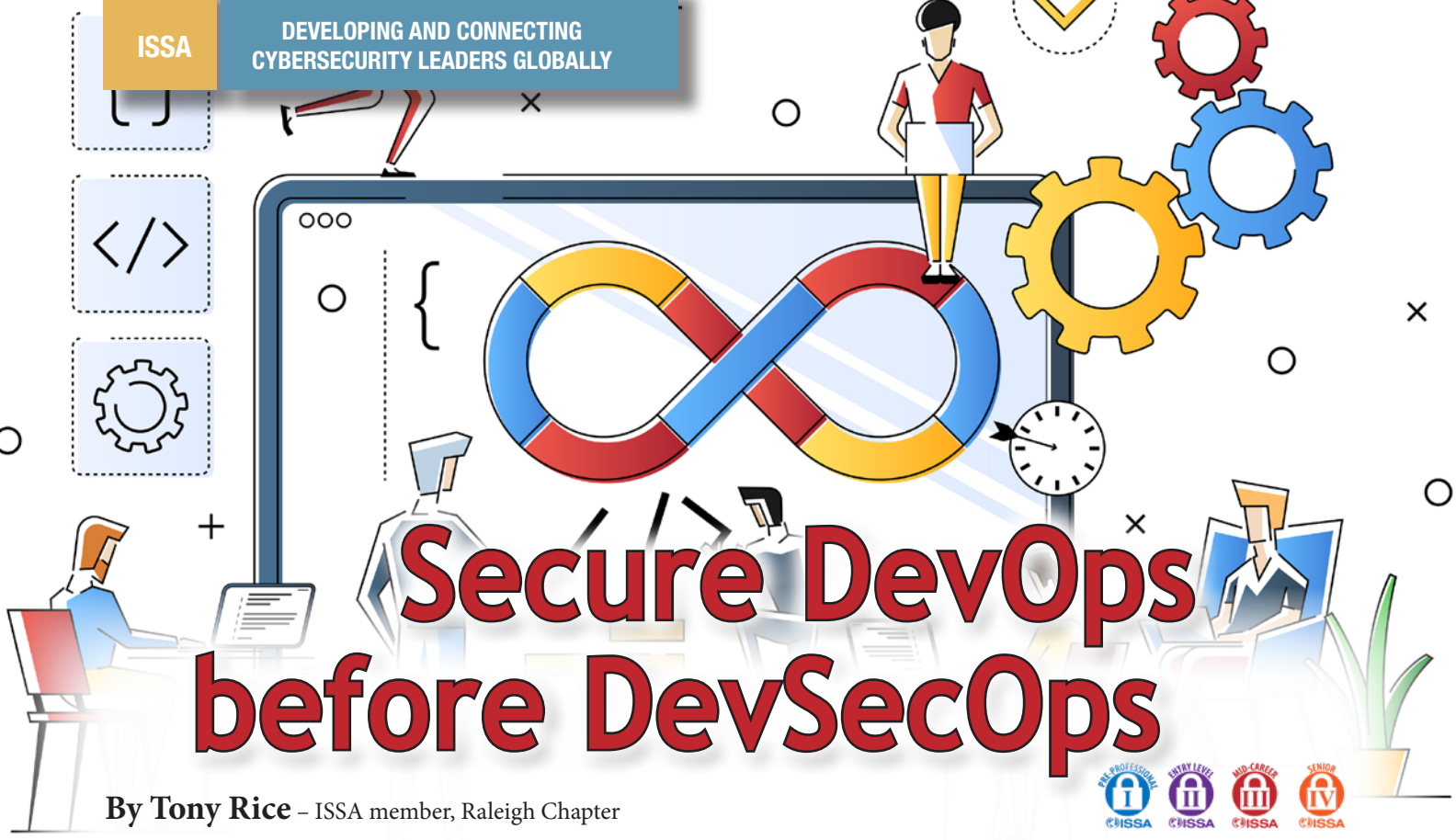

Secure DevOps before DevSecOps

By Tony Rice – ISSA member, Raleigh Chapter

This article discusses the opportunities DevSecOps offers to stand up infrastructure in a consistent secure way as well as move discovery of security flaws earlier and more often in the software development life cycle, with a back-to-basics view of securing access to these resources first.

Abstract

DevOps offers a world of possibilities to automate security checks to find vulnerabilities earlier, before they are deployed and when they are least expensive to fix. It also offers opportunities to more closely partner with developers and operations personnel. But before we turn DevOps into DevSecOps, the build pipelines enabling this automation must be locked down to prevent information about the vulnerabilities found from being leaked the way credentials are leaking today.

When security professionals hear the term "DevOps," a rainbow of automated security checks come to mind. Visions of vulnerabilities revealed early and often dance in their heads, quickly morphing DevOps into DevSecOps.

We dream of all that agile development methodologies offer to more closely partner with developers and operations:

- Helping developers stop the flow of OWASP Top 10[1] examples into the codebase.
- Guiding operations personnel to the consistency that infrastructure-as-code provides.
- Putting an end to the vulnerability whack-a-mole that artisanal handcrafted (and never again patched) instances bring.

Tool vendors stoke these fires with talk of how easily their tools with prepackaged test suites integrate into existing pipelines. It's even easier if you migrate everything over to their flavor of DevOps. And the dashboards, so many dashboards, each providing evidence of the progress being made to leadership eager to know how the investment is paying off.

Can DevSecOps finally fulfill the promise of moving vulnerability discovery to the left of development process where it is cheaper and easier? As time between deliveries shrinks from months to hours, DevSecOps must enable these things and more.

Security professional, heal thyself

Before rushing off to integrate the dynamic application security test suite and code scanner—the ones that cost so much yet get used so little—into a maturing DevOps pipeline, consider the threat landscape that DevOps itself introduces.

Functional and other tests in a DevOps pipeline describe where the product doesn't work. Security tests describe where the product is vulnerable. This is sensitive information that

1 OWASP Top 10, 2017 https://www.owasp.org/index.php/Top_10-2017_Top_10.

should be generated, stored, and transmitted in an appropriately secured environment.

Carelessly implemented Dev[Sec]Ops can broaden an already vulnerable attack surface. Your DevOps pipelines weren't built in a day and neither should your DevSecOps. So where to start?

Begin by securing the DevOps pipeline itself, using the same principals you've been preaching to the development and operation teams. A basic continuous integration/continuous deployment (CI/CD) pipeline is made of segments implemented by individual tools, each with its own set of credentials:

- **Source code repository**: to track changes as the development team moves the product forward such as Git, Bitbucket, Subversion, SourceForge, CodeCommit, Cloud Source, etc.[2]
- **Artifact repository**: to store the fruits of those builds such as Artifactory, Archiva, Nexus, etc.[3]
- **Deployment host**: somewhere to deploy the product such as cloud compute services like Amazon Web Services (AWS) EC2, Azure Virtual Machine, or Google Cloud Platform (GCP) Compute, or maybe serverless services like AWS Lambda, Azure Functions, or GCP FAAS; containers; or an even an on-premise host.
- **Automation platform**: to keep things moving through the pipeline when they should and stop their progress when they shouldn't such as Jenkins, Bamboo, CircleCI, GitLab, CodeBuild, etc.[4]

2 Neil Chue Hong, "Choosing a Repository for Your Software Project," Software Sustainability Institute, https://www.software.ac.uk/choosing-repository-your-software-project.

3 Carlos Sanchez, "Using Repository Managers," DZone Refcard #181, https://dzone.com/refcardz/binary-repository-management.

4 R. Vaasanthi, et al, "Comparative Study of DevOps Build Automation Tools," International Journal of Computer Applications, July 2017, https://www.ijcaonline.org/archives/volume170/number7/vaasanthi-2017-ijca-914908.pdf.

Many environments also include:

- **Communication tools** such as Slack, HipChat, WebEx Teams, etc. that enable developer collaboration and bots to ensure visibility of DevOps results.
- **Workflow and defect management tools** like Jira, Trello, Bugzilla, etc.

Begin by looking at the couplings along that pipeline of tools. Are they secure or is your hard work leaking into the hands of the bad guys? An automation platform like Jenkins is the grand central station of this system and a good place to start securing CI/CD efforts.

If your CI/CD pipeline grew out of a developer experiment that became a critical production resource over time, you might find developer personal userids and passwords throughout its configuration. This can lead to failing pipelines when that person changes the password. It is tough to remember all the places those credentials have been used. You might find that one developer account across each of those tools in the pipeline.

Those person-to-machine credentials should be replaced with machine-to-machine, rotated on a regular basis. Also use access tokens rather than userid and passwords whenever possible. If your tool of choice doesn't support access tokens, that's a good sign to keep looking at other tools.

However, as CI/CD pipelines proliferate, and they will, you'll quickly find that managing all those secrets will become cumbersome and risky.

Stop checking in credentials

You need look no further than files publicly available via GitHub.com for one of the most common problems in DevOps: storing configuration files and source code containing secrets in plain text, in a source-code repository

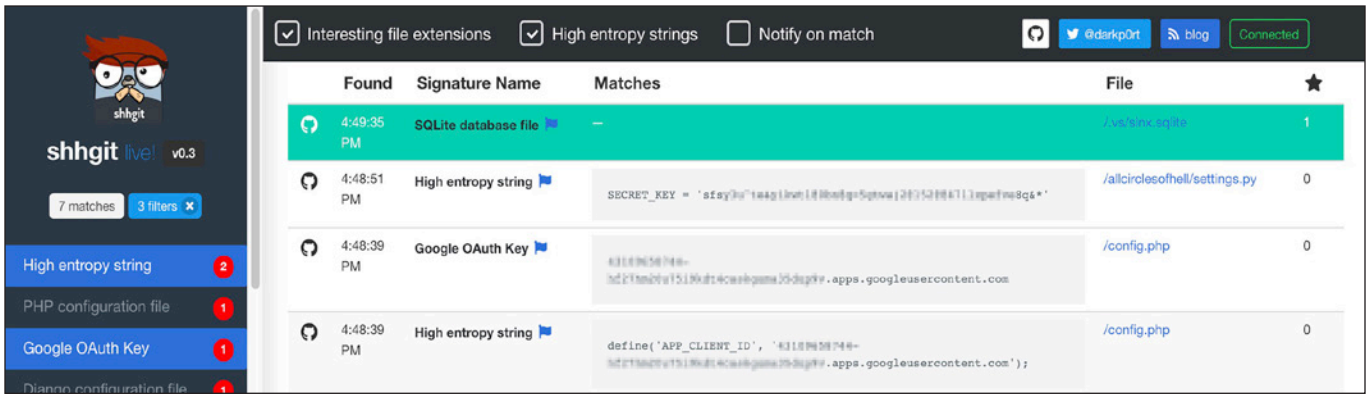

Figure 1 – Shhgit revelations

Attackers don't need to go to the trouble of a man-in-the-middle attack to steal credentials when someone has made it so easy by checking in everything they need to access each tool along the pipeline, even the destination hosts where the application is deployed to.

A code search of json files publicly visible on GitHub.com using the string *aws_access_key_id filename:*.json* produces several thousand results. You can watch people make this mistake every few seconds publicly, in near real time with shhgit.[5] The open source project by Paul Prince flags secrets and sensitive files as they are checked into GitHub in near real time (figure 1).

The Internet is forever and so is GitHub.

A similar search in commit comments containing the phrase *removing password* brings up page after page of examples of developers who caught and corrected their mistake.[6] But remember the Internet is forever. That leaked secret remains in the repository's history for all to see.

If a secret is exposed like this, change the password, revoke the token. Better yet, prevent this easy-to-make mistake in your DevOps pipeline by removing the possibility of using hard-coded credentials at all.

Now stop managing credentials

A secrets broker like HashiCorp's Vault, Confidant, Keywhiz, etc., or similar services offered by cloud service providers like AWS, Docker, or Google Cloud offer a different approach to help deal with the growing problem of secrets sprawl as DevOps becomes the norm. These brokers not only reduce the attack surface by implementing secrets programmatically via short-lived tokens, they also take care of lifecycle management, freeing you from the time-consuming and error-prone chore of rotating passwords or keys by hand.[7]

Secrets brokers like Vault provide an encrypted, single source of truth for secrets instead of spreading them across Jenkins Masters. It natively supports each of the authentication methods used by platforms mentioned here (AWS, Azure, and GCP) as well as some you might need in the future like Kubernetes, GitHub, etc.[8] You can control much of this right from your Jenkins Grand Central Station with plugins that support secrets brokers like Vault.

Role-based access control

You probably already have access control on source code, maybe even enforcing roles to limit interaction between branches and locking down sensitive areas to a trusted set of developers. This protects the source code, but don't forget to provide similar protections to binaries.[9] They are intellectual property as well, not that far removed from source code thanks to decompilers and other reverse engineering tools like the NSA's Ghidra.[10]

No user, neither flesh-and-blood nor DevOps machine-to-machine accounts, should be provided more access than necessary.[11]

Out of the box, most tools, especially artifact repositories like Artifactory, know nothing about your business. Default configurations are often designed to enable getting the tool up and running quickly rather than encouraging good security practices. Your first step with a new CI/CD tool should be configuring access to reflect the development workflow and users and systems that will interact with it.

Artifact repositories should have a minimum of three roles:

- Limited privileges intended for users and automation accounts for fetching artifacts for use in builds and orchestration tasks as read only.
- Limit privileges for destructive functions such as deleting artifacts to a separate administrative role.

5 Paul Prince, "Ah shhgit! Find GitHub Secrets in Realtime", GitHub – https://github.com/eth0izzle/shhgit/.

6 Search results, https://github.com/search?q=removing+password&type=Commits.

7 NIST Special Publication 800-53 (Rev. 4) "Alternative Security Mechanisms (CP-13), https://nvd.nist.gov/800-53/Rev4/control/CP-13.

8 HashiCorp Vault, "Auth Methods," https://www.vaultproject.io/docs/auth.

9 GitHub, "Access Permissions," https://help.github.com/en/articles/access-permissions-on-github.

10 Kelly Sheridan, "NSA Researchers Talk Development, Release of Ghidra SRE Tools," Dark Reading, https://www.darkreading.com/endpoint/nsa-researchers-talk-development-release-of-ghidra-sre-tool/d/d-id/1335536.

11 JFrog, "Configuring Security," https://www.jfrog.com/confluence/display/RTF/Configuring+Security.

- Administrators who are also users should be interacting with these systems with separate accounts to prevent costly mistakes.

Finally, log everything. DevSecOps combines a lot of distinct automation tools together. Things will go wrong and without detailed logs, finding out which tool(s) contributed and how is difficult to impossible.[12]

Now you may DevSecOps

Now that the DevOps pipeline is no longer leaking source code, artifacts, and the credentials needed to delete the whole thing, the pipeline is secure enough to get down to the business of improving the security of the products being built there.

A pattern that has been successfully applied to adding functional and other tests to continuous integration pipelines for years can be just as successfully applied using security checks as well: don't try to boil the ocean. Instead, organize all those tests that have been running through your head by a) the time it takes to run them, and b) their tendency to produce false positives.

Place those quick running, low false-positive ones in the pipeline to run on each code check. These are your security smoke tests, the most basic of sanity tests. They will catch simple problems that only waste developer, QA, and now security personnel time when they aren't stopped early. One way to determine if a test belongs in this frequently run set of tests: If you don't trust it enough to automatically notify developers or open defects, it doesn't belong in that set of very frequently run tests.

Tests that take longer to run like dynamic application security scans or those that produce false positives still provide value but should done periodically, in a separate pipeline, perhaps weekly. For those long-running, less than-reliable tests, carve out those that can be run quickly. The key to maximizing the value DevSecOps provides is continuing to evolve the tests that are run and when. For example, static security scanning is fast but notorious for producing lots of false positives. But some of those tests are quite reliable and possible only in that context, like those that find hard-coded secrets in source code. Carve those high-value, low-noise tests out and move them into the pipeline where they will be run more frequently.

Vendor tools part of, but not the complete solution

Tool vendors talk a lot about the volume of tests available across the tools they sell because it demonstrates the value of investing in their products. But that tool was built to help a broad spectrum of customers with only the most generic knowledge of your products, environment, or customers. Enabling on each option on each test will generate more noise than value, leading developers to ignore the tests.

Start small with free tools like OWASP Zed Attack Proxy (ZAP) for dynamically scanning web application projects.[13] Enable tests selectively. As you find tests that provide value, add them to the DevSecOps pipeline that make sense based on how much assistance the developer, operations engineer, or whoever will be acting on those results must perform. Some tests should be run on every code check-in, some once a week, some only before deployment.

Credentials accidentally leaked to a public source code repository should be revoked and changed. Better yet manage them in a secrets store like Vault.

You can't improve what you don't measure

Now that you've got security tests running, don't let pass/fail be the only metric you gather.

Use those measurements not just to identify problems as they are introduced into the codebase, but as an indicator of where the team needs help. When static code analysis routinely finds credentials and other secrets stored in source code (and checked into the repository), use this as an opportunity to educate those developers on more secure ways to use credentials, maybe extending a secrets broker like Vault into the application as well.

Additional value can be gained from these tests beyond individual results by looking for patterns in consecutive results. Also, when vulnerabilities found in scans are associated to defects, subsequent scans can be used to validate that the fixes submitted by developers actually resolved the vulnerability.[14]

Rushed DevSecOps Is Insecure DevSecOps

DevSecOps offers tremendous opportunity to accelerate the software development life cycle while enabling consistent, constantly improvable deployment of infrastructure. Hastily implemented, or prematurely moving half-baked, even experimental DevOps implementations could expose your source code, infrastructure and customer data.

About the Author

Tony Rice, CISSP, is a DevSecOps architect at Cisco. He regularly speaks on effectively incorporating application security into DevOps and leveraging the data that it generates to not only find vulnerabilities but generate evidence of compliance across a multitude of standards. He may be reached at trice@cisco.com.

12 NIST Special Publication 800-53 (Rev. 4), "Audit Review, Analysis, and reporting (AU-6), https://nvd.nist.gov/800-53/Rev4/control/AU-6.

13 OWASP Zed Attack Proxy Project, https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

14 Center for Internet Security, "CIS Control 3: Continuous Vulnerability Management," https://www.cisecurity.org.