Security and the DevSecOps Platform

Approaches, Methods, and Tools

WNDRVR

INTRODUCTION

Security spans many kinds of technologies, markets, and devices. It's becoming more common to enable hardware-based security features through use of powerful software and embedded firmware across several platforms, especially PC, Intel® devices, and Arm®-based processors, as well as throughout the various stages of a product's lifecycle.

As development processes shift to find new, creative, innovative ways to protect sensitive functions, hackers and malicious users also find new ways to infiltrate source code and even development environments and code repositories. This continues to happen at all levels, even government-based entities. Developers must now create code with security in mind during all phases of development and operations. This is called DevSecOps.

During the beginning phase of development, a security assessment should be conducted to identify which security features are needed. Here, designers will identify assets and their vulnerabilities, create a list of security implementations to protect the asset from each vulnerability or group of vulnerabilities, and create another list of security-related events that should be logged in the event of a breach or system security failure.

The developer should always consider the many standard security specifications, guides, and reference designs that exist. For this particular article, though, the focus will be on the following specifications:

- Committee on National Security Systems Policy 15 (CNSSP 15)
- Recommendation for Key Management (NIST 800-57)
- Security and Privacy Controls for Federal Information Systems and Organi-zations (NIST 800-53)
- Zero Trust Architecture (NIST 800-207)

TABLE OF CONTENTS

WNDRVR

HOW SECURE SHOULD YOUR SYSTEM BE?

In security, there's often the question of how secure your system should be, or what kind of "security strength" should be used. This strength is defined as a number associated with the amount of work that is required to break a cryptographic algorithm, represented in bits.

As described in NIST 800-57 (Part 1) and CNSSP 15, a security strength of 128 bits is acceptable. It is worth noting, though, that as quantum computing is becoming more practical, this estimate stands to change a great deal in the very near future.

Figure 1 represents a typical U.S. Department of Defense (DoD) DevSecOps environment, showing the user interface, repositories, and flow of data from the dev environments to test, integration, and release. This configuration can be scaled with native cloud or hybrid cloud environments, which ensures that security principles can be included and continuously integrated throughout the course of the project.

A list of some of the key assets of this environment can be derived from the diagram shown above, including repos, dev, and test environments; build tools; connectivity; configuration of each component; persistent storage; and event logs. Once the assets are identified, an assessment of their vulnerabilities must be performed. The relative threat of information security to these assets includes a variety of unauthorized uses, such as access, disclosure, disruption, modification, and destruction.

Figure 2 shows a table highlighting some key areas that might require additional security features to help mitigate these threats.

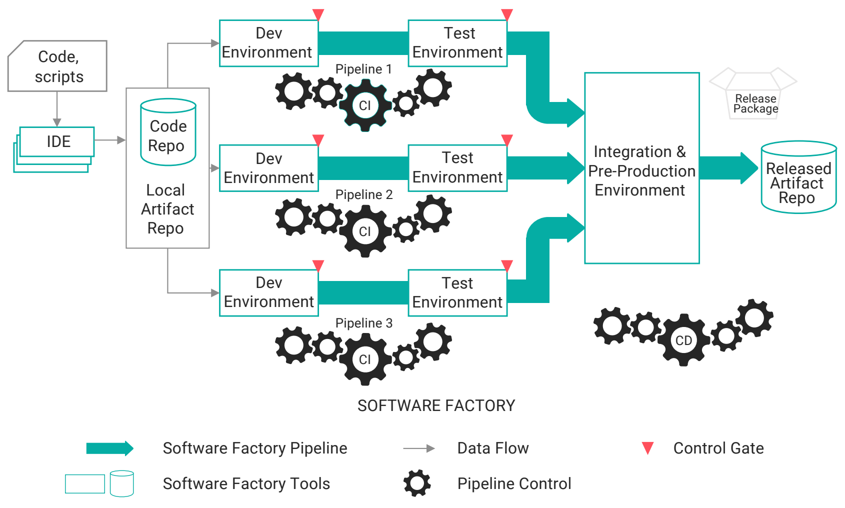

Figure 1. DoD DevSecOps environment

ASSET	UNAUTHORIZED					
	ACCESS	USE	DISCLOSURE	DISRUPTION	MODIFICATION	DESTRUCTION
REPOSITORIES	X		X		X	X
SOFTWARE COMPONENTS		X			X	X
BUILD TOOLS		X			X	X
CONNECTIVITY	X		X	X	X	
CONFIGURATION	X		X		X	X
PERSISTENT STORAGE	X	X	X		X	X
EVENT LOGS	X	X	X		X	X
HSM	X	X	X	X	X	X

Figure 2. Matrix of DevSecOps assets and associated threats

DEVSECOPS TECHNOLOGIES AND PRIVILEGES

It's important to consider which technology should be used to protect these vulnerabilities, as well as the capabilities associated with that technology. This starts with the hardware security module (HSM), which is the root of trust for the DevSecOps environment. The HSM is a purpose-built, cryptographic provider including code signing for both symmetric and asymmetric keys and is tightly controlled, requiring two-person authorization for any changes made. It's important to have certificate revocation services implemented within the environment and a well-configured key hierarchy in place that supports frequent certificate rotation and RSA-4096 keys to meet the 128-bit security strength requirements.

Privilege must be managed at the associated levels. As the level increases, the number of users with privileges should decrease. At the DevSecOps access level (accounting for most users), VPN credentials are needed. Above this are what's called elevated users, which require the same VPN credentials but with an added layer of multifactor authentication, such as a time-based, one-time password application. The most privileged users may require what's called two-person control, which aims to mitigate compromised identities at the individual level.

HOW TO SECURE A DEVSECOPS LAYOUT AND ITS COMPONENTS

A typical DevSecOps environment contains many components involving communications that are not just encrypted but also authenticated and authorized. The diagram shown in Figure 3 demonstrates an Istio service mesh layout containing proxies (or sidecars) to front each service, thereby providing mutually authenticated TLS connections and logging of each connection as it occurs. Additionally, only a small set of cipher suites are allowed (TLS_DHE_RSA_WITH_AES_256_GCM_SHA384(0x00,0x9F), TLS_ECDHE_RSA_WITH_AES_256_GSM_SHA384(0xC0,0x30)).

During the development process, it's important to apply the same principles while building the environment. This includes patched common vulnerabilities and exposures (CVEs) within each component to ensure that mTLS communications are used throughout the environment (including containers with no hard-coded secrets or root privilege), that components are configured to industry standards, and that the host machine components are patched and secured.

Additionally, all components within the DevSecOps environment must generate security logs while the security information and event management (SIEM) system processes the logs, determines the appropriate response, and confirms that the security policy of the environment is maintained. Protecting these log files is critical, as it can show the hacker what is (and what isn't) being monitored.

Figure 4 shows a diagram from the NIST 800-207 publication on zero-trust architecture. The diagram has been annotated on the outside, mapping the technologies to the individual components that must work together to enable a zero-trust architecture.

CVE scanning and mitigation isn't enough for vulnerability management. All components and their versions must be tracked, and because there are so many (hundreds), tooling is required for assistance. Parsing repositories of the logs can be used to make sure all components are identified, while the threat intelligence feed helps monitor how the current threats impact the component being used in the environment. Figure 5 shows a layout of key activities that must work together to form a solid vulnerability management capability.

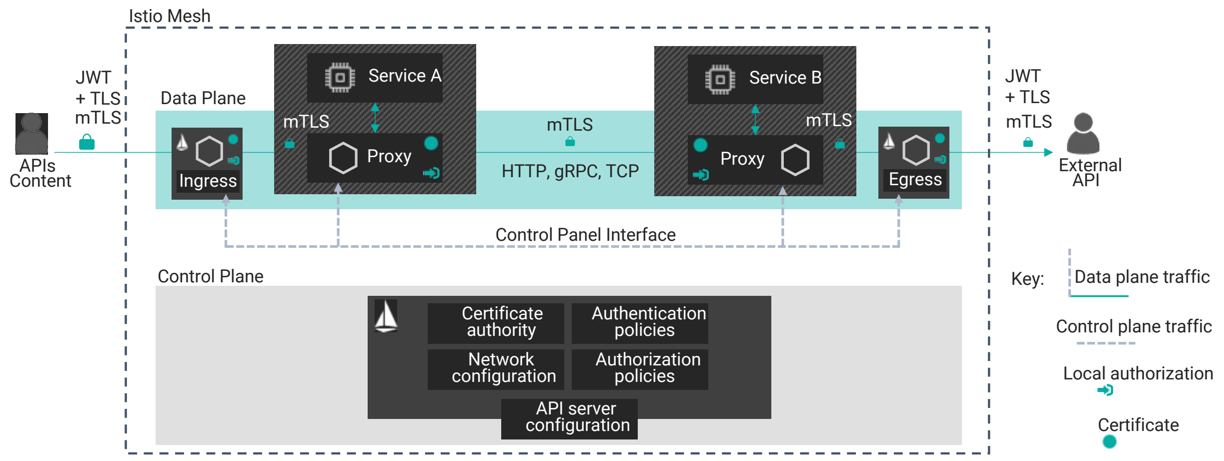

Figure 3. Istio service mesh

WNDRVR

THE IMPORTANCE OF DIE PRINCIPLES AND PEN TESTING

Incorporating the DIE model during design can greatly help with creating a robust, secure environment. These principles can be broken down as follows:

- **Distributed:** Multiple systems support the same overarching goal
- **Immutable:** The infrastructure doesn't change after it's deployed
- **Ephemeral:** The infrastructure has a short lifespan

Enabling a short container lifespan supports simplified distribution of updated secrets without requiring a container restart or ingestion processing, while also incorporating short-lived certificates that minimize the need for revocation services and key compromise remediation.

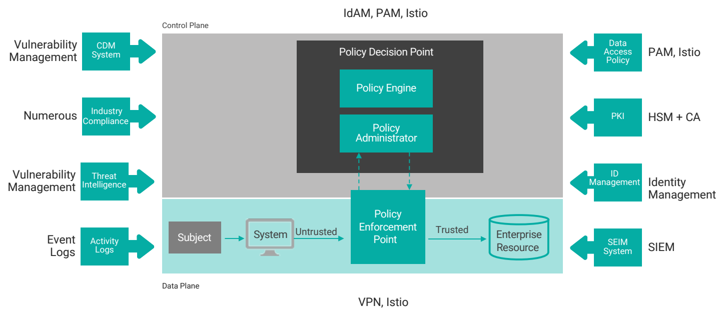

Figure 4. Zero-trust architecture

A cloud-native DevSecOps environment affords a major advantage over a static development environment. Creating a new, separate instance of the DevSecOps environment within a cloud service provider can remove any concern during third-party pen testing if a crash occurs.

In this arrangement, four levels of pen testing can be supported. Figure 6 shows the three main levels of testing for permissions (in addition to outside attackers with no access). Having an external vendor that specializes in vulnerability discovery provides major value in ensuring the security of the DevSecOps environment.

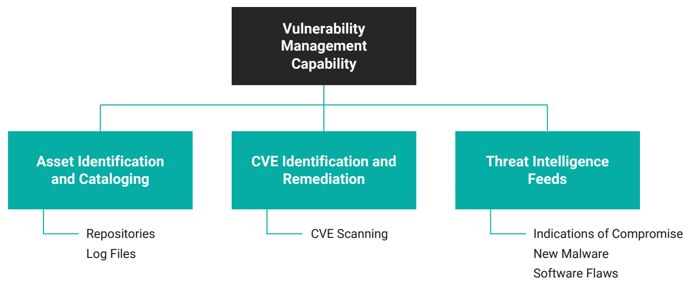
Vulnerability Management Capability
Asset Identification and Cataloging
CVE Identification and Remediation
Threat Intelligence Feeds
Repositories
Log Files
CVE Scanning
Indications of Compromise
New Malware
Software Flaws

Figure 5. Vulnerability management capability and its associated activities

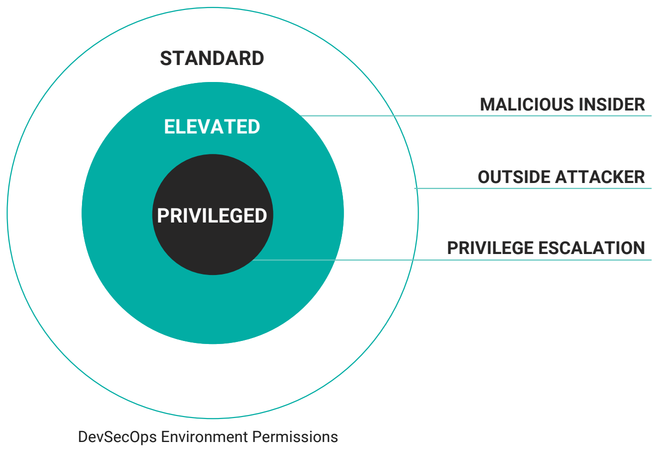

Figure 6. Levels of permission for third-party pen testing

CONCLUSION

As security testing becomes incorporated into the development lifecycle, it's important to make sure that the development itself is secure. Many technologies and related capabilities exist that can be used as the foundation for securing the DevSecOps environment. These include the use of an HSM, ID management, mutually authenticated and encrypted communication, a series of scanning tools, security event logging, and zero-trust network principles.

Security requires constant monitoring for vulnerabilities and updates, while application of the DIE principles shortens the lifespan of each container in the environment, limiting the time that a hacker has access to the components.

Finally, validation with third-party pen-test vendors provides a multifaceted security evaluation of the environment based on user privilege level to verify the security policy of the DevSecOps environment.

WNDRVR