CLI Refactoring Plan: YAML-Only Configuration

Overview

This document outlines the complete refactoring plan to transform the security scanner CLI from a complex command-line argument system to a simple, YAML-only configuration approach. This refactoring will make the tool more maintainable, standardized, and suitable for CI/CD pipelines.

Current State Analysis

Existing Architecture

Based on the codebase analysis, the current system has:

Core Components:

- **CLI System** (`cli.py`): 462 lines with extensive argument parsing
- **Configuration System** (`core/config.py`): Complex dataclass-based config with legacy compatibility
- **Scanner Service** (`scanner_service.py`): Main orchestration service
- **Scanner Framework** (`scanners/`): 11 different scanners with base class architecture
- **Output System** (`output/`): Multiple formatters (JSON, HTML, SARIF)
- **Models** (`core/models.py`): Rich data models for scan targets, results, and findings

Current Scanner Support:

- **Vulnerability Scanners**: Trivy, Grype
- **SBOM Generation**: Syft
- **Container Security**: Dockle, Hadolint
- **Infrastructure as Code**: Checkov, Conftest
- **Secrets Detection**: TruffleHog, GitLeaks
- **SAST**: Semgrep (recently added)

PROF

Current Configuration Format:

```yaml
# Current config.example.yaml format
docker_images: [...]
git_repositories: [...]
kubernetes_manifests: [...]
terraform_code: [...]

trivy:
  enabled: true
  timeout: 600
  severity_threshold: "HIGH"
  additional_args: [...]
```

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

parallel_scans: true
max_workers: 4
```

Current CLI Complexity:

- 50+ command-line arguments across 8 argument groups
- Mixed configuration (CLI args + config files)
- Complex validation and override logic
- Legacy compatibility layers

Problems with Current Approach:

1. **CLI Complexity**: Too many command-line options making it hard to use
2. **Configuration Drift**: Multiple ways to configure the same thing
3. **Maintenance Burden**: Complex argument parsing and validation
4. **CI/CD Unfriendly**: Long command lines, not configuration-as-code
5. **Documentation Overhead**: Need to document both CLI and config options

Proposed Solution: YAML-Only Configuration

New Directory Structure

```
devsecops-security-scanner/
├── templates/
│   ├── scan-requests/
│   │   ├── basic-scan.yaml
│   │   ├── full-security-audit.yaml
│   │   ├── container-scan.yaml
│   │   ├── source-code-scan.yaml
│   │   ├── infrastructure-scan.yaml
│   │   └── secrets-scan.yaml
│   └── ci-cd/
│       ├── github-actions.yaml
│       ├── gitlab-ci.yaml
│       └── jenkins.yaml
├── examples/
│   ├── multi-target-scan.yaml
│   ├── development-workflow.yaml
│   └── production-audit.yaml
└── security_scanner/
    ├── core/
    │   ├── scan_request.py (new)
    │   ├── config.py (enhanced)
```

PROF

```
    │   └── template_generator.py (new)
    └── cli.py (simplified)
```

New YAML Scan Request Format

```yaml
# Standard scan request format
scan_request:
  id: "optional-custom-id"
  description: "Human readable description"
  created_by: "user@example.com"

targets:
  docker_images:
    - "nginx:latest"
    - "myapp:v1.2.3"
  git_repositories:
    - "/path/to/source/code"
  kubernetes_manifests:
    - "/path/to/k8s/"
  terraform_code:
    - "/path/to/terraform/"
  filesystem_paths:
    - "/path/to/additional/files"

scanners:
  trivy:
    enabled: true
    timeout: 600
    severity_threshold: "HIGH"
    additional_args:
      - "--dependency-tree"
      - "--list-all-pkgs"
  semgrep:
    enabled: true
    timeout: 900
    severity_threshold: "MEDIUM"
    additional_args:
      - "--config=p/security-audit"
      - "--config=p/owasp-top-10"
  # Other scanners...

output:
  base_dir: "security-reports"
  formats:
    - "json"
    - "html"
    - "sarif"
  include_raw: true
  generate_executive_summary: true
```

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

logging:
  level: "INFO"
  file: "scan.log"

# Optional metadata
metadata:
  environment: "development"
  project: "my-web-app"
  version: "v1.2.3"
```

Simplified CLI Interface

```
# New simplified CLI
security-scanner scan-request.yaml

# Utility commands only
security-scanner --list-scanners
security-scanner --check-dependencies
security-scanner --validate-config scan-request.yaml
security-scanner --generate-template basic-scan > basic-scan.yaml
security-scanner --version
```

Implementation Plan

Phase 1: Core Infrastructure (Week 1)

PROF

1.1 Create ScanRequest Model

File: `security_scanner/core/scan_request.py`

```python
@dataclass
class ScanRequestMeta:
    """Metadata for scan request"""
    id: Optional[str] = None
    description: str = ""
    created_by: Optional[str] = None
    created_at: Optional[datetime] = None

@dataclass
class ScanRequest:
    """YAML-based scan request configuration"""
    scan_request: ScanRequestMeta
    targets: ScanTargets
```

```
    scanners: Dict[str, ScannerConfig]
    output: OutputConfig
    execution: ExecutionConfig
    logging: LoggingConfig
    metadata: Dict[str, Any] = field(default_factory=dict)

    @classmethod
    def from_yaml(cls, yaml_path: str) -> 'ScanRequest':
        """Load scan request from YAML file"""

    def validate(self) -> List[str]:
        """Validate scan request and return validation errors"""

    def to_security_scan_config(self) -> SecurityScanConfig:
        """Convert to internal SecurityScanConfig format"""
```

1.2 Enhance YAML Processing

File: `security_scanner/core/config.py`

- Remove JSON support (YAML only)
- Add comprehensive YAML validation
- Enhance error messages for YAML parsing issues
- Add schema validation using `pydantic` or custom validation

1.3 Update Models

File: `security_scanner/core/models.py`

- Add metadata support to all models
- Enhance ScanSummary with request metadata
- Add template identification fields

PROF

Phase 2: CLI Simplification (Week 2)

2.1 Refactor CLI Parser

File: `security_scanner/cli.py`

```
def create_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(
        description="Security Scanner - YAML Configuration Driven",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
Examples:
  # Run scan with YAML configuration
  security-scanner scan-request.yaml

  # Generate templates
```

```
  security-scanner --generate-template basic-scan > basic-scan.yaml
  security-scanner --generate-template full-audit > full-audit.yaml

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

  # Check system
  security-scanner --list-scanners
  security-scanner --check-dependencies
"""
    )

    # Main command - YAML file
    parser.add_argument(
        "scan_request",
        nargs="?",
        help="Path to YAML scan request file"
    )

    # Utility commands
    parser.add_argument("--list-scanners", action="store_true")
    parser.add_argument("--check-dependencies", action="store_true")
    parser.add_argument("--validate-config", metavar="FILE")
    parser.add_argument("--generate-template",
                        choices=["basic-scan", "full-audit", "container-
scan",
                                 "source-code-scan", "infrastructure-
scan", "secrets-scan"])
    parser.add_argument("--version", action="version", version="Security
Scanner 2.0.0")

    return parser

def main() -> int:
    """Simplified main entry point"""
    parser = create_parser()
    args = parser.parse_args()

    # Handle utility commands
    if args.list_scanners:
        return list_scanners()
    elif args.check_dependencies:
        return check_dependencies()
    elif args.validate_config:
        return validate_config(args.validate_config)
    elif args.generate_template:
        return generate_template(args.generate_template)

    # Main scan command
    if not args.scan_request:
        parser.print_help()
        return 1
```

```
    return run_scan(args.scan_request)
```

2.2 Implement Template Generator

File: `security_scanner/core/template_generator.py`

```
class TemplateGenerator:
    """Generate YAML templates for different use cases"""

    def generate_basic_template(self) -> str:
        """Basic scan template with common scanners"""

    def generate_full_audit_template(self) -> str:
        """Comprehensive security audit template"""

    def generate_container_template(self) -> str:
        """Container-focused scanning template"""

    def generate_source_code_template(self) -> str:
        """Source code scanning template with SAST focus"""

    def generate_infrastructure_template(self) -> str:
        """Infrastructure as code scanning template"""

    def generate_secrets_template(self) -> str:
        """Secrets detection focused template"""
```

Phase 3: Template Creation (Week 3)

3.1 Basic Templates

PROF

File: `templates/scan-requests/basic-scan.yaml`

```
scan_request:
  description: "Basic security scan with essential scanners"

targets:
  docker_images: []
  git_repositories: []
  kubernetes_manifests: []
  terraform_code: []
  filesystem_paths: []

scanners:
  trivy:
    enabled: true
    timeout: 600
```

```
    severity_threshold: "MEDIUM"
  grype:
    enabled: true
    timeout: 300
    severity_threshold: "MEDIUM"
  semgrep:
    enabled: true
    timeout: 600
    severity_threshold: "MEDIUM"

output:
  base_dir: "reports"
  formats: ["json", "html"]

execution:
  parallel_scans: true
  max_workers: 2
```

3.2 Specialized Templates

Create templates for:

- **Full Security Audit**: All scanners enabled
- **Container Scan**: Trivy, Grype, Dockle, Hadolint focus
- **Source Code Scan**: Semgrep, TruffleHog, GitLeaks focus
- **Infrastructure Scan**: Checkov, Conftest focus
- **Secrets Scan**: TruffleHog, GitLeaks only

3.3 CI/CD Integration Templates

File: `templates/ci-cd/github-actions.yaml`


```
scan_request:
  description: "GitHub Actions CI/CD security scan"

targets:
  git_repositories: ["."]

scanners:
  trivy:
    enabled: true
    severity_threshold: "HIGH"
  semgrep:
    enabled: true
    severity_threshold: "HIGH"

output:
  formats: ["sarif", "json"]
```

```
execution:
  fail_on_high_severity: true
  max_workers: 2
```

Phase 4: Enhanced Features (Week 4)

4.1 Configuration Validation

- Schema validation with detailed error messages
- Target path validation
- Scanner compatibility checks
- Resource availability validation

4.2 Migration Support

```
# Migration helper (optional)
security-scanner --migrate-from-cli \
  "--docker-image nginx:latest --enable-scanner trivy grype --severity-threshold HIGH" \
  > migrated-config.yaml
```

4.3 Enhanced Logging and Reporting

- Request metadata in all reports
- Template identification in reports
- Better progress reporting with request context

Breaking Changes and Migration

What's Being Removed:

PROF

1. **All CLI target arguments**: `--docker-image`, `--git-repo`, etc.
2. **All CLI scanner arguments**: `--enable-scanner`, `--disable-scanner`, etc.
3. **All CLI output arguments**: `--output-dir`, `--format`, etc.
4. **All CLI execution arguments**: `--parallel`, `--max-workers`, etc.
5. **JSON configuration support**: YAML only

What's Being Kept:

1. **Utility commands**: `--list-scanners`, `--check-dependencies`
2. **All scanner functionality**: No changes to scanner implementations
3. **All output formats**: JSON, HTML, SARIF support maintained
4. **All configuration options**: Just moved to YAML

Migration Strategy:

1. **Template-based migration**: Provide templates for all common use cases
2. **Documentation**: Comprehensive before/after examples
3. **Gradual rollout**: Can be done as v2.0 with clear migration guide

Template Examples

Basic Web Application Scan

```yaml
scan_request:
  description: "Web application security scan"

targets:
  git_repositories: ["/workspace/webapp"]
  docker_images: ["webapp:latest"]
  kubernetes_manifests: ["/workspace/k8s/"]

scanners:
  trivy:
    enabled: true
    severity_threshold: "HIGH"
  semgrep:
    enabled: true
    severity_threshold: "MEDIUM"
    additional_args:
      - "--config=p/owasp-top-10"
  trufflehog:
    enabled: true
    severity_threshold: "HIGH"
  checkov:
    enabled: true
    severity_threshold: "MEDIUM"

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

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

Development Workflow Scan

```yaml
scan_request:
  description: "Fast development security check"

targets:
  git_repositories: ["."]
```

```yaml
scanners:
  semgrep:
    enabled: true
    timeout: 300
    severity_threshold: "HIGH"
  trufflehog:
    enabled: true
    timeout: 180
    severity_threshold: "HIGH"

output:
  base_dir: "dev-scan"
  formats: ["json"]

execution:
  parallel_scans: true
  max_workers: 2
  fail_on_high_severity: false
```

Implementation Benefits

1. Simplicity

- Single way to configure scans
- Clear, readable YAML format
- Template-driven quick start
- Self-documenting configurations

2. Maintainability

- Reduced CLI code complexity (462 → ~100 lines)
- Single configuration path
- Easier testing and validation
- Clear separation of concerns

PROF

3. CI/CD Integration

- Configuration as code
- Version controllable scan configs
- Reusable templates
- Environment-specific configurations

4. User Experience

- Template-based quick start
- Clear validation messages
- Rich metadata support
- Better error reporting

5. Standardization

- Consistent YAML format
- Standardized field names
- Clear documentation
- Industry best practices

Technical Implementation Details

Configuration Processing Flow:

```
YAML File → ScanRequest → Validation → SecurityScanConfig →
ScannerService
```

Validation Pipeline:

1. **YAML Syntax**: Valid YAML format
2. **Schema Validation**: Required fields present
3. **Target Validation**: Paths exist and accessible
4. **Scanner Validation**: Enabled scanners available
5. **Resource Validation**: Sufficient resources for configuration

Error Handling:

- Clear, actionable error messages
- Field-level validation errors
- Suggestions for common mistakes
- Template recommendations

Testing Strategy

PROF

1. Unit Tests

- ScanRequest model validation
- Template generation
- YAML parsing edge cases
- Configuration conversion

2. Integration Tests

- End-to-end with templates
- CLI command validation
- Scanner integration
- Output generation

3. Migration Tests

- All current functionality preserved
- Template coverage for common use cases
- Performance equivalence

Timeline and Milestones

Week 1: Foundation

- [ ] Create ScanRequest model
- [ ] Enhance YAML processing
- [ ] Update validation system
- [ ] Unit tests for core models

Week 2: CLI Refactoring

- [ ] Simplify CLI parser
- [ ] Implement template generator
- [ ] Update main entry point
- [ ] Integration tests

Week 3: Templates

- [ ] Create all scan request templates
- [ ] Create CI/CD integration templates
- [ ] Create example configurations
- [ ] Template validation tests

Week 4: Documentation & Polish

- [ ] Update all documentation
- [ ] Migration guide
- [ ] Performance testing
- [ ] User acceptance testing

PROF

Success Metrics

Code Quality:

- CLI code reduction: 462 → ~100 lines (78% reduction)
- Configuration paths: Multiple → Single YAML path
- Test coverage: >90% for new components

User Experience:

- Setup time: Reduced by 60% with templates
- Configuration errors: Reduced by 80% with validation
- Documentation clarity: Single format to document

Maintainability:

- Argument parsing complexity: Eliminated
- Configuration drift: Eliminated
- Legacy compatibility burden: Removed

Risk Mitigation

Breaking Changes:

- **Risk**: Users dependent on CLI arguments
- **Mitigation**: Comprehensive templates + migration guide

Migration Effort:

- **Risk**: Complex migration for existing users
- **Mitigation**: Template generator + automated migration tools

Feature Parity:

- **Risk**: Missing functionality in YAML format
- **Mitigation**: Complete feature mapping + extensive testing

This refactoring transforms the security scanner into a modern, configuration-driven tool that's more suitable for DevSecOps workflows while maintaining all existing functionality and improving the user experience significantly.