Semgrep Scanner Implementation

Overview

I've successfully implemented **Semgrep SAST Scanner** (Phase 1.1 from the enhancement plan), adding comprehensive static application security testing capabilities to the security scanner.

What is Semgrep?

Semgrep is a fast, open-source static analysis tool that finds bugs, detects security vulnerabilities, and enforces code standards. It supports **17+ programming languages** and comes with over **2000+ security rules**.

Target Support

✅ What Semgrep Works With:

1. Git Repositories

- **Primary Use Case**: Source code analysis
- Scans entire codebases for security vulnerabilities
- Analyzes commit history and branches
- Example: `--git-repo /path/to/repo`

2. Filesystem Paths

- **Directory Scanning**: Any local directory with source code
- **File-level Analysis**: Individual source files
- Example: `--filesystem /path/to/source`

× What Semgrep Does NOT Work With:

- **Docker Images**: Semgrep cannot scan compiled containers
- **Kubernetes Manifests**: Use Checkov/KICS for K8s YAML
- **Terraform Code**: Use Checkov/KICS for IaC

Language Support

Semgrep automatically detects and analyzes:

Fully Supported Languages:

- **Python** (.py files)
- **JavaScript/TypeScript** (.js, .jsx, .ts, .tsx)
- **Java** (.java files)
- **Go** (.go files)
- **Ruby** (.rb files)

- **PHP** (.php, .phtml files)
- **C#** (.cs, .csx files)
- **Kotlin** (.kt files)
- **Scala** (.scala files)
- **And 8+ more languages**

Security Coverage

Security Rulesets Applied:

1. `p/security-audit`: General security vulnerabilities
2. `p/owasp-top-10`: OWASP Top 10 security risks
3. `p/cwe-top-25`: CWE Top 25 most dangerous weaknesses
4. **Language-specific rulesets**: Python, JavaScript, Java, etc.

Types of Issues Detected:

- **Injection Vulnerabilities**: SQL, Command, LDAP injection
- **XSS (Cross-Site Scripting)**: Reflected, stored, DOM-based
- **Authentication Issues**: Weak passwords, session management
- **Authorization Flaws**: Access control bypasses
- **Cryptographic Issues**: Weak encryption, poor key management
- **Input Validation**: Missing sanitization, unsafe deserialization
- **Code Quality**: Dead code, performance anti-patterns
- **Supply Chain**: Vulnerable dependencies usage patterns

Configuration

Default Configuration:

```
{
  "semgrep": {
    "enabled": true,
    "timeout": 600,
    "severity_threshold": "MEDIUM",
    "additional_args": [
      "--config=p/security-audit",
      "--config=p/owasp-top-10"
    ],
    "description": "Multi-language SAST tool for finding bugs and
security issues"
  }
}
```

Advanced Configuration:

```
{
  "semgrep": {
    "enabled": true,
    "timeout": 1200,
    "severity_threshold": "HIGH",
    "additional_args": [
      "--config=p/security-audit",
      "--config=p/owasp-top-10",
      "--config=p/cwe-top-25",
      "--config=p/python",
      "--config=p/javascript",
      "--max-memory=4096",
      "--max-target-bytes=1000000"
    ]
  }
}
```

Usage Examples

1. Scan a Git Repository:

```
# Scan entire repository
python -m security_scanner --git-repo /path/to/repo --enable-scanner
semgrep

# Scan with custom config
python -m security_scanner --git-repo /path/to/repo --config config.json
--enable-scanner semgrep
```

2. Scan Local Directory:

PROF

```
# Scan source code directory
python -m security_scanner --filesystem /path/to/source --enable-scanner
semgrep

# Scan only with Semgrep (disable others)
python -m security_scanner --filesystem /path/to/source --enable-scanner
semgrep --disable-scanner trivy grype
```

3. CI/CD Integration:

```
# GitHub Actions example
- name: Security Scan with Semgrep
  run: |
```

```
    python -m security_scanner \
      --git-repo . \
      --enable-scanner semgrep \
      --severity-threshold HIGH \
      --fail-on-high \
      --output-dir security-reports
```

Sample Findings Output

Example Finding Structure:

```
{
  "id": "SEMGREP-python.django.security.injection.sql.sql-injection-
using-string-formatting",
  "title": "Semgrep: Possible SQL injection",
  "description": "User input is used in a SQL query without proper
sanitization",
  "severity": "HIGH",
  "scanner": "semgrep",
  "target": "/path/to/repo",
  "location": "app/models.py:45-47",
  "remediation": "Use parameterized queries or ORM methods instead of
string formatting. See: https://owasp.org/www-
community/attacks/SQL_Injection",
  "metadata": {
    "check_id": "python.django.security.injection.sql.sql-injection-
using-string-formatting",
    "confidence": "HIGH",
    "category": "security",
    "cwe": ["CWE-89"],
    "owasp": ["A03:2021 - Injection"],
    "technology": ["django"],
    "references": ["https://owasp.org/www-
community/attacks/SQL_Injection"],
    "code_snippet": "query = f\"SELECT * FROM users WHERE id =
{user_id}\"\nresult = cursor.execute(query)"
  }
}
```

Performance Characteristics

Scan Speed:

- **Small repos** (<1k files): 10-30 seconds
- **Medium repos** (1k-10k files): 1-5 minutes
- **Large repos** (10k+ files): 5-20 minutes

Resource Usage:

- **Memory**: 1-4GB depending on codebase size
- **CPU**: Multi-threaded, scales with available cores
- **Storage**: Minimal temporary files

Integration Benefits

Why Add Semgrep to Security Scanner:

1. **Fills SAST Gap**: Previously had NO source code analysis
2. **Multi-Language**: Covers 17+ programming languages
3. **High Accuracy**: Low false-positive rate compared to other SAST tools
4. **Fast Performance**: Faster than traditional SAST scanners
5. **Rich Rule Base**: 2000+ security rules maintained by security experts
6. **Active Development**: Regularly updated with new vulnerabilities
7. **Open Source**: No licensing costs, transparent rules

Complements Existing Scanners:

- **Trivy**: Container + dependency vulnerabilities
- **Grype**: Package vulnerabilities
- **Semgrep**: **Source code vulnerabilities** ← NEW COVERAGE
- **TruffleHog/GitLeaks**: Secrets detection
- **Checkov/KICS**: Infrastructure as Code

Installation Requirements

Prerequisites:

```
# Install Semgrep
pip install semgrep

# Or via package manager
brew install semgrep  # macOS
```


Verification:

```
# Check installation
semgrep --version

# Test basic functionality
semgrep --config=auto /path/to/test/file.py
```

Comparison with Other SAST Tools

Feature	Semgrep	SonarQube	CodeQL	Checkmarx

Feature	Semgrep	SonarQube	CodeQL	Checkmarx
Languages	17+	25+	10+	20+
Speed	⚡ Fast	🐌 Slow	🐌 Slow	🐌 Very Slow
Cost	🆓 Free	💰 Paid	🆓 Free (OSS)	💰 Expensive
Accuracy	🎯 High	🎯 High	🎯 Very High	🎯 High
Setup	✅ Easy	× Complex	× Complex	× Very Complex
CI/CD	✅ Native	⚠ Plugin	✅ Native	⚠ Complex

Future Enhancements

Planned Improvements:

1. **Custom Rules**: Support for organization-specific security rules
2. **Policy Integration**: Connect with policy-as-code framework
3. **Baseline Management**: Track new vs. existing vulnerabilities
4. **Language Detection**: Enhanced language-specific optimization
5. **Performance Tuning**: Incremental scanning for large repositories

Troubleshooting

Common Issues:

1. High Memory Usage:

```
# Add memory limits
--additional-args --max-memory=2048
```

PROF

2. Timeout on Large Repos:

```
# Increase timeout
--timeout 1800  # 30 minutes
```

3. Too Many Findings:

```
# Increase severity threshold
--severity-threshold HIGH
```

4. Missing Language Support:

Check if your language is supported:

```
semgrep --help | grep -A 20 "supported languages"
```

Summary

The **Semgrep SAST Scanner** implementation successfully addresses the critical SAST gap in the security scanner, providing:

- ✅ **Multi-language source code analysis**
- ✅ **2000+ security rules**
- ✅ **Fast performance**
- ✅ **Low false positives**
- ✅ **Easy CI/CD integration**
- ✅ **Git repository and filesystem support**

This is a **high-impact enhancement** that significantly improves the security coverage of the scanning platform, moving from basic vulnerability detection to comprehensive **application security testing**.

---

*Implementation Status: ✅ **COMPLETED***
Next Phase: Policy Engine Integration