# Scanner Integration System

Comprehensive integration of the custom-built scanner module with the main API, providing security scanning capabilities including vulnerability assessment, compliance checking, and network discovery.

## 🎯 Overview

The Scanner Integration System provides: - **Multiple Scan Types** - Internal network, vulnerability, compliance, and web application scanning - **Job Management** - Track scan execution, status, and results with full audit trail - **Result Processing** - Store, analyze, and report on scan findings - **Template System** - Predefined scan configurations for consistent scanning - **Target Management** - Managed scan targets with credentials and metadata - **Compliance Policies** - Framework-based compliance checking (NIST, CIS, PCI, etc.) - **Reporting** - Generate executive and technical reports from scan results

## 🏗️ Architecture Components

### Scanner Module Integration

// Backend API Integration with RBAC  
const BackendAPIIntegration = require('../../scanner/lib/integration/backend-api-integration');  
  
const scannerIntegration = new BackendAPIIntegration({  
 rbac: {  
 roles: {  
 'admin': {   
 permissions: ['internal-scan', 'vuln-scan', 'compliance-scan', 'web-scan', 'view-results', 'delete-results', 'schedule-scans']   
 },  
 'security-analyst': {  
 permissions: ['vuln-scan', 'compliance-scan', 'web-scan', 'view-results', 'schedule-scans']  
 },  
 'auditor': {  
 permissions: ['view-results']  
 },  
 'user': {  
 permissions: ['view-results']  
 }  
 }  
 },  
 resultsDir: process.env.SCANNER\_RESULTS\_DIR || './scan-results'  
});

### Database Schema

-- Core scan execution tracking  
scan\_jobs (id, scan\_type, target, configuration, status, initiated\_by,   
 error\_message, created\_at, completed\_at, updated\_at)  
  
-- Scan results and findings  
scan\_results (id, scan\_job\_id, scan\_type, target, results, summary,   
 file\_path, created\_at)  
  
-- Individual findings from scans  
scan\_findings (id, scan\_result\_id, finding\_type, severity, title, description,   
 recommendation, cve\_id, cvss\_score, port, service, evidence,   
 status, assigned\_to, resolved\_at, resolved\_by, created\_at, updated\_at)  
  
-- Scheduled and recurring scans  
scan\_schedules (id, name, description, scan\_type, target, configuration,   
 schedule, enabled, created\_by, last\_run, next\_run, created\_at, updated\_at)  
  
-- Predefined scan configurations  
scan\_templates (id, name, description, scan\_type, configuration, is\_default,   
 created\_by, created\_at, updated\_at)  
  
-- Managed scan targets  
scan\_targets (id, name, description, target, target\_type, credentials, tags,   
 metadata, enabled, created\_by, created\_at, updated\_at)  
  
-- Compliance and security policies  
scan\_policies (id, name, description, policy\_type, framework, rules, enabled,   
 created\_by, created\_at, updated\_at)  
  
-- Generated reports  
scan\_reports (id, name, description, report\_type, scan\_job\_ids, format,   
 file\_path, generated\_by, generated\_at, expires\_at, download\_count, created\_at)

### Service Architecture

const scannerService = {  
 // Scan Execution  
 executeInternalScan(scanConfig, userId),  
 executeVulnerabilityScan(target, scanConfig, userId),  
 executeComplianceScan(target, scanConfig, userId),  
  
 // Results Management  
 getAllScanJobs(filters, pagination),  
 getScanJobById(jobId),  
 getScanStatistics(),  
  
 // Utility Methods  
 sendScanNotification(eventType, data, userId)  
};

## 📋 Scan Types and Capabilities

### Internal Network Scanning

const internalScanConfig = {  
 networkRange: 'auto', // Auto-detect or specify range  
 scanType: 'quick', // quick, comprehensive, stealth  
 ports: [22, 80, 443, 3389], // Specific ports to scan  
 excludeHosts: ['192.168.1.1'], // Hosts to exclude  
 timeout: 300, // Scan timeout in seconds  
 maxConcurrency: 10, // Max concurrent connections  
 enableServiceDetection: true, // Enable service detection  
 enableOSDetection: false, // Enable OS fingerprinting  
 customOptions: {} // Additional scanner options  
};  
  
// Execute internal scan  
POST /api/v1/scanner/internal-scan

### Vulnerability Scanning

const vulnerabilityScanConfig = {  
 target: '192.168.1.100', // Target host or IP  
 scanType: 'basic', // basic, full, web, database  
 ports: [80, 443, 8080], // Specific ports to scan  
 credentials: { // Optional authentication  
 username: 'admin',  
 password: 'password',  
 keyFile: '/path/to/key'  
 },  
 excludeCVEs: ['CVE-2021-1234'], // CVEs to exclude  
 severity: 'medium', // Minimum severity: low, medium, high, critical  
 timeout: 1800, // Scan timeout in seconds  
 customOptions: {} // Additional scanner options  
};  
  
// Execute vulnerability scan  
POST /api/v1/scanner/vulnerability-scan

### Compliance Scanning

const complianceScanConfig = {  
 target: '192.168.1.100', // Target host or IP  
 frameworks: ['nist', 'cis'], // Compliance frameworks to check  
 scanType: 'configuration', // configuration, policy, full  
 credentials: { // Optional authentication  
 username: 'admin',  
 password: 'password'  
 },  
 customPolicies: ['/path/to/policy.xml'], // Custom policy files  
 timeout: 1800, // Scan timeout in seconds  
 customOptions: {} // Additional scanner options  
};  
  
// Execute compliance scan  
POST /api/v1/scanner/compliance-scan

## 🔍 Scan Results and Findings

### Scan Job Lifecycle

const scanJobStates = {  
 pending: 'Scan queued for execution',  
 running: 'Scan currently executing',  
 completed: 'Scan finished successfully',  
 failed: 'Scan encountered an error',  
 cancelled: 'Scan was cancelled by user'  
};  
  
// Track scan progress  
GET /api/v1/scanner/jobs/{jobId}/status  
  
// Get detailed results  
GET /api/v1/scanner/jobs/{jobId}

### Finding Management

const findingTypes = {  
 vulnerability: 'Security vulnerability found',  
 compliance: 'Compliance violation detected',  
 configuration: 'Configuration issue identified'  
};  
  
const severityLevels = {  
 low: 'Low impact finding',  
 medium: 'Medium impact finding',  
 high: 'High impact finding',  
 critical: 'Critical security issue'  
};  
  
const findingStatuses = {  
 open: 'Finding needs attention',  
 resolved: 'Finding has been fixed',  
 false\_positive: 'Finding is not valid',  
 accepted\_risk: 'Risk accepted by management'  
};

### Result Processing

// Example scan result structure  
const scanResult = {  
 scanId: 'scan\_20240115\_103045',  
 timestamp: '2024-01-15T10:30:45Z',  
 summary: {  
 total: 15,  
 critical: 2,  
 high: 5,  
 medium: 6,  
 low: 2  
 },  
 findings: [  
 {  
 type: 'vulnerability',  
 severity: 'critical',  
 title: 'Remote Code Execution in Web Server',  
 description: 'Buffer overflow vulnerability allows remote code execution',  
 cveId: 'CVE-2024-1234',  
 cvssScore: '9.8',  
 port: 80,  
 service: 'http',  
 recommendation: 'Update web server to latest version',  
 evidence: {  
 request: 'GET /vulnerable-endpoint HTTP/1.1',  
 response: 'HTTP/1.1 500 Internal Server Error'  
 }  
 }  
 ],  
 outputFile: '/scan-results/scan\_20240115\_103045.json'  
};

## 📊 Analytics and Reporting

### Scan Statistics

// Get comprehensive scan statistics  
GET /api/v1/scanner/statistics  
  
const statistics = {  
 totalScans: 1247,  
 statusBreakdown: [  
 { status: 'completed', count: 892 },  
 { status: 'running', count: 12 },  
 { status: 'failed', count: 121 }  
 ],  
 typeBreakdown: [  
 { scanType: 'vulnerability', count: 567 },  
 { scanType: 'internal', count: 234 },  
 { scanType: 'compliance', count: 123 }  
 ],  
 recentScans: 45 // Last 30 days  
};

### Report Generation

const reportTypes = {  
 executive: 'High-level summary for management',  
 technical: 'Detailed technical findings',  
 compliance: 'Compliance status and gaps'  
};  
  
const reportFormats = {  
 pdf: 'PDF document',  
 html: 'HTML web page',  
 json: 'JSON data format'  
};

## 🔐 Security and Access Control

### Role-Based Permissions

const scannerPermissions = {  
 'internal-scan': 'Execute internal network scans',  
 'vuln-scan': 'Execute vulnerability scans',  
 'compliance-scan': 'Execute compliance scans',  
 'web-scan': 'Execute web application scans',  
 'view-results': 'View scan results and findings',  
 'delete-results': 'Delete scan jobs and results',  
 'schedule-scans': 'Create and manage scheduled scans',  
 'admin': 'Full scanner administration'  
};  
  
// Permission checking in routes  
router.post('/internal-scan',   
 requirePermission('scanner', 'internal-scan'),  
 scannerController.executeInternalScan  
);

### Audit Logging

// All scanner actions are logged  
const auditEvents = {  
 'internal\_scan\_completed': 'Internal scan finished',  
 'vulnerability\_scan\_completed': 'Vulnerability scan finished',  
 'compliance\_scan\_completed': 'Compliance scan finished',  
 'scan\_job\_deleted': 'Scan job deleted',  
 'result\_accessed': 'Scan result viewed'  
};

## 🚀 API Endpoints

### Scan Execution

POST /api/v1/scanner/internal-scan # Execute internal network scan  
POST /api/v1/scanner/vulnerability-scan # Execute vulnerability scan  
POST /api/v1/scanner/compliance-scan # Execute compliance scan

### Results Management

GET /api/v1/scanner/jobs # List all scan jobs with filtering  
GET /api/v1/scanner/jobs/{jobId} # Get scan job details with results  
GET /api/v1/scanner/jobs/{jobId}/status # Get scan status  
POST /api/v1/scanner/jobs/{jobId}/cancel # Cancel running scan  
GET /api/v1/scanner/statistics # Get scan statistics

## 💡 Usage Examples

### Execute Vulnerability Scan

const executeVulnScan = async (target) => {  
 const scanConfig = {  
 target: target,  
 scanType: 'basic',  
 severity: 'medium',  
 timeout: 1800  
 };  
  
 const response = await fetch('/api/v1/scanner/vulnerability-scan', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify(scanConfig)  
 });  
  
 const result = await response.json();  
 return result.data; // { scanJobId, resultId, scanId, status, summary }  
};

### Monitor Scan Progress

const monitorScan = async (jobId) => {  
 const checkStatus = async () => {  
 const response = await fetch(`/api/v1/scanner/jobs/${jobId}/status`, {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
   
 const { data } = await response.json();  
   
 if (data.status === 'completed') {  
 console.log('Scan completed successfully');  
 return await getScanResults(jobId);  
 } else if (data.status === 'failed') {  
 console.error('Scan failed:', data.errorMessage);  
 return null;  
 } else {  
 console.log(`Scan status: ${data.status}`);  
 setTimeout(checkStatus, 5000); // Check again in 5 seconds  
 }  
 };  
  
 return checkStatus();  
};

### Process Scan Results

const processScanResults = async (jobId) => {  
 const response = await fetch(`/api/v1/scanner/jobs/${jobId}`, {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
  
 const { data } = await response.json();  
   
 // Process findings by severity  
 const findings = data.results[0]?.results?.findings || [];  
 const criticalFindings = findings.filter(f => f.severity === 'critical');  
 const highFindings = findings.filter(f => f.severity === 'high');  
   
 console.log(`Found ${criticalFindings.length} critical and ${highFindings.length} high severity issues`);  
   
 // Create tickets for critical findings  
 for (const finding of criticalFindings) {  
 await createSecurityTicket(finding);  
 }  
   
 return data;  
};

### Administrative Operations

// Get scan statistics for dashboard  
const getScanStats = async () => {  
 const response = await fetch('/api/v1/scanner/statistics', {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
   
 return await response.json();  
};  
  
// Cancel running scan  
const cancelScan = async (jobId) => {  
 const response = await fetch(`/api/v1/scanner/jobs/${jobId}/cancel`, {  
 method: 'POST',  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
   
 return await response.json();  
};  
  
// List recent scans  
const getRecentScans = async () => {  
 const params = new URLSearchParams({  
 page: 1,  
 limit: 10,  
 sortBy: 'createdAt',  
 sortOrder: 'desc'  
 });  
  
 const response = await fetch(`/api/v1/scanner/jobs?${params}`, {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
  
 return await response.json();  
};

## 🎯 Key Benefits

### Integrated Security Scanning

* **Unified Interface** - Single API for all scan types and operations
* **Centralized Results** - All scan data stored and managed in main database
* **Role-Based Access** - Granular permissions for different user types
* **Audit Trail** - Complete logging of all scanner activities

### Enterprise Features

* **Job Management** - Track scan execution with status monitoring
* **Template System** - Consistent scan configurations across teams
* **Compliance Integration** - Built-in support for major frameworks
* **Reporting** - Generate executive and technical reports

### Developer Experience

* **RESTful API** - Standard HTTP endpoints with comprehensive documentation
* **Real-time Status** - Monitor scan progress with status endpoints
* **Flexible Configuration** - Customizable scan parameters for different needs
* **Error Handling** - Proper error responses and status codes

The Scanner Integration System provides enterprise-grade security scanning capabilities with comprehensive management, reporting, and audit features suitable for government and enterprise environments.

# Settings Management System

Comprehensive application configuration management system with support for multiple data types, categories, and public/private settings.

## 🎯 Overview

The Settings Management System provides: - **Flexible Configuration** - Support for string, number, boolean, JSON, and array data types - **Categorized Settings** - Organize settings by functional categories - **Public/Private Settings** - Control which settings are accessible to frontend - **Bulk Operations** - Update multiple settings in a single operation - **Type Validation** - Automatic validation and conversion of setting values - **Audit Trail** - Complete logging of all setting changes

## 🏗️ Database Schema

-- Settings table with flexible data types  
CREATE TABLE settings (  
 id SERIAL PRIMARY KEY,  
 key VARCHAR(255) NOT NULL UNIQUE,  
 value TEXT,  
 data\_type setting\_data\_type DEFAULT 'string' NOT NULL,  
 category VARCHAR(255) DEFAULT 'general' NOT NULL,  
 description TEXT,  
 is\_public BOOLEAN DEFAULT false NOT NULL,  
 is\_editable BOOLEAN DEFAULT true NOT NULL,  
 created\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL,  
 updated\_at TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP NOT NULL  
);  
  
-- Session table for user session management  
CREATE TABLE session (  
 sid VARCHAR(255) PRIMARY KEY,  
 sess JSON NOT NULL,  
 expire TIMESTAMPTZ NOT NULL  
);  
  
-- Data type enum  
CREATE TYPE setting\_data\_type AS ENUM ('string', 'number', 'boolean', 'json', 'array');

## 📊 Setting Categories

### Application Settings

* app\_name - Application display name
* app\_version - Current version
* app\_description - Application description
* maintenance\_mode - Maintenance mode toggle

### Security Settings

* session\_timeout - Session timeout duration
* password\_min\_length - Minimum password length
* password\_require\_special - Require special characters
* max\_login\_attempts - Maximum failed login attempts
* lockout\_duration - Account lockout duration

### Email Settings

* smtp\_host - SMTP server hostname
* smtp\_port - SMTP server port
* smtp\_secure - Use secure connection
* email\_from\_address - Default from address
* email\_from\_name - Default from name

### Scanner Settings

* scanner\_max\_concurrent - Maximum concurrent scans
* scanner\_default\_timeout - Default scan timeout
* scanner\_results\_retention - Results retention period

### UI Settings

* ui\_theme - Default theme (light/dark)
* ui\_items\_per\_page - Default pagination size
* ui\_enable\_tooltips - Enable tooltips

## 🚀 API Endpoints

### Public Access

GET /api/v1/settings/public # Get public settings (no auth)

### Settings Management

GET /api/v1/settings # List all settings with filtering  
POST /api/v1/settings # Create new setting  
GET /api/v1/settings/{id} # Get setting by ID  
PUT /api/v1/settings/{id} # Update setting by ID  
DELETE /api/v1/settings/{id} # Delete setting by ID  
GET /api/v1/settings/key/{key} # Get setting by key  
PUT /api/v1/settings/key/{key} # Update setting by key

### Utility Endpoints

GET /api/v1/settings/categories # Get all categories  
PUT /api/v1/settings/bulk-update # Bulk update settings

## 💡 Usage Examples

### Get Public Settings (Frontend)

const getPublicSettings = async () => {  
 const response = await fetch('/api/v1/settings/public');  
 const { data } = await response.json();  
  
 // Settings are grouped by category  
 console.log(data.application.app\_name.value); // "RAS Dashboard"  
 console.log(data.ui.ui\_theme.value); // "light"  
  
 return data;  
};

### Create Setting

const createSetting = async (settingData) => {  
 const response = await fetch('/api/v1/settings', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 key: 'custom\_feature\_enabled',  
 value: true,  
 dataType: 'boolean',  
 category: 'features',  
 description: 'Enable custom feature functionality',  
 isPublic: false,  
 isEditable: true  
 })  
 });  
  
 return await response.json();  
};

### Update Setting by Key

const updateSetting = async (key, value) => {  
 const response = await fetch(`/api/v1/settings/key/${key}`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({ value })  
 });  
  
 return await response.json();  
};  
  
// Usage  
await updateSetting('app\_name', 'My Custom Dashboard');  
await updateSetting('max\_login\_attempts', 3);  
await updateSetting('ui\_theme', 'dark');

### Bulk Update Settings

const bulkUpdateSettings = async (settings) => {  
 const response = await fetch('/api/v1/settings/bulk-update', {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 'Authorization': `Bearer ${token}`  
 },  
 body: JSON.stringify({  
 app\_name: 'Updated Dashboard',  
 session\_timeout: 7200,  
 ui\_theme: 'dark',  
 scanner\_max\_concurrent: 10  
 })  
 });  
  
 const { data } = await response.json();  
 console.log(`Updated ${data.summary.successful} settings successfully`);  
  
 return data;  
};

### Get Settings with Filtering

const getSettings = async (filters = {}) => {  
 const params = new URLSearchParams({  
 category: filters.category || '',  
 search: filters.search || '',  
 page: filters.page || 1,  
 limit: filters.limit || 50,  
 sortBy: filters.sortBy || 'category',  
 sortOrder: filters.sortOrder || 'asc'  
 });  
  
 const response = await fetch(`/api/v1/settings?${params}`, {  
 headers: { 'Authorization': `Bearer ${token}` }  
 });  
  
 return await response.json();  
};  
  
// Usage examples  
const securitySettings = await getSettings({ category: 'security' });  
const searchResults = await getSettings({ search: 'password' });  
const paginatedResults = await getSettings({ page: 2, limit: 10 });

## 🔧 Data Type Handling

### String Settings

{  
 key: 'app\_name',  
 value: 'RAS Dashboard',  
 dataType: 'string'  
}

### Number Settings

{  
 key: 'session\_timeout',  
 value: 3600,  
 dataType: 'number'  
}

### Boolean Settings

{  
 key: 'maintenance\_mode',  
 value: false,  
 dataType: 'boolean'  
}

### JSON Settings

{  
 key: 'ui\_config',  
 value: {  
 theme: 'light',  
 sidebar: { collapsed: false },  
 notifications: { enabled: true }  
 },  
 dataType: 'json'  
}

### Array Settings

{  
 key: 'allowed\_domains',  
 value: ['example.com', 'subdomain.example.com'],  
 dataType: 'array'  
}

## 🔐 Security Features

### Access Control

* **Role-based permissions** for settings management
* **Public/private setting** distinction
* **Editable/read-only** setting protection

### Audit Logging

* All setting changes are logged with user context
* Before/after values tracked for changes
* Complete audit trail for compliance

### Data Validation

* Type validation for all data types
* Value conversion and sanitization
* Error handling for invalid data

## 🎯 Key Benefits

### Flexible Configuration

* **Multiple Data Types** - Support for all common data types
* **Categorized Organization** - Logical grouping of related settings
* **Public API** - Frontend access to public settings without authentication

### Enterprise Features

* **Bulk Operations** - Efficient mass updates
* **Audit Trail** - Complete change tracking
* **Type Safety** - Automatic validation and conversion

### Developer Experience

* **RESTful API** - Standard HTTP endpoints
* **Comprehensive Documentation** - Full Swagger documentation
* **Type Conversion** - Automatic handling of data type conversions

The Settings Management System provides a robust, flexible foundation for application configuration with enterprise-grade security and audit capabilities.