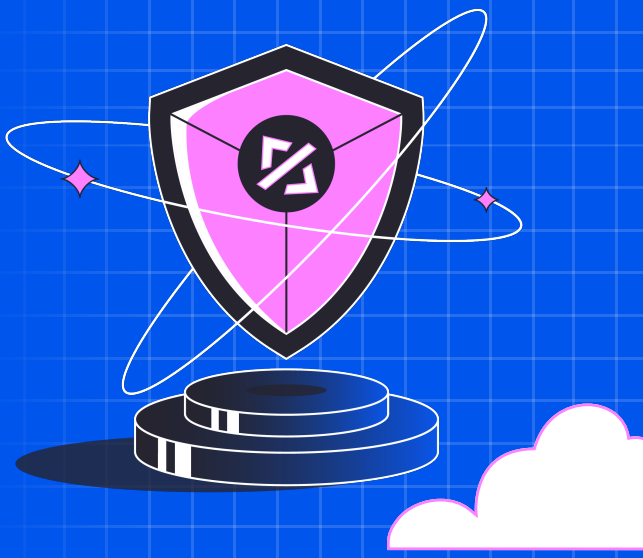
DevOps Security Best Practices

This cheat sheet offers a deep dive into secure coding, infrastructure security, and vigilant monitoring and response. A blend of industry insights and practical experience, each section combines theoretical knowledge with hands-on tips for application.

As you explore this guide, you'll uncover essential security practices, from input validation to zero-trust architecture, empowering you to create a secure DevOps environment. Let's jump right in.



Secure coding practices

1 Input validation

Input validation is the frontline defense against injection attacks, such as SQL injection and cross-site scripting (XSS). It involves validating and sanitizing user inputs to ensure they conform to the expected format before your application processes them.

Implementing stringent input validation can prevent a wide array of security vulnerabilities, like unauthorized and malicious inputs, safeguarding your application from potential exploits and data breaches.

Actionable items

- **Always validate, sanitize, and escape user input:**
  Employ regular expressions to validate input formats rigorously, utilize sanitization functions to remove unwanted characters, and apply escape functions to secure output.

```
# PHP code snippet demonstrating input validation using regular
expressions

if (preg_match("/^[a-zA-Z0-9_]{1,}$/", $username)) {

// Valid username

} else {

// Invalid username

}
```

- **Use libraries and frameworks that offer built-in input validation:**

  Frameworks with built-in validation functionalities streamline the validation process and enhance security.

2 Avoid hardcoded secrets

Hardcoding secrets, such as API keys and database credentials in the codebase, is a dangerous practice because it exposes these sensitive details to anyone who has access to the code, making the system vulnerable to unauthorized access and data breaches.

Actionable items

- **Use secrets management tools like HashiCorp Vault or AWS Secrets Manager:**

  These platforms provide safe storage and administration of sensitive information, ensuring unified oversight and the ability to produce secrets dynamically.

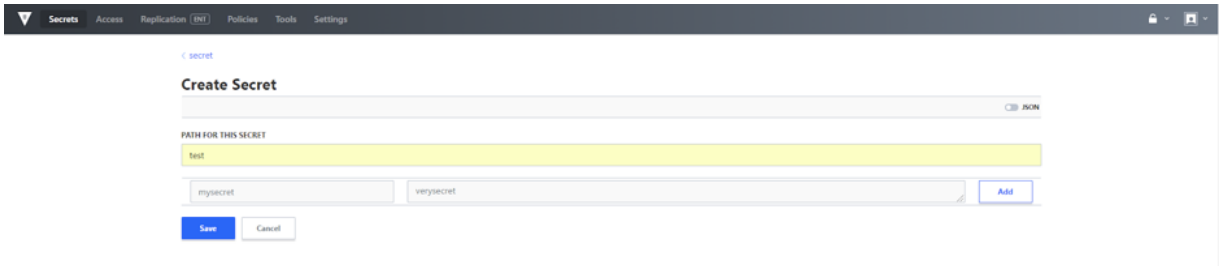


Figure 1: Vault dashboard (Source: Vault)

- **Integrate security scanners in CI/CD pipelines:**

  Integrate tools like GitGuardian or TruffleHog into your CI/CD pipelines to automatically scan for hardcoded secrets in the codebase.

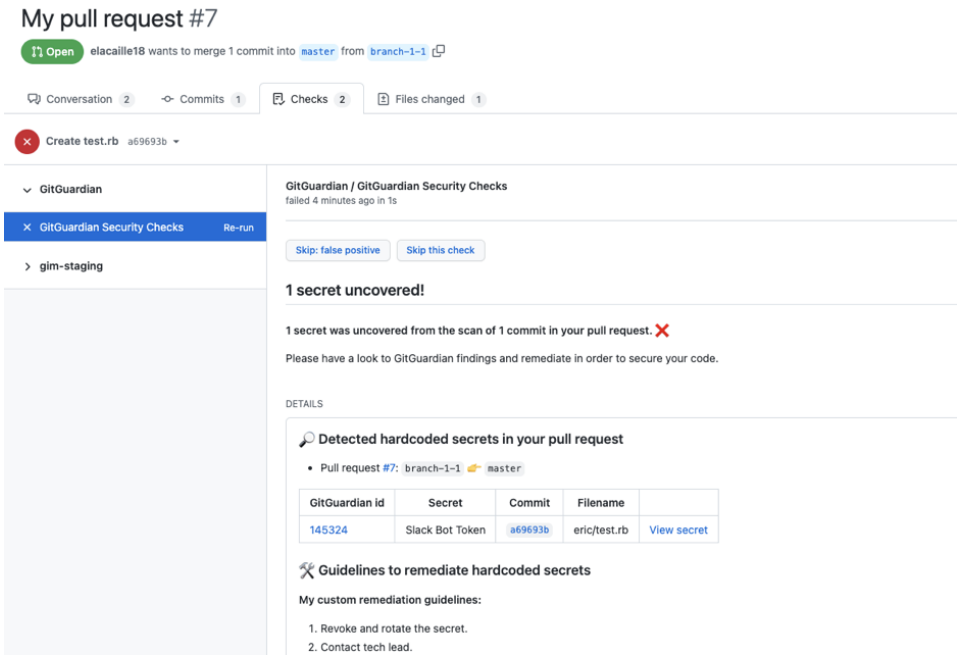


Figure 2: GitGuardian in pull-request flow (Source: GitGuardian Docs)

- **Regularly rotate and audit secrets:**
  Establish routines for rotating secrets periodically and auditing access logs to detect any unauthorized access or anomalies.

```
# Shell command to rotate secrets using AWS CLI

aws secretsmanager rotate-secret --secret-id MyAwesomeSecret
```

- **Leverage ephemeral secrets:**
  Instead of using long-lived secrets, consider using ephemeral secrets that have a short lifespan. This limits the window of opportunity for an attacker, even if they manage to access the secret. Ephemeral secrets can be created and destroyed on the fly based on the specific use case, ensuring that secrets don't linger longer than necessary.

```
# Python code to create a secret and delete it afterwards

import boto3

from datetime import datetime, timedelta

# Initialize the Secrets Manager client

client = boto3.client('secretsmanager')

# Create a temporary secret

response = client.create_secret(

Name='MyEphemeralSecret',

SecretString='ThisIsMyTemporarySecretValue'

)

secret_arn = response['ARN']

# Use the secret in your application

# Schedule the deletion of the secret after it's used - 1 hour
later

delete_date = datetime.utcnow() + timedelta(hours=1)

client.schedule_secret_deletion(

SecretId=secret_arn,

ForceDeleteWithoutRecovery=True,

DeletionDate=delete_date

)
```

Infrastructure security

1 Immutable infrastructure

Immutable infrastructure is a paradigm where infrastructure is replaced rather than updated, meaning once a server is deployed, it is never modified, and any changes necessitate the deployment of a new server. This approach relies on automated tools to manage infrastructure, which ensures consistency and reliability.

By adopting an immutable infrastructure, organizations can avoid configuration drifts—the situation where servers and their settings start to differ or deviate from the original setup over time, often due to manual changes or interventions.

Actionable items

- **Implement infrastructure as code (IaC):**

  Utilize IaC tools like Terraform or AWS CloudFormation to define and provision infrastructure. IaC supports the principles of immutability by ensuring that infrastructure can be easily versioned, replicated, and destroyed.

```
# Terraform script example to create an immutable AWS EC2 instance

resource "aws_instance" "example" {

ami = "ami-0c55b159cbfafe1f0"

instance_type = "t2.micro"

}
```

- **Automate build and deployment pipelines:**

  Leverage CI/CD tools like Jenkins or GitLab CI/CD to automate the build and deployment processes. Automation keeps each deployment consistent, repeatable, and traceable, aligning with immutable infrastructure principles.

```
# GitLab CI/CD pipeline example for building and deploying an
application

stages:

- build

- deploy

build_job:

stage: build

script: echo "Building the application"

deploy_job:

stage: deploy

script: echo "Deploying the application"
```

- **Integrate IaC scanning in IDEs and CI/CD pipelines:**

  Incorporate IaC scanning tools to analyze infrastructure definitions for potential misconfigurations. These tools can be integrated both within the integrated development environment (IDE) and at the pipeline level. Configurations can be set to block merges or deployments if identified misconfigurations pose a risk, ensuring code quality and security.

2 Network segmentation

Immutable infrastructure is a paradigm where infrastructure is replaced rather than updated, meaning once a server is deployed, it is never modified, and any changes necessitate the deployment of a new server. This approach relies on automated tools to manage infrastructure, which ensures consistency and reliability.

By adopting an immutable infrastructure, organizations can avoid configuration drifts—the situation where servers and their settings start to differ or deviate from the original setup over time, often due to manual changes or interventions.

Actionable items

- **Regularly review and update segmentation rules:**

  Employ network monitoring tools to continuously oversee network traffic so you can quickly update segmentation rules based on observed traffic patterns and potential threats.

- **Monitor inter-segment traffic:**

  Utilize network monitoring tools to analyze traffic between segments, detecting anomalies and ensuring compliance with segmentation policies.

```
# Shell command to monitor network traffic using tcpdump

tcpdump -i any -nn -s0 -c 1000
```

- **Adopt a service mesh for refined network control:**

  Implement service meshes like Istio, Consul, or Linkerd to provide more granular control and observability of inter-service communication in microservices architectures. Service meshes offer features like load balancing, service-to-service authentication, and traffic routing.

- **Monitor inter-segment traffic:**

  Utilize network monitoring tools to analyze traffic between segments, detecting anomalies and ensuring compliance with segmentation policies.

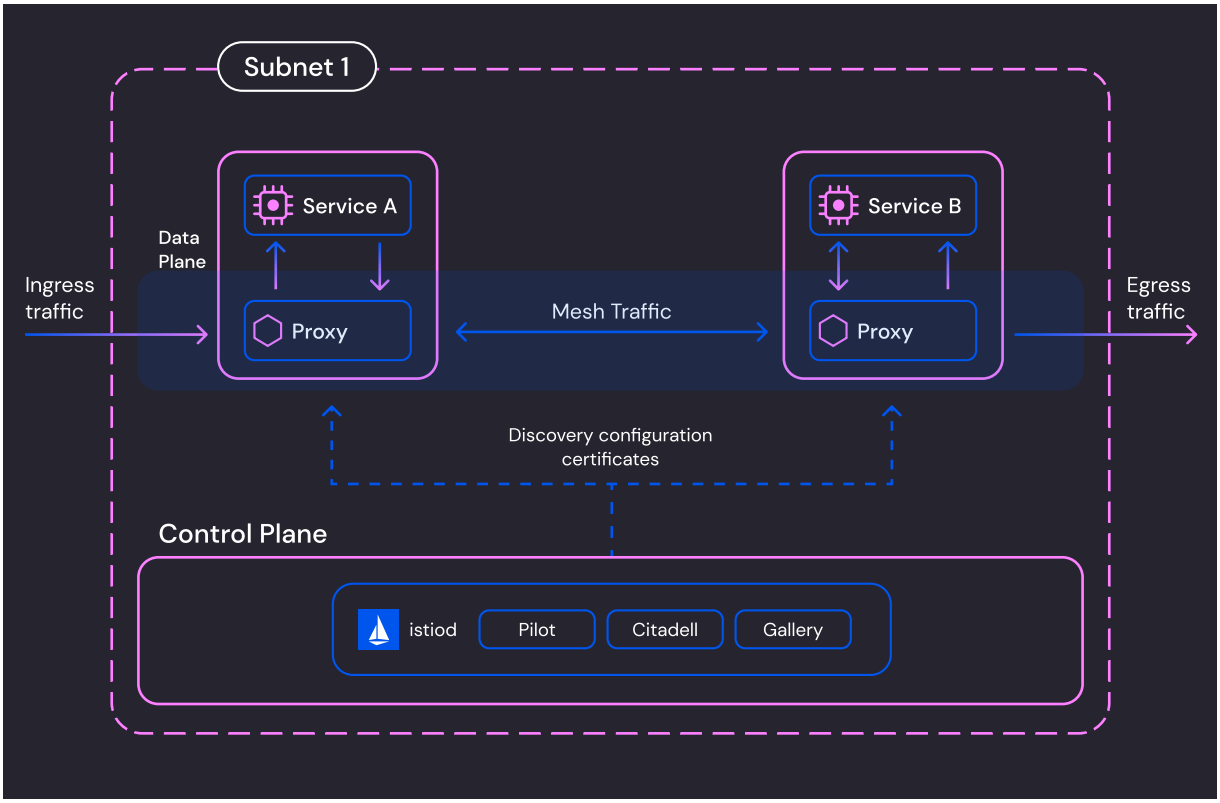


Figure 3: Istio Architecture (Source: Istio Docs)

- **Implement zero-trust architecture (ZTA) with mutual TLS (mTLS) between services:**

  Embrace a zero-trust approach, where the default state is to not trust any entity, inside or outside the network. By employing mTLS, you ensure that both parties in a communication authenticate each other and that the data transmitted between them remains confidential and tamper-proof. For more information about zero trust, see the following section.

3 Zero-trust architecture

As we've seen, zero-trust architecture (ZTA) is a security model that necessitates strict identity verification for every person and device trying to access resources on a private network, irrespective of whether they are inside or outside of the network perimeter. Adopting a zero-trust architecture significantly enhances security by reducing the risk of insider threats and preventing lateral movements in the case of breaches.

Actionable items

- **Implement identity access management (IAM) solutions:**

  Take advantage of IAM solutions to manage users and their respective access rights with precision, granting individuals only the necessary permissions for their roles.

- **Use multi-factor authentication (MFA) for all access points:**

  Implement MFA to add an additional layer of security, requiring users to authenticate using two or more verification methods, which significantly reduces the risk of unauthorized access.

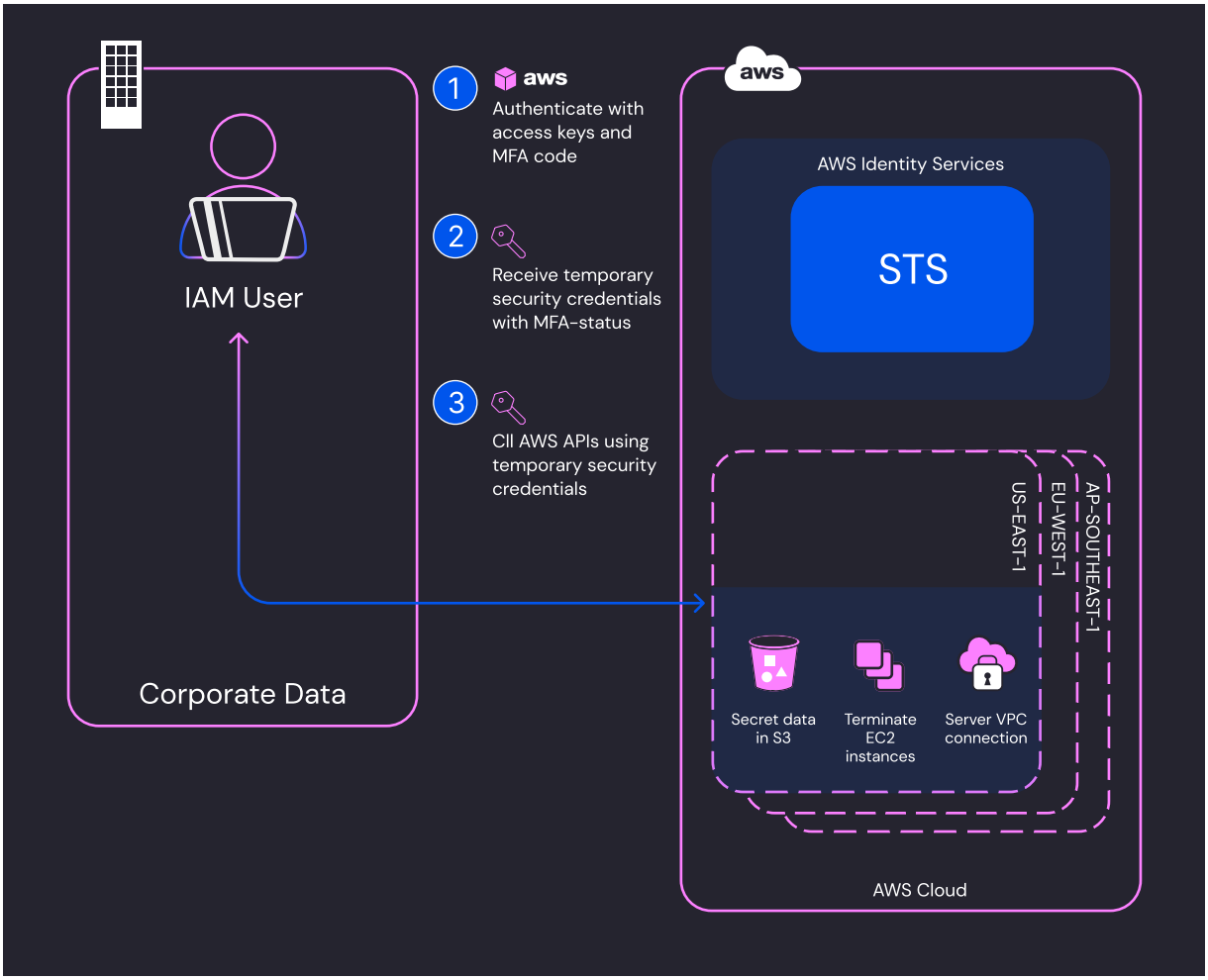


Figure 4: API access with multi-factor authentication in AWS (Source: AWS Blog)

- **Regularly update access policies:**

  Assess and modify access policies to reflect shifts in user roles, duties, and risk profiles on a regular basis.

```
# Shell command to update access policies using a policy management
tool

policy-tool update --policy-id access_policy_123 --allow-role
engineer
```

- **Implement secure remote access solutions:**

  For accessing resources securely, especially in remote or hybrid working scenarios, employ solutions like HashiCorp Boundary, Google's BeyondCorp, or AWS Secure Remote Access. These tools offer secure access to infrastructure and applications without exposing them directly to the internet, aligning with the principles of zero trust.

Monitoring and response

1 Real-time monitoring

Real-time monitoring involves the continuous surveillance of a network to promptly detect anomalies and potential threats. With tools that analyze system metrics and logs to provide insights into the health and performance of your infrastructure, you can be sure that potential threats are neutralized right away. These tools can pinpoint specific issues like unusual traffic patterns, unauthorized access attempts, or sudden spikes in resource usage, offering a granular view of any potential vulnerabilities.

Actionable items

- **Use monitoring tools like ELK Stack, Splunk, or Grafana:**

  Once you have a comprehensive monitoring solution in place, you can get deep insights into system health through data aggregation, analysis, and visualization.

Figure 5: Grafana dashboard (Source: Grafana)

- **Set up alerts for suspicious activities:**

  Configure alerting systems to immediately notify the relevant teams when potential threats are detected, facilitating timely interventions.

```
# Example alert for AlertManager to detect requests with high
latency

groups:

- name: example

rules:

- alert: HighRequestLatency

expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5

for: 10m

labels:

severity: page

annotations:

summary: High request latency
```

2 Incident response

Incident response is a systematic method for handling the consequences of a security breach or cyberattack. This approach involves a unified strategy for detecting, containing, eliminating, recovering from, and drawing lessons from security events. This strategy is often documented in an incident-response plan that outlines the processes to follow and the roles and responsibilities of each team member. Having a well-articulated incident-response strategy means your organization can respond to security incidents swiftly and effectively, minimizing the potential damage and downtime—and helping teams learn and evolve to prevent future incidents.

Actionable items

- **Develop an incident-response plan:**

  Create a comprehensive plan detailing the actions to undertake during an incident, encompassing communication guidelines and restoration methods.

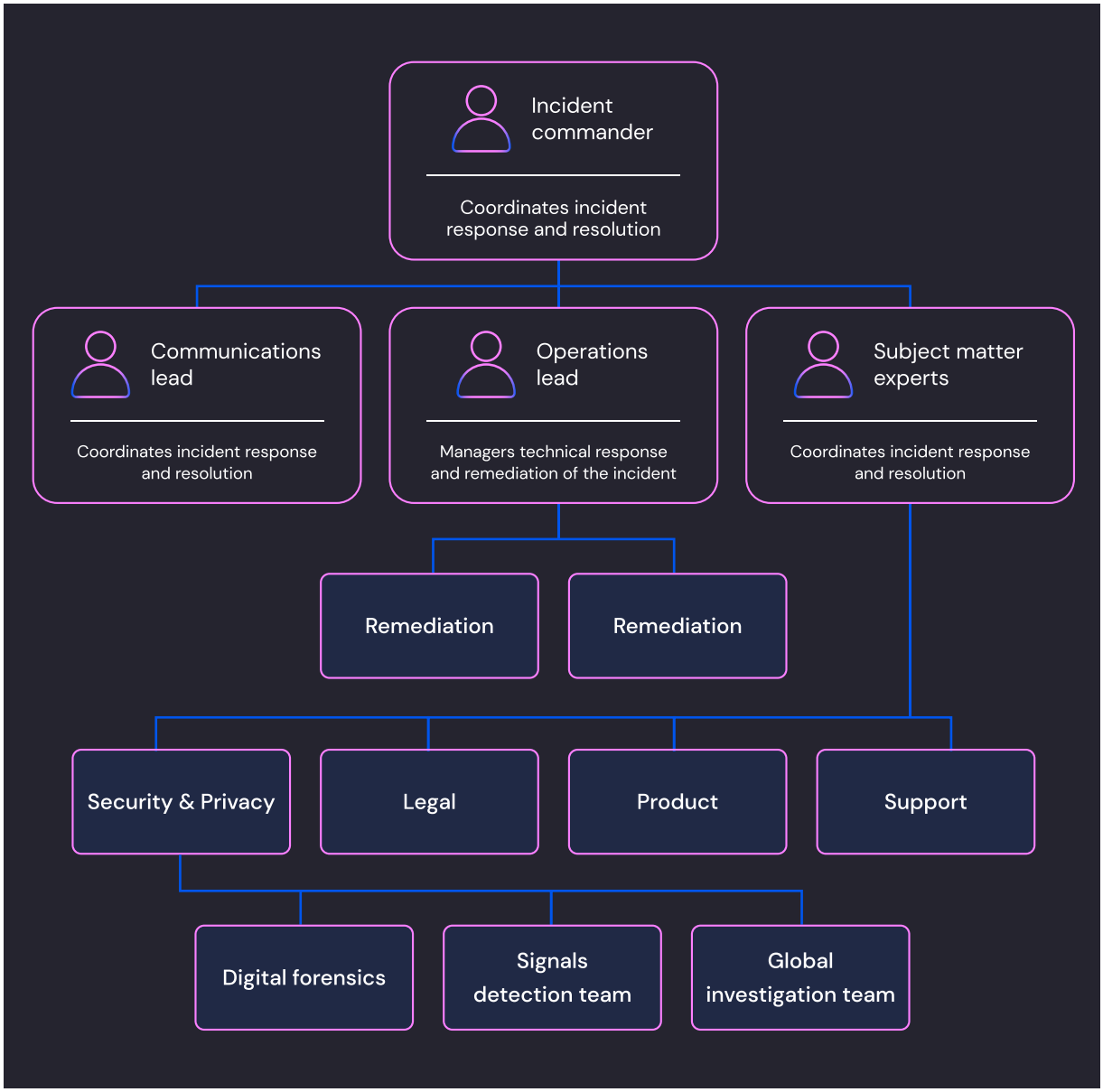


Figure 6: Incident response organization chart for GCP (Source: GCP Docs)

- **Conduct regular incident-response drills:**

  Simulate potential security incidents to train your teams to respond effectively. Afterwards, identify areas for improvement in the incident-response strategy and update your plan accordingly.

- **Integrate chaos engineering practices:**

  Incorporate chaos engineering exercises into your operational routines. Deliberately introduce controlled disruptions to test and improve the resilience of your applications and infrastructure. By simulating potential issues, you'll ensure your systems can support incidents when they arise, fostering a proactive security posture.

3 Feedback loop

The feedback loop in DevOps security refers to the iterative process of learning and improving from past incidents. It involves analyzing breaches post-resolution to understand the root causes and integrating the lessons learned into the DevOps processes. Implementing a feedback loop not only enhances your organization's resilience but also creates a culture of continuous improvement, where your security posture adapts dynamically based on real-world experiences.

Actionable items

- **Hold post-incident reviews:**

  After resolving an incident, conduct a review to analyze what happened, what was done well, and what could be improved. Document your findings for future reference.

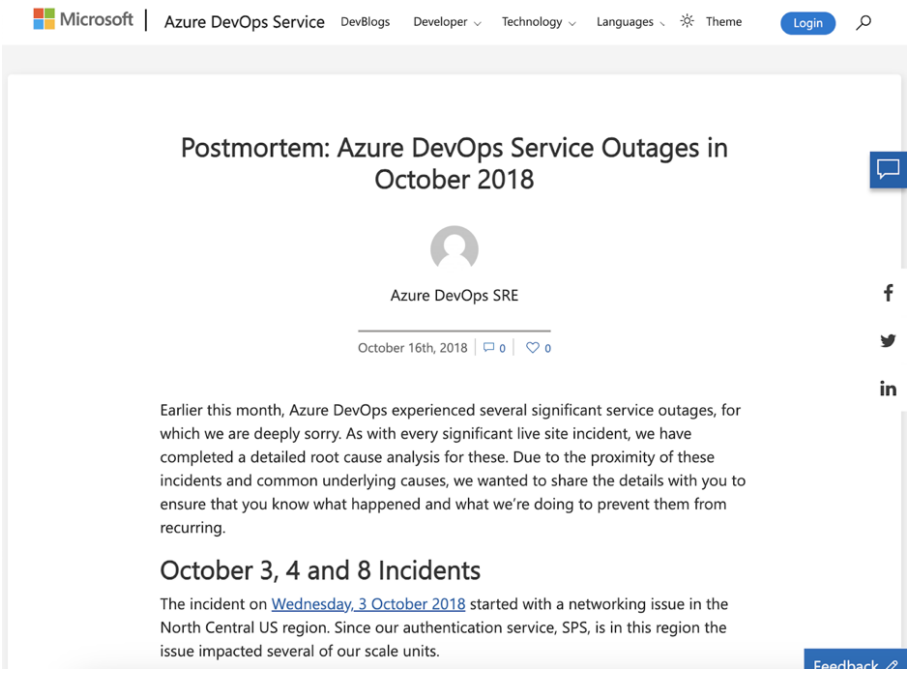


Figure 7: Post-mortem report from Azure (Source: Azure Dev Blog)

- **Update processes based on lessons learned:**

  Utilize the insights gained from post-incident reviews to update existing processes and strategies. By doing so, you not only foster a proactive and adaptive security posture but also enhance threat-detection capabilities, improve response times, and strengthen overall system resilience.

Conclusion

The strategies and best practices outlined in this cheat sheet will help you adopt a proactive approach to security so you can stay ahead of potential threats. But to truly elevate your organization's DevOps security, consider integrating Wiz into your cloud security operations. Wiz is a revolutionary platform designed for comprehensive cloud security, offering a range of features, including:

- **Cloud misconfiguration:** Wiz connects to your cloud environment and gives you complete visibility and actionable context on your most critical misconfigurations, so your teams can proactively and continuously improve your cloud security posture.
- **Vulnerability management:** Wiz gives complete visibility across containers, Kubernetes, and cloud environments in minutes without agents. Use the power of the security-graph to analyze and prioritize risk with complete context. Detect malicious behavior occurring in Kubernetes clusters in real-time for rapid responses.
- **IaC scanning:** Wiz detects vulnerabilities, secrets, and misconfigurations in IaC templates, container images, and VM images early in the development workflow.
- **Threat detection and response:** Wiz continuously monitors your cloud workloads for suspicious activity and collects intelligence from cloud providers to proactively detect and respond to unfolding threats.

By streamlining your cloud security with Wiz, you get not only a centralized platform but also a tool that offers ruthless risk prioritization and unparalleled visibility across your cloud infrastructure. Schedule a demo with Wiz to delve deeper into how we can simplify DevSecOps for your organization.