Gitleaks Periodic Scan

# Overview:

This document outlines the setup, execution, and importance of the Gitleaks Periodic Scan pipeline in Azure DevOps. This pipeline is crucial for ensuring the security of all repositories within a project by automatically scanning for secrets and sensitive data on a weekly basis. The results of these scans help identify potential security vulnerabilities that must be addressed immediately by the respective repository owners.

# 1. Purpose:

The Gitleaks Periodic Scan pipeline is designed to enhance security across all repositories in the project by regularly scanning for any exposed secrets or sensitive information. By identifying such leaks early, the team can take immediate action to mitigate risks, ensuring the integrity and confidentiality of the project.

# 2. Pipeline Workflow:

- \*\*Checkout Repositories:\*\* The pipeline checks out all repositories within the Azure DevOps project. This step is essential because Gitleaks needs access to the repository’s files and commit history to perform a thorough scan.  
- \*\*Gitleaks Scan:\*\* Each repository is scanned using Gitleaks, a widely adopted tool for detecting secrets such as API keys, tokens, passwords, and other sensitive information in Git repositories. Regular scans ensure that any newly introduced secrets are identified promptly.  
- \*\*Report Generation:\*\* A detailed report is generated for each repository, listing any secrets or sensitive information found. These reports are critical for tracking and addressing potential vulnerabilities.  
- \*\*Scheduling:\*\* The pipeline is configured to run every weekend, ensuring that the scans are conducted regularly without requiring manual intervention.

# 3. Detailed Pipeline Configuration:

## 3.1 Checkout Step

\*\*Description:\*\* The pipeline begins by checking out each repository in the project. This step involves pulling the latest code and commit history from the remote repository to the local environment where the scan will be conducted.

\*\*Why It’s Important:\*\*

- \*\*Comprehensive Scanning:\*\* By checking out the entire repository, Gitleaks can scan not only the current state of the code but also the commit history. This is crucial because secrets can often be accidentally committed and then removed in later commits, but they still exist in the history.  
- \*\*Full Coverage:\*\* Ensures that every file and branch in the repository is accessible for scanning, providing full coverage and minimizing the chances of missing any secrets.

## 3.2 Gitleaks Scan Step

\*\*Description:\*\* After the repository is checked out, Gitleaks runs a scan across the codebase. It checks for patterns that match known secret types, such as API keys, passwords, or tokens.

\*\*Why It’s Important:\*\*

- \*\*Automated Security Check:\*\* Gitleaks automates the detection of sensitive information, reducing the manual effort required for code reviews and increasing the chances of catching hidden secrets.  
- \*\*Proactive Protection:\*\* Regular scans help in identifying issues before they can be exploited, thus maintaining the security posture of the project.

## 3.3 Report Generation and Publishing

\*\*Description:\*\* For each repository, Gitleaks generates a JSON report summarizing the results of the scan. This report includes details of any potential leaks, such as the type of secret, the location in the code, and the commit it was introduced in.

\*\*Why It’s Important:\*\*

- \*\*Actionable Insights:\*\* The reports provide actionable insights that the team can use to address any security issues. They allow the team to quickly identify the specific areas in the codebase that need attention.  
- \*\*Audit Trail:\*\* The reports serve as an audit trail, documenting what was found during each scan and the subsequent actions taken.

\*\*Action Required:\*\*  
- \*\*Immediate Remediation:\*\* The respective repository owners are required to fix any leaks identified in the report immediately. Failure to do so could result in significant security vulnerabilities.  
- \*\*Whitelisting:\*\* In case of false positives, review the findings carefully. If necessary, whitelist the items that are not actual leaks but be cautious to ensure that only legitimate false positives are ignored.

# 4. Recommendations for Remediation and Whitelisting:

## 4.1 Remediation Process

\*\*Immediate Action:\*\* For any secrets identified in the reports, immediate action should be taken. This includes removing the secret from the codebase and revoking any exposed credentials. Use environment variables or secret management tools (e.g., Azure Key Vault) to store sensitive data securely.

\*\*Commit History Review:\*\* If a secret is found in the commit history, consider using tools like Git filter-branch or BFG Repo-Cleaner to completely remove the secret from the history.

\*\*Secret Rotation:\*\* Ensure that any exposed credentials are rotated immediately to prevent unauthorized access.

## 4.2 Whitelisting Known Secrets

\*\*False Positives:\*\* Occasionally, Gitleaks might flag strings that look like secrets but are actually harmless (e.g., test data). In such cases, these can be whitelisted to prevent them from being flagged in future scans.

\*\*How to Whitelist:\*\*

- Modify the Gitleaks configuration file to include the patterns or files to be ignored.  
- Use the --config flag with a custom configuration file when running Gitleaks to apply these whitelisting rules.  
\*\*Important Note:\*\* Be cautious when whitelisting to ensure that only false positives are ignored and no real secrets are accidentally excluded from the scan.

# 5. Example Pipeline Configuration:

trigger: none  
   
schedules:  
 - cron: "0 0 \* \* 6" # Runs every Saturday at midnight  
 displayName: Weekly Gitleaks Scan  
 branches:  
 include:  
 - main  
 always: true  
  
pool:  
 vmImage: 'ubuntu-latest'  
  
steps:  
- checkout: self  
   
- script: |  
 # Iterate over all repositories  
 for repo in $(az repos list --project "YourProjectName" --query "[].name" -o tsv); do  
 echo "Checking out $repo"  
 git clone https://your-ado-instance/YourProjectName/\_git/$repo  
 cd $repo  
   
 # Run Gitleaks scan  
 gitleaks detect --source=. --report=gitleaks-report-$repo.json  
   
 # Go back to the main directory  
 cd ..  
 done  
 displayName: 'Run Gitleaks Scan on All Repositories'

Notes:  
- Replace YourProjectName with the actual name of your Azure DevOps project.  
- Ensure that Gitleaks is installed and available in the pipeline environment.

# 6. Conclusion:

The Gitleaks Periodic Scan pipeline is a vital tool in maintaining the security of your code repositories. By regularly scanning for secrets and ensuring prompt remediation of any findings, the team can effectively minimize the risk of sensitive data exposure. This process not only protects the project but also aligns with best practices for secure software development.

\*\*Action Items for the Team:\*\*

- Regularly review the reports generated by Gitleaks and take immediate action on any findings.  
- Educate team members on the importance of not committing secrets to the repository.  
- Continuously improve the pipeline by refining whitelisting rules and updating Gitleaks to the latest version.