4. Security and compliance

└ 4.3 Secret and sensitive data management

└ 4.3.4 Prevent sensitive data leakage in pipelines

---

1. What are the primary sources of sensitive data leakage in pipelines?
2. How do you securely manage secrets in Azure Pipelines?
3. How do you securely manage secrets in GitHub Actions?
4. What logging configurations help avoid data exposure?
5. How can environment variables be secured in pipelines?
6. How do you enforce secret masking in Azure Pipelines?
7. What are best practices for reviewing pipeline YAML for leaks?
8. How does GitHub detect and prevent secret exposure automatically?
9. What tools can scan pipelines for secrets or credentials?
10. How do you ensure sensitive data is not logged by custom scripts or tools?

---

1. What are the primary sources of sensitive data leakage in pipelines?

- Hardcoded secrets,
- unmasked output logs,
- exposed environment variables,
- misconfigured scripts,
- or use of unsecured third-party tools.

---

2. How do you securely manage secrets in Azure Pipelines?

Use Library variable groups with secret variables, or link *Azure Key Vault*. Access them via `$(secretName)` and ensure secret masking is enabled.

---

3. How do you securely manage secrets in GitHub Actions?

Store secrets in *Settings → Secrets and variables → Actions*. Access using `${{ secrets.MY_SECRET }}`. They're automatically masked in logs.

---

4. What logging configurations help avoid data exposure?

- Disable debug logging when not needed,
- avoid echo on secret variables,
- and never write secrets to stdout or log files.

---

5. How can environment variables be secured in pipelines?

- Use runtime injection from secret stores, never hardcode.
- In *Azure*, define as secret variables.
- In *GitHub*, use secrets and avoid exporting them to global env.

---

6. How do you enforce secret masking in Azure Pipelines?

Mark variables as "*Keep this value secret*" when defining. *Azure* auto-masks them in logs if exact string match is found during execution.

7. What are best practices for reviewing pipeline YAML for leaks?

- Scan for hardcoded strings resembling keys/passwords
- Use templates to centralize secret usage
- Avoid inline scripts containing credentials

---

8. How does GitHub detect and prevent secret exposure automatically?

GitHub Advanced Security performs **secret scanning**. If a known pattern (e.g., AWS keys) is pushed, it alerts and can revoke the secret via partner integrations.

---

9. What tools can scan pipelines for secrets or credentials?

- GitHub Advanced Security (Code/secret scanning)
- Microsoft Defender for DevOps
- Open-source tools: truffleHog, gitleaks

---

10. How do you ensure sensitive data is not logged by custom scripts or tools?

- Sanitize script output,
- use set +x in bash,
- avoid printing env variables,
- redirect logs,
- and validate tools do not expose variables implicitly.