Software Assurance Tips

A product of the Software Assurance Tips Team[1]

Jon Hood

Monday 1st November, 2021

1 Additional Risks to DevSecOps Pipelines

Updated Monday 1st November, 2021

I remain a huge fan of DevSecOps pipelines. The benefit to including automated security scans and checks into the development lifecycle fosters a culture of security. Alerting developers to bad habits and patterns early in the development process is one of the best ways to avoid costly reprogramming efforts in the future. Nevertheless, the upsides of DevSecOps pipelines should not cause someone to overlook the additional risks and attack vectors introduced by implementing such processes.

1.1 Increased Attack Area

Additional software, management hooks, and authenticated entities are needed to set up an automated DevSecOps pipeline. The goal is to secure this pipeline so that a level of confidence and assurance in the products that come out of it can be achieved. Yet the authentication to build servers, scan servers, fuzzing engines, code coverage algorithms, and whatever else is being used to help secure the code each come with a new attack vector. Automation scripts can be harvested to glean credentials. Rules can be added and modified to evade detection. And each additional tool added to the networked automation must be maintained.

Our current SOP for non-DevSecOps environments is to perform scans entirely offline. This prevents the scanning tools themselves, binaries being dynamically analyzed, and scripts being interpreted from phoning home. This is not always a possible step to take in a fully connected DevSecOps environment. Furthermore, it's rare for each development shop to set up individual authenticators for each program in their pipeline. Suppose that one of the projects in an organization give access to an outside entity to change third-party code in a section of their repository. Assuming that this third-party repository is also configured for scanning, any vulnerabilities or misconfigurations of the scan tools can expose code that is not part of that external organization's repository.

The biggest risk of combining each program into a single DevSecOps process is giving the keys of the kingdom—the entirety of every project's code and build process—to the administrators that maintain the DevSecOps environment. Handing off this responsibility instead of keeping it disparate means that a single compromise to your DevSecOps management credentials gives an attacker access to your entire kingdom.

1.2 Accepting the Risk

Are you working on projects that have a high level of secrecy? Is the additional cost of setting up your own pipeline for this particular project worth it so that exposures on a centralized DevSecOps environment do not impact your program's security? Do you trust the tools being used in your agile supply chain? If you're considering a centralized DevSecOps environment for your organization, these are serious questions that must be answered.

Praising the benefits of a centralized DevSecOps pipeline should not be interpreted as overlooking the additional security concerns it can introduce. Consider the code coverage tool, Codecov. Earlier this year, one of their uploading mechanisms was compromised in a discreet way that exposed hundreds of credentials (including this author's).[2] For more than two months, attackers passively intercepted the credentials of Codecov users, including tokens that authenticated to more repositories than just the ones using Codecov. When developers work to build test cases, code coverage statistics, and automated scanning in a DevSecOps environment, it stings a little bit more when the tools that are used to secure the pipeline are the ones that become compromised.

References

[1] Jon Hood, ed. SwATips. https://www.SwATips.com/.

[2] Ax Sharma. Codecov hack aftermath. hundreds breached, many more to follow. Security Report. Apr. 27, 2011. URL: https://securityreport.com/codecov-hack-aftermath-hundreds-breached-many-more-to-follow/ (visited on 11/01/2021).