Eccouncil

(312-97)

Certified DevSecOps Engineer (ECDE)

Total: **100 Questions**

Link:

MYEXAM.FR

C. By creating a cookbook, defining a trigger, security response, and notification

D. By creating a playbook, defining a trigger, Alert a response, and notification

Answer: C

Explanation:

Sqreen, now part of Datadog, is a Runtime Application Self-Protection (RASP) solution designed to protect applications from attacks in real-time. To effectively manage and respond to security incidents, Sqreen leverages a structured approach that includes defining detection rules and corresponding actions.

The process of configuring Sqreen to send notification alerts to Slack involves setting up specific policies within the platform. At the core of this are "cookbooks." In Sqreen's terminology, a **cookbook** is a collection of predefined or custom security policies, rules, and actions that dictate how the RASP agent should behave when specific events occur. These cookbooks encapsulate the logic for threat detection and response.

Within a cookbook, a **trigger** defines the specific conditions or security events that Sqreen should look for. This could be anything from a detected SQL injection attempt, a cross-site scripting (XSS) attack, an unusual login pattern, or an unhandled exception. The trigger acts as the "if" part of a conditional statement, identifying when a security event of interest has occurred.

Once a trigger condition is met, Sqreen needs to perform an action. This action is defined by a **security response**. A security response specifies what Sqreen should do immediately after a trigger fires. Common security responses include blocking the malicious request, logging the event, or, crucially, generating an alert. Generating an alert means flagging the incident for further attention.

Finally, to communicate these generated alerts to a team, a **notification** mechanism is employed. Notifications define how and where these alerts should be sent. Sqreen integrates with various communication platforms, including Slack. Therefore, after a security response generates an alert, a notification policy directs Sqreen to send the details of that alert to a configured Slack channel. This ensures that Terry's team receives real-time information about security issues directly in their collaborative workspace.

Option C accurately reflects this structured workflow:

1. **Creating a cookbook:** This establishes the overall policy set for protecting the application.
2. **Defining a trigger:** This specifies the exact security event that needs to be detected.
3. **Security response:** This dictates the action Sqreen takes upon detection, which includes generating an alert.
4. **Notification:** This configures the delivery of that alert to Slack.

This approach aligns with best practices for automated incident response in cloud-native environments, where RASP tools are critical for providing real-time, in-application protection and seamless integration with DevOps workflows for immediate team awareness.

For further research:

Datadog/Sqreen documentation on rules and responses: While Sqreen is now integrated into Datadog, its core concepts for RASP rule definition and responses remain relevant. Searching Datadog's documentation for "RASP rules," "security signals," or "integrations" will provide context.

Example: Datadog Security Platform Documentation
Specific information about RASP capabilities within Datadog might be found under Application Security Management (ASM) or Cloud Security Management (CSM): Datadog Application Security Management
General RASP concepts: OWASP RASP Project

AWS WAF Documentation: https://aws.amazon.com/waf/
Amazon CloudWatch Documentation: https://aws.amazon.com/cloudwatch/
AWS Config Documentation: https://aws.amazon.com/config/

Question: 7

GainInsights is an IT company that develops mobile applications software. On February 11, 2022, the organization became a victim of a cyber-attack. The attacker targeted the organization's application and compromised some important functionality. After the incident, the DevSecOps team of GainInsights identified the cause of the security issue, resolved it, and noted it for future reference. Based on this information, which of the following set of tests was conducted by GainInsights?

A. Security smoke tests
B. Security acceptance tests
C. White box testing
D. Blameless post-mortem

Answer: D

Explanation:

The correct answer is D. Blameless post-mortem.

A blameless post-mortem is a structured process conducted after an incident, such as the cyber-attack GainInsights experienced, to understand its root causes, contributing factors, impact, and the effectiveness of the response. The scenario explicitly states that "After the incident, the DevSecOps team of GainInsights identified the cause of the security issue, resolved it, and noted it for future reference." These actions perfectly encapsulate the core objectives of a blameless post-mortem.

The initial step of "identifying the cause" involves performing a thorough root cause analysis to pinpoint the exact vulnerabilities or systemic failures that led to the compromise. "Resolved it" refers to the implementation of both immediate mitigations and long-term fixes to prevent recurrence. Most critically, "noted it for future reference" signifies the documentation of lessons learned, which includes outlining preventative measures, enhancing security controls, refining incident response plans, and updating organizational knowledge. This feedback loop is essential for continuous improvement within a DevSecOps framework.

The term "blameless" is key because it fosters a culture of psychological safety, encouraging team members to openly discuss failures without fear of individual reprimand. This approach shifts the focus from individual blame to systemic issues, process gaps, and technical deficiencies that require collective attention and resolution. By embracing this practice, organizations like GainInsights can transform security incidents into valuable learning opportunities, thereby strengthening their overall security posture and operational resilience over time.

In contrast, the other options describe different types of security activities. Security smoke tests are preliminary checks to ensure basic security functionality is working, typically performed before or during deployment. Security acceptance tests verify that security requirements are met, usually prior to or as part of a formal release. White box testing is a proactive security testing methodology where testers have full knowledge of the system's internal structure and code, generally performed during the development or pre-deployment phases. None of these options accurately describe the comprehensive post-incident analysis, resolution, and learning process undertaken by GainInsights after the cyber-attack.

Authoritative Links for Further Research:

file containing a list of keywords or regular expressions to search for. In this scenario, wordlist/keywordsfile.txt would contain various patterns, including "yahoo" and other sensitive terms relevant to the e-commerce application, that GitGraber should actively seek. This feature is crucial for comprehensive pattern matching across a repository.

3. **-q "\yahoo\"**: The -q or --query option allows for specifying an additional, direct keyword or regular expression. While the -w option handles a file of patterns, -q enables William to include an immediate, specific search term like "yahoo" alongside the patterns from the wordlist file. This demonstrates GitGraber's flexibility in combining broad pattern searches with targeted queries.
4. **-s**: The -s or --stream option is fundamental to William's requirement for "real-time" detection. When enabled, GitGraber continuously monitors GitHub events, such as new commits, pushes, or pull requests, as they happen. This streaming capability ensures that any newly introduced secrets are immediately flagged, significantly reducing the exposure window and preventing them from lingering unnoticed in the codebase. Real-time monitoring is a cornerstone of continuous security in modern CI/CD pipelines.

The other options (A, B, C) are incorrect because they use flags that are not designed for specifying a wordlist file in GitGraber. For instance, -g is typically used for a GitHub personal access token, and -k is for a single keyword directly on the command line, not a file containing multiple patterns. Using these incorrect flags would either lead to a command execution error or prevent GitGraber from properly loading the intended keywordsfile.txt.

Implementing such secret detection tools directly into the DevSecOps pipeline is vital for upholding application security. It aligns with the "shift-left" security principle by catching credential leaks, API keys, and other sensitive data early, before they are deeply embedded or deployed. This proactive measure mitigates supply chain risks, prevents unauthorized access, and strengthens the overall security posture of cloud-based software products, thereby protecting against common attack vectors in cloud computing environments.

Authoritative Links for further research:

GitGraber GitHub Repository: https://github.com/hisxo/gitGraber
OWASP Top 10 (A01:2021-Broken Access Control & A04:2021-Insecure Design - related to secret management): https://owasp.org/www-project-top-ten/
NIST SP 800-207 Zero Trust Architecture (context for securing access and preventing unauthorized disclosure): https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

Question: 11

Sandra Oliver joined SinClare Soft Pvt. Ltd. as a DevSecOps engineer in January of 2010. Her organization develops software and web applications related to the healthcare industry. Using IAST runtime security testing technology, she is detecting and diagnosing security issues in applications and APIs. The IAST solution used by Sandra encompasses a web scanner with an agent that works inside the server that hosts the application to provide additional analysis details such as the location of the vulnerability in the application code. Based on the given information, which of the following IAST solutions is Sandra using?

A. Active IAST
B. Semi-active IAST
C. Semi-passive IAST
D. Passive IAST

Answer: A

Explanation:

input and passing it as an argument to the docker inspect command.

docker inspect retrieves low-level information about Docker objects, typically outputting a comprehensive JSON document. To extract just the port details, the --format (or -f) flag is used, specifying a Go template for precise output formatting. The argument ': Ports=' employs .Ports as a Go template field, which directly accesses the "Ports" section within the detailed JSON output of docker inspect. This section enumerates all port mappings, including the container's private port, any host-bound public port, and the protocol (e.g., tcp, udp). The single quotes ensure the entire template string is passed correctly, while the colon and space serve as an aesthetic prefix to the port data.

This command is vital for DevSecOps engineers like Peter, as unnecessary exposed ports represent a significant security vulnerability by increasing the attack surface. Regularly auditing exposed ports aligns with DevSecOps principles of "shift-left" security, integrating checks early to ensure only strictly necessary ports are accessible. This adheres to the principle of least privilege and maintains a robust security posture, preventing potential unauthorized access.

Options B and C are incorrect because docker inspect lacks an --all flag; its use would result in an "unknown flag" error. Option A, docker inspect --format : Ports, is syntactically flawed for a Go template and would fail to correctly extract port information, as .Ports is the proper field access.

Authoritative Links for Further Research:

docker ps documentation: https://docs.docker.com/engine/reference/commandline/ps/
docker inspect documentation: https://docs.docker.com/engine/reference/commandline/inspect/
xargs (Linux man page - general utility): https://man7.org/linux/man-pages/man1/xargs.1.html
Docker Go Template formatting: https://docs.docker.com/config/formatting/

Question: 15

Thomas Gibson has been working as a DevSecOps engineer in an IT company that develops software products and web applications related to law enforcement. To automatically execute a scan against the web apps, he would like to integrate InsightAppSec plugin with Jenkins. Therefore, Thomas generated a new API Key in the Insight platform. Now, he wants to install the plugin manually. How can Thomas install the InsightAppSec plugin manually in Jenkins?

A. By creating a .hpi file and uploading to his Jenkins installation
B. By creating a .conf file and uploading to his Jenkins installation
C. By creating a .war file and uploading to his Jenkins installation
D. By creating a .zip file and uploading to his Jenkins installation

Answer: A

Explanation:

To manually install the InsightAppSec plugin in Jenkins, Thomas must use the standard Jenkins plugin file format, which is a .hpi file.

A .hpi (or .jpi) file is a special type of JAR (Java Archive) file specifically designed to package Jenkins plugins. It contains all the necessary compiled Java code, resources, and a plugin manifest file that Jenkins uses to understand and load the plugin. When a plugin is developed, it's compiled and packaged into this .hpi format for distribution and installation.

The manual installation process in Jenkins involves navigating to the "Manage Jenkins" section, then selecting

The correct command Gabriel should use to delete the secret in Vault is vault kv delete secret/wejskt.

Vault is a powerful secrets management tool developed by HashiCorp, essential for securing sensitive data in modern IT infrastructures, particularly in cloud computing and DevSecOps pipelines. It provides a centralized service to store, access, and encrypt secrets across various systems. The Key-Value (KV) secrets engine is one of the most commonly used backends in Vault, designed for storing arbitrary secrets like API keys, database credentials, or configuration data, which is precisely what Gabriel used to store secret/wejskt.

To interact with the KV secrets engine, the vault kv subcommand is utilized. This subcommand allows users to perform various operations such as put (write), get (read), delete (remove), and list secrets within a mounted KV engine. In Gabriel's scenario, he aims to remove the secret he wrote, making the delete operation the appropriate choice.

The delete subcommand directly instructs the Vault server to remove the secret located at the specified path (secret/wejskt) within the KV engine. When using a KV secrets engine version 2 (KVv2), which is common, the delete operation soft-deletes the secret, marking it as deleted but retaining its historical versions. This can be beneficial for auditing or potential recovery, aligning with good DevSecOps practices for traceability and incident response. If a complete, irrecoverable removal (destroying all versions) were needed, the vault kv destroy command would be used, but the question specifically asks to "delete" the secret.

Let's analyze why the other options are incorrect:

A. vault kv del secret/wejskt: While del is a common abbreviation for delete in many command-line interfaces, the official HashiCorp Vault CLI consistently uses the full delete subcommand for the KV secrets engine. del is not a standard or recognized alias for this specific operation in the Vault CLI.
B. vault kv -delete secret/wejsktyup[and **C. vault kv -del secret/wejskt**: The hyphen (-) prefix before delete or del signifies a command-line flag, not a subcommand. Subcommands in Vault are positional arguments that follow the main command and engine (e.g., vault kv). Therefore, these commands are syntactically incorrect for performing a delete operation and would result in an error or be interpreted as an attempt to pass an unrecognized flag.

In the context of the DevSecOps principles, the ability to quickly and effectively delete secrets is crucial, especially following security incidents like the brute-force attack detected by Splunk. Rapid secret revocation ensures that compromised or potentially compromised secrets are no longer valid, thereby minimizing the attack surface and mitigating further risk to the company's sensitive data and infrastructure in the cloud.

For further authoritative research on HashiCorp Vault's KV secrets engine and its commands, you can refer to:

HashiCorp Vault KV Secrets Engine documentation: https://developer.hashicorp.com/docs/vault/secrets/kv
Vault CLI kv command reference: https://developer.hashicorp.com/docs/vault/commands/kv
Vault CLI kv delete command reference: https://developer.hashicorp.com/docs/vault/commands/kv/delete

Question: 18

Michael Rady recently joined an IT company as a DevSecOps engineer. His organization develops software products and web applications related to online marketing. Michael deployed a web application on Apache server. He would like to safeguard the deployed application from diverse types of web attacks by deploying ModSecurity WAF on Apache server. Which of the following command should Michael run to install ModSecurity WAF?

A. sudo apt install libapache2-mod-security2 –x
B. sudo apt install libapache2-mod-security2 –z
C. sudo apt install libapache2-mod-security2 –w
D. sudo apt install libapache2-mod-security2 –y

Authoritative Links for Further Research:

OWASP Secure Design Principles: https://owasp.org/www-community/Secure_Design_Principles
NIST Special Publication 800-27A, Engineering Principles for Information Technology Security (A Baseline for Protecting Information Technology Systems):
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-27a.pdf
Microsoft Security Development Lifecycle (SDL): https://www.microsoft.com/en-us/securityengineering/sdl/process

Question: 21

Jason Barry has been working as a DevSecOps engineer in an IT company that develops software products and applications for ecommerce companies. During the build-time check, Jason discovered SQL injection and XXS security issues in the application code. What action does the build-time check perform on the application code?

A. It will stop the build process
B. It will ignore the security issue and continue the build process
C. It will send a message to issue and project management tool and continue with deploy-time check
D. It will send an alert to SIEM and continue with test-time check

Answer: A

Explanation:

Build-time checks are a fundamental component of a robust DevSecOps pipeline, embodying the "shift-left" security principle by integrating security measures early in the Software Development Life Cycle (SDLC). When Jason discovered severe vulnerabilities like SQL injection and Cross-Site Scripting (XSS) during this phase, the correct and most secure action is to halt the build process.

This action serves as an automated security gate, preventing critical flaws from propagating further down the Continuous Integration/Continuous Delivery (CI/CD) pipeline. SQL injection and XSS are high-impact vulnerabilities that can lead to data breaches, unauthorized access, and compromised user experiences. Allowing a build containing such known issues to proceed to testing, let alone deployment, would introduce unacceptable risk and significantly increase the cost and complexity of remediation later on.

By stopping the build (Answer A), the DevSecOps pipeline enforces a critical quality and security standard. This ensures that only code meeting defined security thresholds is compiled, packaged, and prepared for subsequent stages. This immediate halt forces developers to address and fix the vulnerabilities at their source, where changes are least disruptive and most cost-effective. Cloud-native CI/CD platforms like AWS CodePipeline, Azure DevOps, and GitLab CI/CD are designed to orchestrate these automated gates, failing builds when security scanner thresholds are met.

Conversely, ignoring the issue (B), merely sending a message and continuing (C), or proceeding to later checks (D) would undermine the entire purpose of early security testing. These alternatives would imply a reactive security posture rather than the proactive, preventative approach central to DevSecOps, ultimately leaving the application vulnerable to exploitation. Therefore, stopping the build is a non-negotiable step to maintain the integrity and security of the software product.

For further research:

OWASP Top 10 Web Application Security Risks: https://owasp.org/www-project-top-ten/
NIST Special Publication 800-160, Vol. 1 - Developing Cyber Resilient Systems:
https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf (See Section 3.2.3, "Security

PortSwigger (Burp Suite) on Burp Collaborator/OAST:

https://portswigger.net/burp/documentation/collaborator
https://portswigger.net/burp/documentation/desktop/functions/scanner/auditing-techniques#out-of-band
OWASP (General OAST Concept):

https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/00-Introduction.html (While not a direct OAST page, OWASP covers the vulnerabilities OAST aims to find, like SSRF, XXE, blind SQLi, which inherently require OAST-like techniques for reliable detection).

Question: 25

Walter O'Brien recently joined as a junior DevSecOps engineer in an IT company located in Lansing, Michigan. His organization develops robotic process automation software for various clients stretched across the globe. Walter's team leader asked him to configure username and user email for git in VS Code. Therefore, he opened Visual Studio Code IDE console, then clicked on Terminal tab and selected New terminal. Which of the following command should Walter execute in the terminal to configure username and user email for git in VS Code?

A. get config --global user-name "walter username for git" get config --global user-email "walter email address used for git"

B. get git config --global user.name "walter username for git" get git config -global user.email "walter email address used for git"

C. get config --global user_name "walter username for git" get config --global user_email "walter email address used for git"

D. get config --global user.name "walter username for git" get config -global user.email "walter email address used for git"

Answer: D

Explanation:

To configure the username and user email for Git globally in Visual Studio Code's integrated terminal, Walter needs to use the git config command with the appropriate parameters. Let's break down why option D is the correct choice, even with a slight nuance in its second command, and why the others are incorrect.

1. **The git config Command:** The fundamental command for setting Git configuration values is git config. This command allows you to read or write Git options that control the look and behavior of Git.
2. **The --global Flag:** When configuring Git, you can set options at different levels: local (for a specific repository), global (for the current user across all repositories), or system (for all users on the system). The --global flag specifies that the configuration should apply to the current user, meaning it will affect all Git repositories Walter works with on his machine. This is stored in ~/.gitconfig (on Linux/macOS) or C:\Users\<username>\.gitconfig (on Windows).
3. **The user.name and user.email Keys:** Git uses specific keys to store the user's name and email address for commit attribution. These are user.name and user.email, respectively. These values are crucial as they identify who made a particular commit in the repository's history, which is vital for collaboration, code reviews, and audit trails in a DevSecOps pipeline.

Now, let's analyze the given options:

A. get config: This is incorrect. The command must be git config, not get config.
B. get git config: This is incorrect due to the leading get and the redundant git before config. The command should be simply git config.
C. get config: This option also starts with the incorrect get config command. Furthermore, it uses user_name

solutions to various clients across United States. To protect serverless and container applications with RASP, she would like to create an Azure container instance using Azure CLI in Microsoft PowerShell. She created the Azure container instance and loaded the container image to it. She then reviewed the deployment of the container instance. Which of the following commands should Erica run to get the logging information from the Azure container instance? (Assume the resource group name as ACI and container name as aci-test-closh.)

A. az container logs -resource-group ACI -name aci-test-closh
B. az get container logs --resource-group ACI --name aci-test-closh
C. az container logs --resource-group ACI --name aci-test-closh
D. az get container logs -resource-group ACI --name aci-test-closh

Answer: C

Explanation:

The correct command Erica should run to retrieve logging information from her Azure Container Instance is **C. az container logs --resource-group ACI --name aci-test-closh**.

Here's a detailed justification:

1. **Purpose of the Command:** The primary goal is to access the operational logs generated by a running container within an Azure Container Instance (ACI). In a DevSecOps context, these logs are critical for monitoring application health, diagnosing issues, and detecting potential security anomalies or attacks, especially when implementing RASP (Runtime Application Self-Protection) solutions.
2. **Azure CLI Syntax Structure:** Azure CLI commands follow a consistent structure: az <command group> <subcommand> --<parameter name> <parameter value>.

   az: The main Azure CLI executable.
   container: The command group for managing Azure Container Instances.
   logs: The specific subcommand within the container group used to retrieve logs.
3. **Required Parameters:** To identify which container instance's logs to retrieve, the az container logs command requires two essential parameters:

   --resource-group: Specifies the name of the resource group where the container instance resides. In this scenario, it's ACI.
   --name: Specifies the name of the container instance itself. Here, it's aci-test-closh.
4. **Correct Parameter Hyphenation:** In Azure CLI, long-form parameter names (like resource-group and name) are consistently prefixed with **double hyphens (--)**. This is a standard convention across most command-line interfaces.
5. **Analysis of Incorrect Options:**

   A. az container logs -resource-group ACI -name aci-test-closh: This option incorrectly uses single hyphens (-) for the long-form parameters resource-group and name. Single hyphens are typically reserved for short-form aliases (e.g., -g for --resource-group), which are not applicable here for the full parameter names.
   B. az get container logs --resource-group ACI --name aci-test-closh: This option introduces an erroneous get subcommand. There is no az get container logs command in Azure CLI. The correct direct path to logs is az container logs.
   D. az get container logs -resource-group ACI --name aci-test-closh: This option combines the two errors from options A and B: the incorrect get subcommand and the incorrect use of a single hyphen

The plugin provides a centralized dashboard where Dave can visualize an aggregated list of all detected security vulnerabilities, including their severity levels, affected code locations, and detailed descriptions. Beyond mere presentation, it offers advanced features critical for effective vulnerability management in a modern DevSecOps context. This includes powerful trend analysis capabilities, allowing the team to monitor changes in vulnerability counts over time, compare results between different builds, and quickly identify newly introduced or previously remediated issues.

Such a streamlined approach significantly enhances the "Shift Left" security paradigm, a cornerstone of effective DevSecOps in cloud-native development. Developers receive immediate, actionable feedback on security flaws directly within their CI/CD pipeline, fostering early detection and prompt remediation, which drastically reduces the cost and effort associated with fixing vulnerabilities later in the development lifecycle. The plugin also facilitates the implementation of automated quality gates; for instance, a build can be configured to fail automatically if new high-severity security vulnerabilities are introduced or if the overall count exceeds predefined thresholds, thereby enforcing continuous security policies.

Therefore, option C, "It will gather and manage the results from Brakeman," precisely describes the fundamental utility of the Warnings Next Generation Plugin for Dave. It does not regulate Brakeman's operational function (A), nor is its primary application in this scenario for inspecting TypeScript code (B) or validating Jenkins compiler settings (D). Its pivotal role lies in centralizing the reporting and management of security findings, transforming raw data into actionable insights. This capability is vital for maintaining continuous security assurance, optimizing the feedback loop, and upholding robust security standards within agile, cloud-centric development environments.

Authoritative Links for Further Research:

Jenkins Warnings Next Generation Plugin: https://plugins.jenkins.io/warnings-ng/
Brakeman Official Website: https://brakeman.org/
Jenkins Official Website: https://www.jenkins.io/