Laboratorio: Detección y Remediación de Secretos con Gitleaks

Autores:
SEBASTIAN CAMILO CISNEROS RICARDO TOBAR
ALEXANDER MORENO

UNIREMINGTON IPIALES — Ingeniería de Sistemas

Ruta de trabajo (PowerShell): C:\Users\SEBASTIAN\repo-lab

Índice

1. Resumen

Se realizó un laboratorio práctico para detectar y remediar secretos (API keys, contraseñas, tokens) en un repositorio Git local utilizando Gitleaks (v8.28.0) y BFG Repo-Cleaner. Se generaron reportes, se aplicaron medidas inmediatas y se limpió el historial. Este documento contiene todos los comandos, archivos y evidencias para entrega.

2. Estructura del repositorio (creada)

Ruta/Archivo	Descripción
C:\Users\SEBASTIAN\repo-lab\README.md	Descripción del trabajo y pasos de instalación
C:\Users\SEBASTIAN\repo-lab\slides\presentaci	Presentación (PDF)
C:\Users\SEBASTIAN\repo-lab\demo\demo-log.t	Salida de Gitleaks (consola) - anonimizada
C:\Users\SEBASTIAN\repo-lab\demo\demo-outp	Reporte JSON de la demo
C:\Users\SEBASTIAN\repo-lab\lab\README.md	Requisitos y pasos para reproducir el lab
C:\Users\SEBASTIAN\repo-lab\lab\prueba.txt	Archivo con secreto de prueba (falso)
C:\Users\SEBASTIAN\repo-lab\lab\script_prepara	Script PowerShell para crear archivos de prueba
C:\Users\SEBASTIAN\repo-lab\report.pdf	Informe técnico (máx. 2 páginas)
C:\Users\SEBASTIAN\repo-lab\mitigation.md	Controles y reglas de mitigación
C:\Users\SEBASTIAN\repo-lab\.gitignore	Excluir archivos sensibles reales

3. Comandos ejecutados (paso a paso) - PowerShell

Inicializar repositorio

```
mkdir repo-lab
cd repo-lab
git init
```

Crear archivos de prueba

```
echo API_KEY=MI_SECRET_DE_PRUEBA_123456 > config.txt
echo DB_PASS=contraseña_prueba > secret.txt
```

Configurar identidad Git (si solicita)

```
git config --global user.name "SEBASTIAN CAMILO CISNEROS"
git config --global user.email "sebastian@example.com"
```

Agregar y commitear

```
git add .
git commit -m "Primer commit: agrego secretos de prueba"
```

Verificar gitleaks version

```
.\gitleaks.exe --version
```

Escaneo workspace (archivos actuales)

```
.\gitleaks.exe detect --source . --no-git --report-format json --report-path demo\demo-output.js
```

Escaneo historial (commits)

```
.\gitleaks.exe detect --source . --report-format json --report-path demo\report_history.json
```

Crear archivo de prueba (prueba.txt)

```
echo PASSWORD=PRUEBA_SECRETA_12345 > lab\prueba.txt
git add lab\prueba.txt
git commit -m "prueba: agrego archivo con contraseña de demostracion"
```

Generar reporte y abrir

```
notepad demo\demo-output.json
```

Preparar BFG: archivo con secretos a eliminar

```
echo PRUEBA_SECRETA_12345 > passwords-to-remove.txt
```

Clonar mirror y ejecutar BFG

```
git clone --mirror C:\Users\SEBASTIAN\repo-lab repo-lab-mirror.git
cd repo-lab-mirror.git
java -jar C:\tools\bfg.jar --replace-text ..\passwords-to-remove.txt
git reflog expire --expire=now --all
git gc --prune=now --aggressive
```

Verificar eliminación

```
git log --all -S "PRUEBA_SECRETA_12345"
```

4. Resultados (ejemplo de reporte JSON detectado)


```
[
  {
    "RuleID": "generic-api-key",
    "Description": "Detected a Generic API Key...",
    "StartLine": 569,
    "Match": "DB_PASSWORD=8ae31cacf141669ddfb5da",
    "File": "README.md"
  },
  {
    "RuleID": "sidekiq-secret",
    "Description": "Discovered a Sidekiq Secret...",
    "StartLine": 51,
    "Match": "BUNDLE_ENTERPRISE__CONTRIBSYS__COM=cafebabe:deadbeef",
    "File": "README.md"
  }
]
```


Además se generó demo\demo-log.txt con la salida de consola resumida (ej: 'WRN leaks found: 8').

5. Mitigación y limpieza de historial (BFG)

Pasos realizados:

1) Eliminar archivo del working tree y actualizar .gitignore

```
git rm lab\prueba.txt
echo lab\prueba.txt >> .gitignore
git add .gitignore
git commit -m "remediacion: elimino prueba.txt y actualizo .gitignore"
```

2) BFG: reemplazar/eliminar cadenas en historial

```
git clone --mirror C:\Users\SEBASTIAN\repo-lab repo-lab-mirror.git
cd repo-lab-mirror.git
java -jar C:\tools\bfg.jar --replace-text ..\passwords-to-remove.txt
git reflog expire --expire=now --all
git gc --prune=now --aggressive
```

3) Verificar con gitleaks y búsquedas en el repo

```
.\gitleaks.exe detect --source . --no-git --report-format json --report-path demo\report_after.j
Select-String -Path * -Pattern "PRUEBA_SECRETA_12345" -SimpleMatch -List
```

6. Prevención y monitoreo

Medidas técnicas y organizativas recomendadas:

• Usar Secret Managers: HashiCorp Vault, AWS Secrets Manager, Azure Key Vault.

• Integrar Gitleaks en CI (GitHub Actions/GitLab CI) para bloquear PRs con secretos.

• Pre-commit hooks locales que ejecuten Gitleaks antes de cada commit.

• Configurar EDR/IDS/SIEM para correlacionar alertas y detectar exfiltración.

• Aplicar AppLocker/SRP para restringir ejecución de binarios no autorizados.

7. Archivos incluidos en la entrega (subir a GitHub)

• README.md (documentación del trabajo)

• slides/presentacion.pdf (presentación)

• demo/demo-log.txt (salida consola, anonimizada)

• demo/demo-output.json (reporte JSON)

• lab/README.md (requisitos y pasos)

• lab/prueba.txt (archivo de prueba)

• lab/script_preparacion.ps1 (script de creación de pruebas)

• report.pdf (informe técnico, 2 páginas)

• mitigation.md (controles y políticas)

• .gitignore

8. Anexos: archivos de configuración y hooks

gitleaks.toml (regla personalizada para detectar PRUEBA_*)

```
[[rules]]
id = "evidence-test-key"
description = "Detecta claves de prueba usadas en laboratorio"
regex = '''(PRUEBA_[A-Z0-9_]+|EVIDENCIA_[A-Z0-9_]+)'''
tags = ["test","internal"]
```

Pre-commit hook (archivo .git/hooks/pre-commit)

```
#!/bin/sh
# Verifica con gitleaks antes de permitir commit
./gitleaks.exe detect --source . --no-git
if [ $? -ne 0 ]; then
  echo "Gitleaks detectó secretos. Abortando commit."
  exit 1
fi
exit 0
```

GitHub Actions (gitleaks) - workflow ejemplo (.github/workflows/gitleaks.yml)

```
name: gitleaks-scan
on: [push, pull_request]
jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Run gitleaks
        uses: gitleaks/gitleaks-action@v2
        with:
          args: detect --source . --report-format json --report-path gitleaks-report.json
      - name: Upload report
        uses: actions/upload-artifact@v4
        with:
          name: gitleaks-report
          path: gitleaks-report.json
```