powered by the GitHub Advisory Database.

"When a security vulnerability is reported in an action, our team of security researchers will create an advisory to document the vulnerability, which will trigger an alert to impacted repositories," they wrote.

At the time of writing, the GitHub Advisory Database has 8,543 advisories that have been reviewed, 1,560 of these have been classified as "critical". But, to demonstrate the scale of the problem facing the open source community, the database shows that there are over 173,000 vulnerabilities in GitHub that have not been reviewed.

There is general consensus that global collaboration is needed to keep open source code secure. In January this year, a number of major tech firms, including Google and IBM, participated in the White House Open Source Software Security Summit.

To coincide with the summit, Kent Walker, president of global affairs at Google and Alphabet, posted a blog discussing the need to secure open source code effectively.

"Growing reliance on open source means it's time for industry and government to come together to establish baseline standards for security, maintenance, provenance and testing – to ensure national infrastructure and other important systems can rely on open source projects," he wrote.

Jamie Thomas, enterprise security executive at IBM, who also attended the summit, said: "Today's meeting made clear that government and industry can work together to improve security practices for open source. We can start by encouraging widespread adoption of open and sensible security standards, identifying critical open source assets that should meet the most rigorous security requirements, and promoting a collaborative national effort to expand skills training and education in open source security and reward developers who make important strides in the field."

Potentially, Dependabot alerts for vulnerable Actions can be linked into continuous integration and deployment (CI/CD) processes to enable developer teams to prioritise developer work and address security issues more quickly.

In this e-guide

Getting more CW+ exclusive content

As a CW+ member, you have access to TechTarget's entire portfolio of 80+ websites. CW+ access directs you to previously unavailable "platinum members-only resources" that are guaranteed to save you the time and effort of having to track such premium content down on your own, ultimately helping you to solve your toughest IT challenges more effectively—and faster—than ever before.

Take full advantage of your membership by visiting http://pro.techtarget.com/CWLP

Images; Fotalia