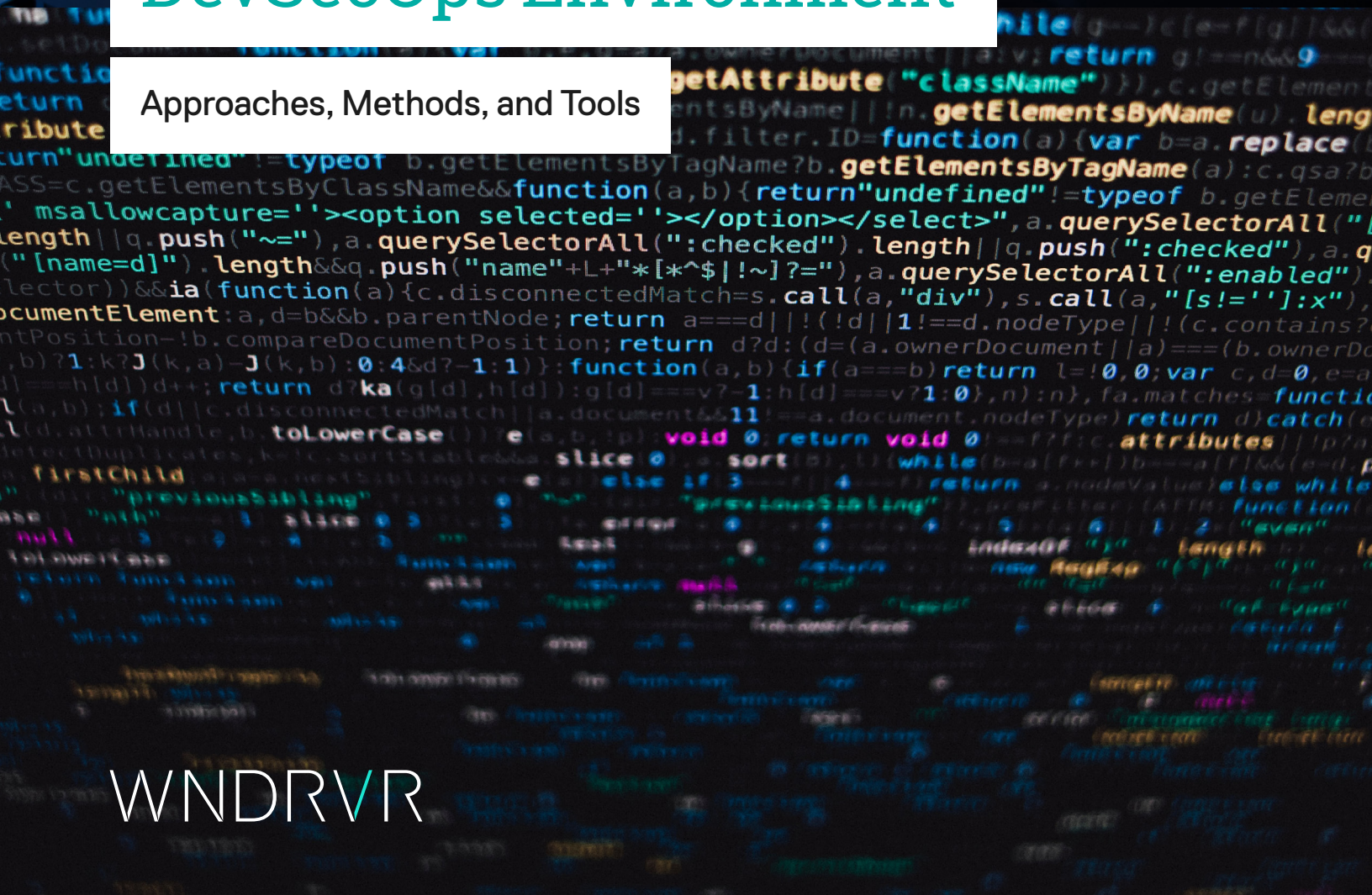
Securing the DevSecOps Environment

Approaches, Methods, and Tools



WNDRVR

EXECUTIVE SUMMARY

DevOps is enabling faster deployment and more secure software for devices by tightly coupling development and operations functions. Much of the security focus is on "shifting left" the security testing in the development lifecycle of the software. However, to put the "Sec" (security) into DevSecOps, the development environment of that DevSecOps pipeline itself must first be secured.

This paper provides a summary of:

- A security assessment of the essential elements of a DevSecOps environment
- The methods used in securing the environment, including:
  - Hardware security module/Private certificate authority
  - Identity management/Privileged access management
  - Mutually authenticated communications
  - Security validation tools
  - Event logging/Security information and event management
  - Zero trust network principles
- Practical usage of the distributed, immutable, and ephemeral (DIE) principles in furthering the security of the DevSecOps environment
- Lessons learned in engaging a third-party penetration test company to confirm that the DevSecOps environment is, indeed, secure

TABLE OF CONTENTS

INTRODUCTION

The recent compromise[1] of a prevalent networking monitoring system brings to the forefront the need to secure the development environment. Companies are now beginning to realize the value of having a secured DevSecOps environment.

As developers shift security left in the lifecycle, the bad actors and attackers are following suit, shifting their attacks further left in the same lifecycle. Information available at the time of this writing regarding this supply chain attack indicates that attackers inserted malicious source code into the development pipeline of the network monitoring tool. By focusing the attack at the development pipeline of the product, the malicious activity went unnoticed and the resulting binary image(s) were digitally signed by the vendor's code signing key. This created the illusion that the distributed code was authentic and validated from the vendor. Unsuspecting customers received "signed" binaries containing the compromised code, thus enabling the attacker's impact on thousands of victims.

On the surface, securing the development environment sounds like a straightforward request: Authenticate users, enable access controls on the repositories, and keep log files. However, an initial survey of the various specifications and best practices related to a DevSecOps environment yields a much longer list:

- *DoD Enterprise DevSecOps Reference Design*
- *Application Container Security Guide (NIST 800-190)*
- *DoD Cloud Computing Security Requirements Guide*
- *Committee on National Security Systems Policy 15 (CNSSP 15)*
- *Container Image Creation and Deployment Guide*
- *Digital Signature Standard (FIPS 186-4)*
- *DoD Container Hardening Guide*
- *Guide to Computer Security Log Management (NIST 800-92)*
- *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations (NIST 800-52)*
- *Privileged Account Management for the Financial Services Sector (NIST 1800-18A)*
- *Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations (NIST 800-171)*
- *Security and Privacy Controls for Federal Information Systems and Organizations (NIST 800-53)*
- *Zero Trust Architecture (NIST 800-207)*

The requirement list lengthens in the support of the three configurations that must be supported by the DevSecOps environment:

- Fully on-premises
- Cloud based
- Hybrid configuration

SECURITY STRENGTH

Security strength is defined[2] as the number of operations required to break a cryptographic algorithm or system. When the goal is to achieve a security strength of 256 bits, these larger cryptographic key sizes can have a negative impact on performance. In reviewing Annex B of CNSSP 15, along with Tables 2, 3, and 4 within NIST SP800-57 (Recommendations for Key Management), Part 1,[3] as guidance, the following cryptographic functions were selected:

- SHA-256, 128 bits of security strength
- RSA-4096, > 128 bits of security strength

A security strength of 128 bits is identified by NIST as "Acceptable" for both "through 2030" and "2031 and beyond."[4]

RSA-4096 was chosen over RSA-3072 as the standard signature algorithm based on the combination of security strength, intersecting support by the components within the DevSecOps environment, NIST compliance, certificate authority (CA), and hardware security module (HSM) support. Security strength can be calculated using the following equation:[5]

$$X = \frac{1.923 \times \sqrt[3]{L \times \ln(2)} \times \sqrt[3]{[\ln(L \times \ln(2))]^2} - 4.69}{\ln(2)}$$

Where L is the key size — in this case, 4096 — the equation defines a security strength of 149.73 bits. However, the security strength of a digital signature (hash function + digital signature algorithm) falls to the weakest algorithm and key sized used.[6]

RSA was chosen over elliptic curve cryptography due to the perceived backdoor created by a U.S. agency.[7]

SECURITY ASSESSMENT OF THE DEVSECOPS ENVIRONMENT

While many types of threat models exist,[8] each results in the following:

- Identification of the assets
- Identification of vulnerabilities to those assets
- List of security implementations to protect the asset from each vulnerability or group of vulnerabilities
- List of security-related events to be logged

To determine the list of assets, a typical DevSecOps environment[9] is represented as shown in Figure 1:

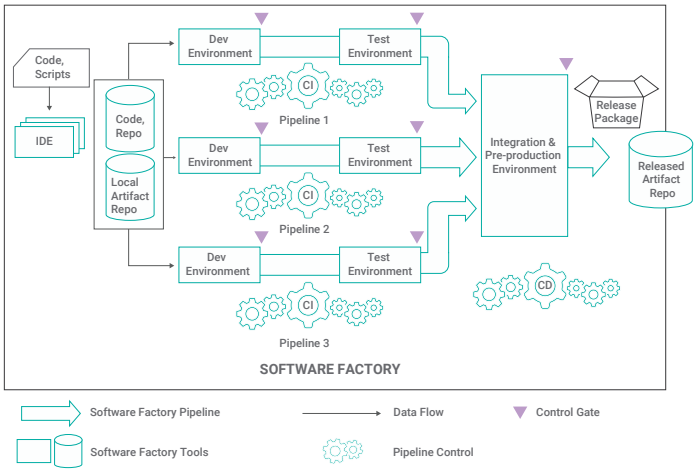


Figure 1. Typical DevSecOps environment

This diagram highlights several assets of the DevSecOps environment that need to be secured, including:

- The repositories
  - Code, local artifact, and released artifact repositories
- The software components
  - IDE, repos, development, and test components
- The build tools themselves:
  - Compilers and linkers
- The connectivity between the components
- The configuration of each component
- The storage elements of these components:
  - For both an on-premises and a cloud-based environment
- The event logs of all components

One asset not shown on the "typical" DevSecOps environment diagram is that of the hardware security module (HSM). The HSM is a physical computing device that safeguards and manages digital keys and performs encryption and decryption functions for digital signatures, strong authentication, and other cryptographic functions.[10]

To determine the vulnerabilities, we can start with a definition:[11] "The term *information security* means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction...." These unauthorized events can be attacks on the DevSecOps system.

Figure 2 identifies the asset vulnerable to each unauthorized event. This list shows the types of attacks from which each asset must be protected and is foundational for determining the security implementations required to protect each asset.

Asset	Unauthorized					
	Access	Use	Disclosure	Disruption	Modification	Destruction
Repositories	x		x		x	x
Software Components		x			x	x
Build Tools		x			x	
Connectivity	x		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

Figure 2. Types of attacks from which each asset must be protected

Multiple technologies are brought together to secure a DevSecOps environment. The capabilities of each are shown in Figure 3.

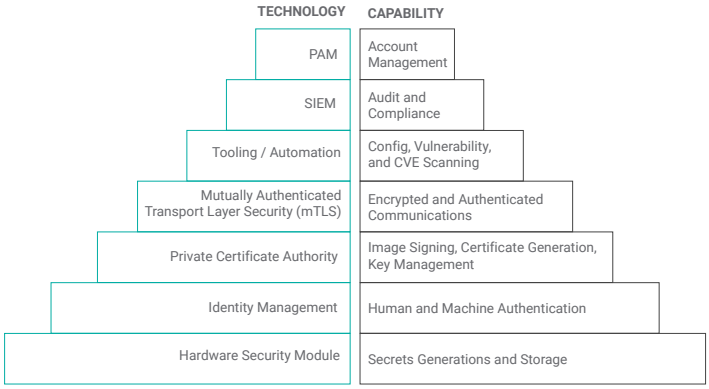


Figure 3. Capabilities of technologies to mitigate identified attacks

The following sections break down the capability of each technology and its application in securing the DevSecOps environment.

Hardware Security Module and Private Certificate Authority

The HSM provides the root of trust for the DevSecOps environment. The HSM is a purpose-built FIPS 140-2/3 Level 3–certified[12] cryptographic system and provides the processing for critical cryptographic operations in the DevSecOps environment. These cryptographic operations include:

- All cryptographic keying material, including X.509 certificates:
  - Code signing asymmetric keys
  - Symmetric key for optional image encryption
  - X.509 certificate generation for mutually authenticated TLS
- Integrity seal for released artifact objects

The HSM is a Public Key Cryptography Standard (PKCS)

#11[13]–compliant device. The interface to the HSM using only PKCS#11 enables support for both a fully on-premises configuration (by use of a physical HSM) or a cloud-based configuration (by use of a cloud service provider HSM).

Integrated with the HSM is a private certificate authority (CA). The CA provides the functionality of creation, verification, storage, and revocation status of all X.509 certificates and interfaces to the HSM via the PKCS#11 interface. The CA also tracks the revocation status of all X.509 certificates within the DevSecOps system. Use of the Onsite Certificate Status Protocol (OCSP) throughout the environment verifies that all X.509 certificates are valid.

Cryptographic key hierarchy for the DevSecOps environment is shown in Figure 4. Cryptoperiods for X.509 certificates used in the DevSecOps environment typically align to the recommendations. However, the TLS and SSH certificates have a significantly shorter lifespan than that of a standard authentication key. These certificates span no more than 24 hours to reduce the impact of a compromise.

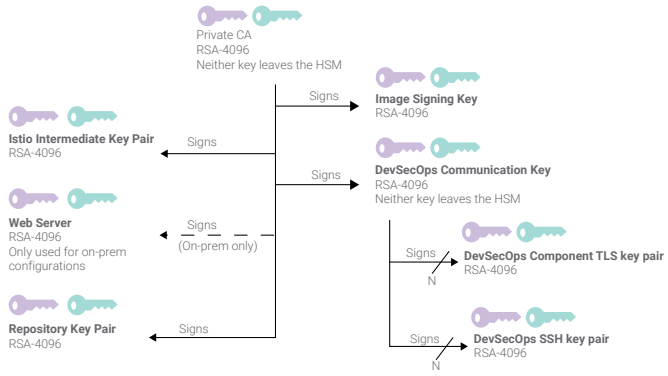


Figure 4. Cryptographic key hierarchy for the DevSecOps environment

Figure 5 lists the summary of each key, the NIST key type, and cryptoperiod.

Key	NIST Key Type	Cryptoperiod	Notes
Private CA	Trust Anchor	< 3 years	Trust Anchor keys
Istio Key Pair	Authentication	1 month	Used as the intermediate key pair for Istio to generate its certs
Web Server	Authentication	< 2 years	Only for fully on-premises configurations
Repository	Signature	Unlimited	Used to provide an integrity seal on the released repo objects
Image Signing	Signature	<2 years	Signs delivered image files
Comms Key	Authentication	1 year	Basis of all lower-level keys Public key never released
TLS Keys	Authentication	Hours	Certs regenerate and are consumed by the container when the container begins execution

Figure 5. Cryptographic key attributes

Note that due to the controlled and managed nature of the DevSecOps environment, neither a Registration Authority (RA) nor a Verification Authority (VA) are required. The act of verifying a certificate is not applicable because the COMSEC lead initiates and manages the key hierarchy for the environment. All components within the DevSecOps environment are known entities.

Identity Management/Privileged Access Management

Identity and access management (IdAM) of users is a vital component of securing the DevSecOps environment, just as it is in an IT environment. The components within the DevSecOps environment are large in number and dynamic by design. Without careful orchestration of which accounts have access and privileged access to these components, catastrophic results can ensue. The DevSecOps environment should be firewalled off from the rest of the corporate network and require access via a virtual private network (VPN) that has a limited connection period associated with it. This approach begins the alignment to the principle of least privilege by reducing the number of corporate users who can access the DevSecOps environment.

Following are some definitions:

- A privileged command is executed on an information system involving the control, monitoring, or administration of the system, including security functions and associated security-relevant information.
- A privileged account is one that can perform operations within a platform without restriction.
- Privileged account management (PAM) is a domain within identity and access management (IdAM) that focuses on monitoring and controlling the use of privileged accounts.[14]
- Privileged access management (PAM [overloaded acronym]) is the summation of strategies and technologies for controlling access to privileged accounts.

As access moves upward in terms of privilege, as shown in Figure 6, a defense-in-depth approach in protecting the DevSecOps environment also increases.

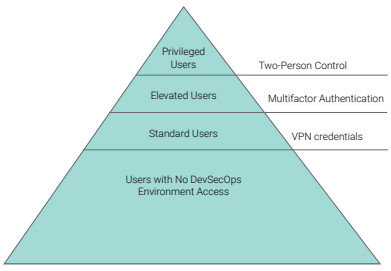


Figure 6. Defense-in-depth approach to user management

The principle of least privilege is expanded from human accounts to include service accounts as well. This is to minimize exposure in case these accounts get compromised. Also, the incorporation of multifactor authentication mechanisms must be employed for elevated human accounts.

Taking the human users within a corporation into consideration, we can apply a defense-in-depth approach in protecting access to the DevSecOps environment, as listed in Figure 7.

Access	Defense Layer
Corporate Network	Corporate ID Credentials
DevSecOps Environment	VPN Credentials
Elevated Commands	MFA
Privileged Commands	Two-Person Control

Figure 7. User access defense-in-depth approach

Taking the intersection of the roles outlined in NIST SP1800-18[15] along with National Initiative for Cybersecurity Careers and Studies[16] and aligning those for a DevSecOps environment development team, we arrive at a hierarchy as shown in Figure 8.

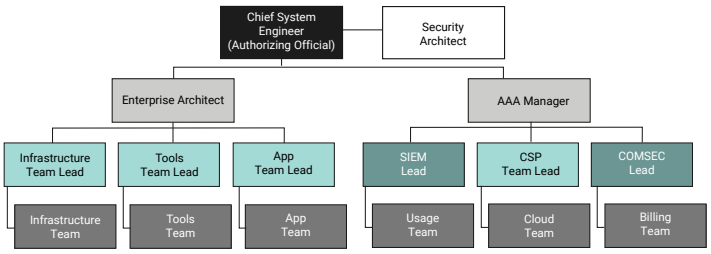


Figure 8. Access-driven hierarchy in a DevSecOps environment

Taking an access-driven hierarchy into consideration enables a simplified alignment to the authorization of the commands that each team member can perform in their day-to-day activities and directly assists in the recognition of two-person authorization for privileged commands within the DevSecOps environment.

Mutually Authenticated Communications

All communications within the DevSecOps environment are both mutually authenticated and encrypted using Transport Layer Security (TLS) version 1.2. Version 1.2 was required because not all components used in the DevSecOps environment support version 1.3. Based on current NSA Guidance,[17] cryptographic parameters are to meet the algorithm requirements defined in Committee on National Security Systems Policy (CNSSP) 15, which leads to a subset of the NIST 800-52 cipher suites (with the removal of AES-CBC ciphers[18]), to ensure perfect forward secrecy.[19] Prioritizing the resulting pruned list of cipher suites results in a very small subset:

- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x00, 0x9F)
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xC0, 0x30)

Though a very small set, it enables an intrusion detection event to be defined. If a client tries to connect to a server with ciphers outside of this list, it can indicate an attack. The same is true from the client perspective: If the server responds with a cipher suite mismatch error, that too indicates an attack.

TLS implementation between the user and the DevSecOps system is dependent upon the DevSecOps environment configuration type. For a hybrid or cloud-based configuration, a certificate provided by a commercial PKI certificate vendor[20] is used. For a fully on-premises configuration, a self-signed certificate from the CA/HSM is used. This type of certificate requires an update to each browser that connects to the DevSecOps environment where the web server certificate must be imported into the browser's list of trusted root certificate authorities.

Communications within the DevSecOps environment extends TLS to use mutually authenticated TLS to ensure that both client and server are authenticated. The service mesh Istio is configured for mTLS[21] and is restricted to the above-listed cipher suites.[22]

Security Validation Tools

The large number of components used in the DevSecOps environment demands that automated security validation tools be in place and tightly integrated into the development and maintenance of the DevSecOps environment itself. While DevSecOps is seen as "shifting left" the security-related testing of the application development, the same tooling that the DevSecOps environment pipeline provides to the application can be used to maintain the DevSecOps environment itself, with frequent updates and maintenance.

In looking at a Kubernetes-based configuration for a DevSecOps environment, there are several areas that must be continuously scanned, as shown in Figure 9.

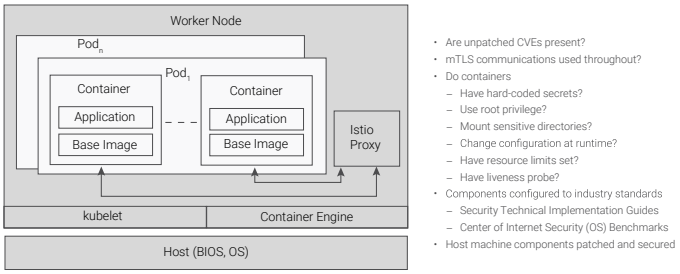


Figure 9. Security scanning components of a Kubernetes-based configuration

In addition to container applications, CVEs must be tracked for both processors and boards used in the DevSecOps environment. While the hardware components can be easily overlooked, careful review and tracking of these issues must occur in parallel with the tracking of the software-specific CVEs as they impact a container-based[23] configuration.[24]

While the market has no shortage of tooling, one approach to tooling is of categorization. The referenced U.S. DoD Enterprise DevSecOps Reference Design[25] contains a listing of different testing tool categories. Initial priority should be given to the following categories:

Tool Category	Reasoning
Static Application Security Test (SAST)	Verify code before deployment and ensure all CVEs are patched
Network Testing	Confirm network configuration
Container Policy	Confirm containers align to a strong security posture
Software License Compliance Checker	Confirm no license violations
Security Compliance Tool	Ensure components are configured to industry standards

Figure 10. Initial DevSecOps environment testing tools

Ensure that all tools are properly configured and that no defaults[26] carry forward into the DevSecOps environment.

EVENT LOGGING/SECURITY INFORMATION AND EVENT MANAGER

Due to the large number of components within a DevSecOps environment, a Security Information and Event Management (SIEM) is required. The input to the SIEM are security events that occur within each DevSecOps component, in the form of audit log entries. The SIEM then ingests these security events and applies a set of rules to determine whether the security policy of the DevSecOps environment is being enforced or a violation has occurred. SIEM rulesets can take the form of "if this, then that" for specific events, and then models can be defined for more abstract scenarios.

Security-related events can be of extreme value to an attacker, as they show what is being monitored and the responses to those events. This requires that the log events must be protected while being sent to and stored within the SIEM. Also, an integrity seal must be applied over the log data to ensure that it is not modified while it is archived. This integrity seal takes the form of a digital signature, and the duration of this archiving can go for as long as six years.[27]

It is of paramount importance that all components within the DevSecOps environment synchronize time to a single time server or a GPS time server and align to a single time zone (GMT). When doing forensic analysis, it is important that the timeline of security events can be accurately reconstructed to determine the violation of the security policy.

ZERO TRUST NETWORK PRINCIPLES

A zero trust architecture (ZTA) is an enterprise cybersecurity architecture that is based on zero trust principles and designed to prevent data breaches and limit internal lateral movement.[28] In the past, architectures assumed that once a subject authenticates into the system, it can be treated as trustworthy from that point forward. ZTA does not rely on implied trustworthiness; it requires a much more granular level of access to each resource and assumes the attacker is present within the system.

Using the technologies defined in this paper, Figure 11 shows the instantiation of the core zero trust components[29] for a DevSecOps environment.

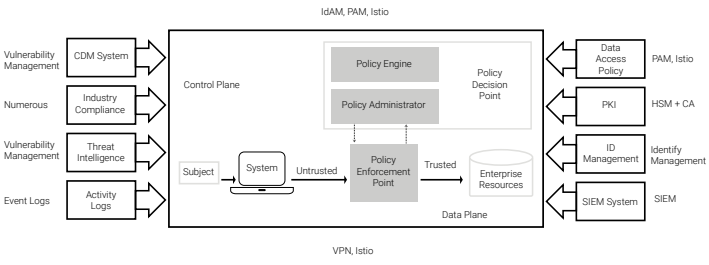


Figure 11. Instantiation of the core zero trust components

The Continuous Diagnostics and Mitigation (CDM) program and threat intelligence system can fall under the program's vulnerability management capability. This capability expands from the narrow focus of CVE scanning to include asset identification/cataloging along with threat intelligence feeds, as shown in Figure 12.

The inclusion of these data further enhances the ability of the organization to perform the task of vulnerability management. Automating the listing of assets (e.g., component and version number) in the DevSecOps environment increases the accuracy of CVE scanning processing. Incorporation of indicators of compromise (IoC) provides a feedback loop affirming that the vulnerability management capability is monitoring the correct parameters within the environment.

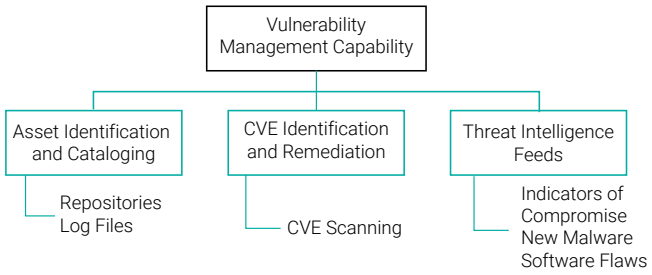


Figure 12. Vulnerability management decomposition

DIE PRINCIPLE APPLICATION

The DIE principles[30] are defined as follows:

- **Distributed:** Multiple systems supporting the same overarching goal
- **Immutable:** Infrastructure that doesn't change after it's deployed
- **Ephemeral:** Infrastructure with a very short lifespan

While the distributed and immutable principles overlie on cloud-native computing in general, the ephemeral principle demands close examination for a DevSecOps environment.

"Ephemerality creates uncertainty for attackers."[31]

Dramatically reducing the time that an orchestrated container executes from the current average of .5 days[32] to tens of minutes limits the amount of damage that an attacker can accomplish. To inject malware or even take complete control over a container that is torn down within tens of minutes is a "whack-a-mole" game that will frustrate even the most persistent attacker.

Enabling a short container lifespan directly supports:

- Simplified distribution of updated secrets without requiring a container restart or ingestion processing
- The use of short-lived certificates to minimize the need for revocation services and key compromise remediation
- Timely deployment of patched container vulnerabilities, not just for the application but for the base image as well

THIRD-PARTY PENETRATION TESTING

"Penetration testing is a specialized type of assessment conducted on information systems or individual system components to identify vulnerabilities that could be exploited by adversaries."[33]

As discussed, there are many moving parts to a DevSecOps environment. Each of these parts and the interfaces to other parts within the environment are doorways for attackers. The complexity of this environment warrants both an internally and an externally resourced pen test activity to aid in identifying vulnerabilities.

Having an external vendor that specializes in vulnerability discovery provides enormous value to ensure the security of the DevSecOps environment. Being rewarded for finding vulnerabilities is a compelling reason for using external vendors.

The DevSecOps environment can be replicated in a separate environment that provides a significant advantage over a production environment. Having a separate environment allows the pen test team to work unconstrained, compared to testing a production system, for concern of interfering with the production system.

To maximize value of the pen test team, four levels of access are used:

- **None:** Has the team test the system the way an outside attacker would
- **Standard:** Allows the team to have access into the system to discovery vulnerabilities, as a hacker would once the environment is breached
- **Elevated:** Allows the team to access the second tier of privileged functions within the environment
- **Privileged:** Ensures that the PAM definitions have been implemented correctly to ensure that the most critical functions within the system require a two-person workflow

These permission rings and attack vectors are shown in Figure 13.

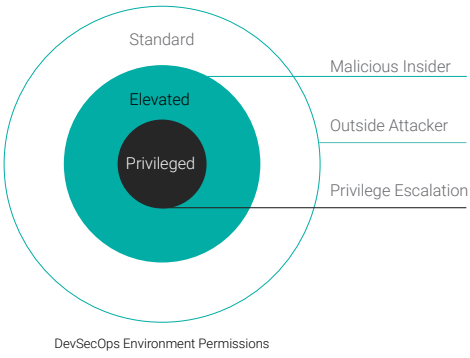


Fig. 13. Penetration testing account access

CONCLUSION

As security testing becomes incorporated into the development lifecycle, it is of paramount importance that the development environment (the DevSecOps environment) itself be secured. Several technologies have been presented and capabilities identified that can be used as the foundation in securing the DevSecOps environment. The combined use of an HSM, IdAM, mutually authenticated encryption paths, scanning tools, security event logging, and zero trust network principles strengthen the security of the DevSecOps environment. Application of the DIE principles extends the security posture of the secured environment by having a short lifespan for each container in the environment. Finally, an approach was presented to leverage third-party penetration vendors to provide multifaceted security evaluation, based on user privilege level, to verify the security policy of the DevSecOps environment.

This paper was written and presented at the Embedded World Digital 2021 conference by Arlen Baker, Principal Security Architect, Technology Office, Wind River®; and Matt Jones, Vice President, Engineering, Wind River.

REFERENCES

1. *us-cert.cisa.gov/ncas/alerts/aa20-352a*
2. *csrc.nist.gov/glossary/term/security_strength*
3. *nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf*
4. Ibid, Table 4
5. *csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf*, Section 7.5
6. *csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf*, section 5.6.2
7. *ieeexplore.ieee.org/abstract/document/7782697*
8. *threatmodeler.com/threat-modeling-methodologies-overview-for-your-business*
9. *dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583*, Figure 9
10. *en.wikipedia.org/wiki/Hardware_security_module*
11. *www.govinfo.gov/content/pkg/USCODE-2011-title44/html/USCODE-2011-title44-chap35-subchapIII-sec3542.htm*
12. *csrc.nist.gov/publications/detail/fips/140/3/final*
13. *docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html*
14. *www.nccoe.nist.gov/sites/default/files/library/sp1800/fs-pam-nist-sp1800-18-draft.pdf*
15. *nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf*, AC-3 (2)
16. *niccs.cisa.gov/workforce-development/cyber-security-workforce-framework*
17. *media.defense.gov/2021/Jan/05/2002560140/-1/-1/0/ELIMINATING_OBSOLETE_TLS_UOO197443-20.PDF*
18. *nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf*, section 3.3.2
19. Ibid, Appendix D
20. *dl.dod.cyber.mil/wp-content/uploads/pki-pke/pdf/unclass-ss_using_commercial_pki_certificates.pdf*, Figure 6
21. *istio.io/latest/docs/reference/config/networking/gateway/#ServerTLSSettings*
22. Ibid
23. *containerjournal.com/features/what-do-containers-have-to-do-with-being-cloud-native-anyway*
24. *meltdownattack.com*
25. *dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdf?ver=2019-09-26-115824-583*, tables 8 through 10
26. *www.bleepingcomputer.com/news/security/fbi-hackers-stole-government-source-code-via-sonarqube-instances*
27. *www.hhs.gov/hipaa/for-professionals/compliance-enforcement/audit/protocol/index.html*, 164.316(b)(2) (i) and others
28. *csrc.nist.gov/publications/detail/sp/800-207/final*
29. Ibid, Figure 2
30. *i.blackhat.com/USA-19/Wednesday/us-19-Shortridge-Controlled-Chaos-The-Inevitable-Marriage-Of-DevOps-And-Security.pdf*
31. Ibid
32. *www.datadoghq.com/docker-adoption*
33. *csrc.nist.gov/publications/detail/sp/800-53/rev-5/final*