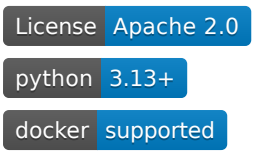
Security Scanner

A comprehensive, modular security scanning service that integrates multiple open-source security tools for Docker images, Git repositories, Kubernetes manifests, and Terraform code.



Overview

The Security Scanner is a YAML-driven DevSecOps tool that orchestrates multiple security scanners to provide comprehensive security analysis across different target types. It's designed for integration into CI/CD pipelines and supports parallel execution, multiple output formats, and extensive configuration options.

Features

🛡️ Comprehensive Security Coverage

- **Vulnerability Scanning**: Trivy, Grype for container and dependency vulnerabilities
- **SAST (Static Analysis)**: Semgrep for multi-language security analysis
- **Container Security**: Dockle for Docker image security, Hadolint for Dockerfile linting
- **Infrastructure as Code**: Checkov for Terraform/CloudFormation/Kubernetes security
- **Secret Detection**: TruffleHog and GitLeaks for credential scanning
- **SBOM Generation**: Syft for Software Bill of Materials
- **Policy Testing**: Conftest for Open Policy Agent (OPA) policy validation

🚀 DevSecOps Ready

PROF

- **YAML Configuration**: Simple, declarative configuration files
- **CI/CD Integration**: Pre-built templates for GitHub Actions, GitLab CI, Jenkins
- **Multiple Output Formats**: JSON, HTML, SARIF, XML reports
- **Parallel Execution**: Configurable parallel scanning for performance
- **Executive Summaries**: High-level security reports for management
- **Severity Filtering**: Configurable severity thresholds per scanner

🎯 Target Support

- **Docker Images**: Scan container images for vulnerabilities and misconfigurations
- **Git Repositories**: Full source code security analysis
- **Kubernetes Manifests**: Security validation for K8s deployments
- **Terraform Code**: Infrastructure security best practices
- **Filesystem Paths**: General file and directory scanning

Quick Start

🐳 Docker Usage (Recommended)

The security scanner is designed to run as a Docker container with all security tools pre-installed (~2GB image). This approach eliminates dependency management and ensures consistent, reproducible scans across different environments.

Docker Run Commands

```
# Pull the latest image
docker pull msrashed/security-scanner:latest

# Generate and run a basic scan
docker run --rm \
  -v $(pwd):/workspace \
  -v $(pwd)/security-reports:/app/reports \
  msrashed/security-scanner:latest \
  sh -c "security-scanner --generate-template basic-scan >
/tmp/scan.yaml && \
         sed -i 's|git_repositories: \[\]|git_repositories:
[\"/workspace\"]|' /tmp/scan.yaml && \
         security-scanner /tmp/scan.yaml"

# Scan with custom configuration
docker run --rm \
  -v $(pwd):/workspace \
  -v $(pwd)/security-reports:/app/reports \
  msrashed/security-scanner:latest \
  security-scanner /workspace/.security-scan.yaml

# Container security scan
docker run --rm \
  -v /var/run/docker.sock:/var/run/docker.sock:ro \
  -v $(pwd)/reports:/app/reports \
  msrashed/security-scanner:latest \
  sh -c "security-scanner --generate-template container-scan >
/tmp/scan.yaml && \
         sed -i 's|docker_images: \[\]|docker_images: [\"nginx:latest\",
\"node:18-alpine\"]|' /tmp/scan.yaml && \
         security-scanner /tmp/scan.yaml"

# Test with example vulnerable files
docker run --rm \
  -v $(pwd)/security-reports:/app/reports \
  msrashed/security-scanner:latest \
  sh -c "security-scanner --generate-template basic-scan >
/tmp/scan.yaml && \
         sed -i 's|git_repositories: \[\]|filesystem_paths:
[\"/app/examples/test-files\"]|' /tmp/scan.yaml && \
         security-scanner /tmp/scan.yaml"
```

PROF

Why Docker?

- ✅ **Zero Dependencies**: All 10+ security tools pre-installed
- ✅ **Consistent Results**: Same environment across all machines
- ✅ **Isolation**: Secure sandboxed execution
- ✅ **Easy CI/CD**: Simple integration with any pipeline
- ✅ **Version Control**: Pin to specific scanner versions
- ✅ **Resource Management**: Configurable memory and CPU limits

Quick Test

```
# Test the scanner with provided vulnerable examples
git clone https://github.com/your-org/security-scanner.git
cd security-scanner
# Check results in ./security-reports/
```

Local Development Installation

```
# Clone the repository
git clone https://github.com/your-org/security-scanner.git
cd security-scanner

# Install dependencies
pip install -r requirements.txt

# Generate and run a scan
python -m src --generate-template basic-scan > scan-config.yaml
python -m src scan-config.yaml
```

PROF

Configuration

YAML Configuration Example

```
scan_request:
  description: "My application security scan"

targets:
  git_repositories: ["."]
  docker_images: ["nginx:latest", "node:18-alpine"]
  kubernetes_manifests: ["./k8s/"]
  terraform_code: ["./infrastructure/"]

scanners:
  trivy:
    enabled: true
```

```
    timeout: 600
    severity_threshold: "HIGH"
  semgrep:
    enabled: true
    timeout: 900
    additional_args:
      - "--config=p/security-audit"
      - "--config=p/owasp-top-10"
  trufflehog:
    enabled: true
    severity_threshold: "HIGH"

output:
  base_dir: "security-reports"
  formats: ["json", "html", "sarif"]
  generate_executive_summary: true

execution:
  parallel_scans: true
  max_workers: 4
  fail_on_high_severity: true
```

Template Generation

The scanner provides built-in templates for common use cases:

```
# Available templates
security-scanner --list-templates

# Generate specific templates
security-scanner --generate-template basic-scan > basic.yaml
security-scanner --generate-template full-audit > comprehensive.yaml
security-scanner --generate-template container-scan > containers.yaml
security-scanner --generate-template development > dev-workflow.yaml
```

PROF

Scanner Details

Scanner	Purpose	Supported Targets	Key Features
Trivy	Vulnerability scanning	Containers, Git repos, Filesystems	CVE detection, dependency analysis, SBOM
Grype	Vulnerability scanning	Containers, Git repos	Anchore's vulnerability database
Semgrep	SAST	Git repos, Filesystems	Multi-language security rules, OWASP Top 10
Dockle	Container linting	Docker images	CIS Docker Benchmark compliance

Scanner	Purpose	Supported Targets	Key Features
Hadolint	Dockerfile linting	Dockerfiles	Best practices enforcement
Checkov	IaC scanning	Terraform, K8s, CloudFormation	Infrastructure security policies
TruffleHog	Secret detection	Git repos	High-entropy secret detection
GitLeaks	Secret detection	Git repos	Git history secret scanning
Syft	SBOM generation	Containers, Git repos	Software inventory and dependencies
Conftest	Policy testing	Various	OPA/Rego policy validation

CI/CD Integration

GitHub Actions

```yaml
name: Security Scan
on: [push, pull_request]

jobs:
  security-scan:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Run Security Scanner
        run: |
          docker run --rm \
            -v ${{ github.workspace }}:/workspace \
            msrashed/security-scanner:latest \
            security-scanner /workspace/.github/security-scan.yaml
      - name: Upload Results
        uses: actions/upload-artifact@v3
        with:
          name: security-reports
          path: security-reports/
```

GitLab CI

```yaml
security-scan:
  stage: security
  image: msrashed/security-scanner:latest
  script:
    - security-scanner .gitlab-security-scan.yaml
  artifacts:
    reports:
      sast: security-reports/*.sarif
```

```
    paths:
      - security-reports/
```

Output Formats

Hierarchical Report Structure

The security scanner now generates reports in a well-organized hierarchical structure:

```
reports/
├── index.html                                  # Auto-generated main
index
├── container-scan-2025-09-01_20-04-46/         # Human-readable scan
ID
│   ├── index.html                              # Auto-generated scan
index
│   ├── summary/                                # Executive and
detailed reports
│   │   ├── executive_summary.html
│   │   └── detailed_report.html
│   ├── targets/                                # Target-specific
reports
│   │   ├── nginx/
│   │   │   ├── combined_findings.json
│   │   │   └── scanners/
│   │   │       ├── trivy.json
│   │   │       ├── grype.json
│   │   │       └── syft.json
│   │   └── ubuntu/
│   │       ├── combined_findings.json
│   │       └── scanners/
│   ├── raw-data/                               # JSON and SARIF outputs
│   │   ├── container-scan-2025-09-01_20-04-46_summary.json
│   │   ├── container-scan-2025-09-01_20-04-46_findings.json
│   │   └── container-scan-2025-09-01_20-04-46.sarif
│   └── metadata/                               # Scan metadata
│       └── scan_metadata.json
```

PROF

Report Features

- **Automatic Index Generation**: Navigation pages at all levels
- **Target Organization**: Results grouped by scan target
- **Multiple Formats**: JSON, HTML, and SARIF reports
- **Executive Summaries**: High-level security reports for management
- **Raw Data Access**: Complete scanner outputs in structured format

JSON Report

```
{
  "scan_id": "container-scan-2025-09-01_20-04-46",
  "start_time": "2025-09-01T20:04:46Z",
  "summary": {
    "total_findings": 42,
    "critical": 2,
    "high": 8,
    "medium": 15,
    "low": 17
  },
  "findings": [...]
}
```


SARIF (for GitHub/IDE integration)

Standards-compliant SARIF format for integration with GitHub Security tab and IDEs.

HTML Report

Interactive HTML reports with executive summaries, detailed findings, and remediation guidance.

Development

Project Structure

```
security-scanner/
├── src/                      # Main source code
│   ├── __init__.py          # Package initialization
│   ├── __main__.py          # CLI entry point
│   ├── cli.py               # Command-line interface
│   ├── scanner_service.py   # Main orchestration service
│   ├── core/                # Core functionality
│   │   ├── config.py        # Configuration models
│   │   ├── models.py        # Data models
│   │   ├── scan_request.py # YAML request parsing
│   │   └── exceptions.py    # Custom exceptions
│   ├── scanners/            # Individual scanner implementations
│   │   ├── base.py          # Base scanner class
│   │   ├── trivy.py         # Trivy scanner
│   │   ├── semgrep.py       # Semgrep scanner
│   │   └── ...              # Other scanners (checkov, gitleaks, etc.)
│   └── output/              # Report generation
│       ├── formatters.py    # Output formatters
│       ├── json_formatter.py  # JSON reports
│       └── sarif_formatter.py # SARIF reports
├── examples/                # Usage examples and test files
│   ├── development-workflow.yaml  # Dev workflow config
│   ├── multi-target-scan.yaml     # Multi-target example
│   └── test-files/          # Sample vulnerable code for testing
```

```
│       ├── vulnerable_javascript.js
│       └── vulnerable_python.py
├── templates/              # Configuration templates
│   ├── scan-requests/      # Scan configuration templates
│   │   ├── basic-scan.yaml
│   │   ├── full-security-audit.yaml
│   │   ├── container-scan.yaml
│   │   └── ...
│   └── ci-cd/              # CI/CD integration templates
│       ├── github-actions.yaml
│       └── gitlab-ci.yaml
├── docs/                   # Documentation
│   ├── USAGE.md           # Detailed usage guide
│   └── DEVSECOPS_ENHANCEMENT_PLAN.md  # Roadmap
├── Dockerfile              # Container image definition
├── requirements.txt        # Python dependencies
├── LICENSE                # Apache 2.0 license
└── README.md              # This file
```

Adding Custom Scanners

1. Create a new scanner class inheriting from `BaseScanner`:

```python
from .base import BaseScanner
from ..core.models import ScanResult, ScanTarget

class CustomScanner(BaseScanner):
    @property
    def name(self) -> str:
        return "custom"

    @property
    def supported_targets(self) -> List[str]:
        return ["git_repository", "filesystem"]

    def _execute_scan(self, target: ScanTarget) -> ScanResult:
        # Implement your scanning logic
        pass
```

2. Register the scanner in `src/scanners/__init__.py`:

```python
from .custom import CustomScanner

AVAILABLE_SCANNERS = {
    # ... existing scanners
    'custom': CustomScanner
}
```

Testing with Example Files

The project includes sample vulnerable code files for testing scanner functionality:

```
# Test with provided vulnerable examples
docker run --rm \
  -v $(pwd):/workspace \
  -v $(pwd)/security-reports:/app/reports \
  msrashed/security-scanner:latest \
  security-scanner --generate-template basic-scan | \
  sed 's|git_repositories: \["."\]|filesystem_paths:
["/app/examples/test-files"]|' | \
  docker run --rm -i \
  -v $(pwd)/security-reports:/app/reports \
  msrashed/security-scanner:latest \
  security-scanner /dev/stdin
```

The example files include:

- `examples/test-files/vulnerable_javascript.js` - JavaScript security vulnerabilities
- `examples/test-files/vulnerable_python.py` - Python security issues

Building the Docker Image

```
docker build -t security-scanner:latest .
```

The Docker image includes all security tools and dependencies in a single container (~2GB).

Configuration Reference

PROF

Scanner Configuration Options

Each scanner supports these common configuration options:

- `enabled`: Boolean to enable/disable the scanner
- `timeout`: Timeout in seconds for the scanner execution
- `severity_threshold`: Minimum severity level to report (CRITICAL, HIGH, MEDIUM, LOW, INFO)
- `additional_args`: Array of additional command-line arguments

Output Configuration

- `base_dir`: Base directory for all reports
- `formats`: Array of output formats (json, html, sarif, xml)
- `include_raw`: Include raw scanner output in reports
- `consolidate_reports`: Merge all scanner results into consolidated reports
- `generate_executive_summary`: Generate high-level executive summary

Execution Configuration

- `parallel_scans`: Enable parallel execution of scanners
- `max_workers`: Maximum number of parallel workers
- `fail_on_high_severity`: Exit with error code on high/critical findings

Troubleshooting

Common Issues

Permission Errors

```
# Fix file permissions for Docker
sudo chown -R $(id -u):$(id -g) ./security-reports
```

Out of Memory

```
execution:
  parallel_scans: false  # Disable parallel execution
  max_workers: 1         # Reduce workers
```

Missing Dependencies

```
# Check scanner dependencies
security-scanner --check-dependencies

# List available scanners
security-scanner --list-scanners
```

PROF

Debug Commands

```
# Validate configuration
security-scanner --validate-config my-scan.yaml

# Check system dependencies
security-scanner --check-dependencies

# Generate debug logs
security-scanner my-scan.yaml --log-level DEBUG
```

Contributing

We welcome contributions! Please see CONTRIBUTING.md for guidelines.

Development Setup

```
git clone https://github.com/your-org/security-scanner.git
cd security-scanner

# Create virtual environment
python -m venv venv
source venv/bin/activate  # or venv\Scripts\activate on Windows

# Install development dependencies
pip install -r requirements.txt
pip install -e .

# Run tests
pytest tests/

# Format code
black security_scanner/
flake8 security_scanner/
```

License

Licensed under the Apache License, Version 2.0. See LICENSE for details.

Support

- **Documentation**: See docs/USAGE.md for detailed usage examples
- **Issues**: Report bugs and feature requests via GitHub Issues
- **Discussions**: Join the discussion in GitHub Discussions

Roadmap

PROF

See docs/DEVSECOPS_ENHANCEMENT_PLAN.md for our comprehensive DevSecOps enhancement roadmap, including:

- Additional SAST scanners (language-specific)
- Policy-as-Code framework with OPA integration
- Advanced threat intelligence integration
- Machine learning-powered risk assessment
- Enterprise multi-tenant architecture

Version: 1.0.0
Python: 3.13+
Docker: Supported
License: Apache 2.0