RHEL STIG RAG System - Installation Guide

A comprehensive guide for installing and configuring the RHEL STIG RAG (Retrieval-Augmented Generation) system, with primary focus on RHEL 9 and secondary support for RHEL 8.

Table of Contents

- Overview
- Prerequisites
- Quick Installation
- Manual Installation
- Docker Installation
- Configuration
- Loading STIG Data
- Usage Examples
- Troubleshooting
- Advanced Configuration
- Maintenance
- Uninstallation

Overview

The RHEL STIG RAG system is an AI-powered assistant that helps with Red Hat Enterprise Linux Security Technical Implementation Guide (STIG) compliance. It provides:

- **Intelligent STIG Guidance**: AI-powered answers to security compliance questions
- **Version-Aware Support**: Primary focus on RHEL 9, secondary support for RHEL 8
- **Multiple Interfaces**: REST API, command-line client, and interactive mode
- **Document Processing**: Supports XML (XCCDF) and JSON STIG formats
- **Semantic Search**: Vector-based search for relevant STIG controls

Prerequisites

System Requirements

- **Operating System**: Linux, macOS, or Windows (Linux recommended)
- **Python**: Version 3.8 or higher

- **Memory**: Minimum 4GB RAM (8GB+ recommended)
- **Storage**: At least 2GB free space
- **Network**: Internet access for downloading dependencies and STIG documents

Software Dependencies

- Python 3.8+
- pip (Python package installer)
- curl (for API testing)
- unzip (for extracting STIG archives)

Installing Prerequisites on Different Systems

RHEL/CentOS/Fedora:

```bash
sudo dnf install -y python3 python3-pip curl unzip git
```

Ubuntu/Debian:

```bash
sudo apt update
sudo apt install -y python3 python3-pip curl unzip git
```

macOS:

```bash
# Install Homebrew if not already installed
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install dependencies
brew install python3 curl git
```

Windows:

- Install Python 3.8+ from python.org
- Install Git from git-scm.com
- Use PowerShell or WSL for installation

Quick Installation

The fastest way to get started is using the automated setup script:

```bash
# Clone the repository
git clone <repository-url>
cd rhel-stig-rag

# Make setup script executable and run
chmod +x setup.sh
./setup.sh
```

The setup script will:

- Create a Python virtual environment
- Install all dependencies
- Create necessary directories
- Generate sample STIG data
- Start the service
- Run basic tests

Quick Installation Options

```bash
# Development mode (with auto-reload)
./setup.sh --dev

# Skip testing (faster setup)
./setup.sh --no-test

# Show help
./setup.sh --help
```

Manual Installation

If you prefer manual control over the installation process:

Step 1: Clone and Setup

```bash
# Clone the repository
git clone <repository-url>
cd rhel-stig-rag

# Create and activate virtual environment
python3 -m venv stig_rag_env
source stig_rag_env/bin/activate

# Upgrade pip and install requirements
pip install --upgrade pip setuptools wheel
pip install -r requirements.txt
```

Step 2: Create Directory Structure

```bash
# Create necessary directories
mkdir -p stig_data
mkdir -p stig_chroma_db
mkdir -p logs
```

Step 3: Generate Sample Data

```bash
# Create sample STIG data for testing
python3 stig_data_collector.py
# Choose option 4 to create sample data
```

Step 4: Start the Service

```bash
# Start the STIG RAG service
python3 rhel_stig_rag.py
```

The service will start on `http://localhost:8000` by default.

Step 5: Load Initial Data

In a new terminal:

```bash
# Activate virtual environment
source stig_rag_env/bin/activate

# Load sample RHEL 9 data (primary)
python3 stig_client.py load stig_data/sample_rhel9_stig.json

# Load sample RHEL 8 data (secondary)
python3 stig_client.py load stig_data/sample_rhel8_stig.json
```

Step 6: Test Installation

```bash
# Test with a basic query
python3 stig_client.py query "How do I verify GPG signatures?"

# Test health check
curl http://localhost:8000/health
```

Docker Installation

For containerized deployment:

Using Docker Compose (Recommended)

```bash
# Clone repository
git clone <repository-url>
cd rhel-stig-rag

# Build and start services
docker-compose up --build

# In another terminal, load sample data
docker-compose exec stig-rag python3 stig_client.py load stig_data/sample_rhel9_stig.json
```

Using Docker Only

```bash
# Build the image
docker build -t rhel-stig-rag .

# Run the container
docker run -d \
  -p 8000:8000 \
  -v $(pwd)/stig_data:/app/stig_data \
  -v $(pwd)/stig_chroma_db:/app/stig_chroma_db \
  -v $(pwd)/logs:/app/logs \
  --name stig-rag \
  rhel-stig-rag

# Load sample data
docker exec stig-rag python3 stig_client.py load /app/stig_data/sample_rhel9_stig.json
```

Configuration

Environment Variables

Copy the configuration template:

```bash
cp .env.example .env
```

Key configuration options in `.env`:

```bash
# Application Settings
APP_HOST=0.0.0.0
APP_PORT=8000

# RHEL Version Priority
DEFAULT_RHEL_VERSION=9
SUPPORTED_RHEL_VERSIONS=8,9

# Vector Store Settings
VECTORSTORE_PATH=./stig_chroma_db
EMBEDDING_MODEL=all-MiniLM-L6-v2

# Language Model Settings
LLM_PROVIDER=huggingface
LLM_MODEL=microsoft/DialoGPT-medium

# Search Settings
DEFAULT_SEARCH_RESULTS=5
MAX_SEARCH_RESULTS=20
PREFER_RHEL9_RESULTS=true
```

Advanced Configuration

Using OpenAI Models

```bash
# In .env file
LLM_PROVIDER=openai
OPENAI_API_KEY=your_openai_api_key_here
OPENAI_MODEL=gpt-3.5-turbo
```

Using Anthropic Claude

```bash
# In .env file
LLM_PROVIDER=anthropic
ANTHROPIC_API_KEY=your_anthropic_api_key_here
ANTHROPIC_MODEL=claude-3-sonnet-20240229
```

Performance Tuning

```bash
# Increase chunk size for larger documents
CHUNK_SIZE=1500
CHUNK_OVERLAP=300

# Adjust search results
DEFAULT_SEARCH_RESULTS=10
MAX_SEARCH_RESULTS=50

# Enable caching
ENABLE_CACHING=true
CACHE_TTL_SECONDS=7200
```

Loading STIG Data

Downloading Official STIG Documents

```bash
# Use the data collector to download official STIGs
python3 stig_data_collector.py

# Options:
# 1. Download RHEL 9 STIG (Primary)
# 2. Download RHEL 8 STIG (Secondary)
# 3. Download both RHEL 9 and 8 STIGs
```

Loading STIG Documents

XML Format (XCCDF)

```bash
# Load RHEL 9 STIG
python3 stig_client.py load /path/to/rhel9-stig.xml

# Load RHEL 8 STIG
python3 stig_client.py load /path/to/rhel8-stig.xml
```

JSON Format

```bash
# Load custom JSON STIG
python3 stig_client.py load /path/to/custom-stig.json
```

Batch Loading

```bash
# Load all STIG files in a directory
for file in stig_data/*.xml; do
    python3 stig_client.py load "$file"
done
```

Usage Examples

Command Line Interface

```bash
# Basic queries (defaults to RHEL 9)
python3 stig_client.py query "How do I configure secure boot?"

# Version-specific queries
python3 stig_client.py query "How do I enable kernel lockdown?" --rhel-version 9
python3 stig_client.py query "How do I enable kernel lockdown?" --rhel-version 8

# Search by STIG ID
python3 stig_client.py search RHEL-09-211010
python3 stig_client.py search RHEL-08-010010

# Health check
python3 stig_client.py health
```

Interactive Mode

bash

```bash
# Start interactive session
python3 stig_client.py interactive

# Commands in interactive mode:
STIG> query How do I secure package management?     # RHEL 9 default
STIG> query9 How do I secure package management?    # RHEL 9 specific
STIG> query8 How do I secure package management?    # RHEL 8 specific
STIG> search RHEL-09-211015
STIG> health
STIG> help
STIG> exit
```

REST API

bash

```bash
# Query endpoint
curl -X POST "http://localhost:8000/query" \
  -H "Content-Type: application/json" \
  -d '{
    "question": "How do I enable GPG checking?",
    "rhel_version": "9"
  }'

# Search by STIG ID
curl "http://localhost:8000/search/RHEL-09-211010"

# Load STIG document
curl -X POST "http://localhost:8000/load-stig?file_path=/path/to/stig.xml"

# Health check
curl "http://localhost:8000/health"
```

API Documentation

Access the interactive API documentation at: `http://localhost:8000/docs`

Troubleshooting

Common Issues

1. Service Won't Start

Problem: `Address already in use` error

Solution:

```bash
# Check if port 8000 is in use
lsof -i :8000

# Kill existing process or change port
export APP_PORT=8001
python3 rhel_stig_rag.py
```

2. Import Errors

Problem: `ModuleNotFoundError` when starting

Solution:

```bash
# Ensure virtual environment is activated
source stig_rag_env/bin/activate

# Reinstall requirements
pip install -r requirements.txt
```

3. Memory Issues

Problem: System runs out of memory

Solution:

```bash
# Reduce chunk size in .env
CHUNK_SIZE=500
CHUNK_OVERLAP=100

# Limit search results
DEFAULT_SEARCH_RESULTS=3
MAX_SEARCH_RESULTS=10
```

4. STIG Loading Fails

Problem: Error loading STIG documents

Solution:

```bash
# Check file permissions
chmod 644 stig_data/*.xml

# Verify file format
file stig_data/your-stig.xml

# Check logs
tail -f logs/stig_rag.log
```

Performance Issues

Slow Queries

1. **Reduce embedding model size**:

   ```bash
   EMBEDDING_MODEL=all-MiniLM-L6-v2  # Smaller, faster model
   ```

2. **Enable caching**:

   ```bash
   ENABLE_CACHING=true
   CACHE_TTL_SECONDS=3600
   ```

3. **Optimize chunk size**:

   ```bash
   CHUNK_SIZE=800
   CHUNK_OVERLAP=150
   ```

High Memory Usage

1. **Use smaller models**:

   ```bash
   LLM_MODEL=microsoft/DialoGPT-small
   ```

2. **Limit vector store size**:

```bash
# Periodically clean old embeddings
rm -rf stig_chroma_db/*
# Reload only essential STIGs
```

Debugging

Enable Debug Logging

```bash
# In .env
LOG_LEVEL=DEBUG

# View logs
tail -f logs/stig_rag.log
```

Test Components

```bash
# Test vector store
python3 -c "
from rhel_stig_rag import STIGVectorStore
vs = STIGVectorStore()
print('Vector store loaded successfully')
"

# Test LLM
python3 -c "
from rhel_stig_rag import STIGRAGSystem, STIGVectorStore
vs = STIGVectorStore()
rag = STIGRAGSystem(vs)
print('RAG system initialized successfully')
"
```

Advanced Configuration

Custom Embedding Models

python

```python
# In rhel_stig_rag.py, modify STIGVectorStore.__init__
self.embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-mpnet-base-v2",  # Higher quality
    model_kwargs={'device': 'cuda'}  # Use GPU if available
)
```

Fine-tuning for Domain-Specific Content

bash

```bash
# Prepare training data
python3 prepare_training_data.py

# Fine-tune model (requires additional setup)
python3 fine_tune_model.py
```

Integration with Configuration Management

Ansible Integration

yaml

```yaml
# ansible-playbook.yml
- name: Query STIG Requirements
  uri:
    url: "http://localhost:8000/query"
    method: POST
    body_format: json
    body:
      question: "How do I configure {{ item }}?"
      rhel_version: "{{ ansible_distribution_major_version }}"
  loop: "{{ security_controls }}"
```

Puppet Integration

puppet

```
# puppet-manifest.pp
$stig_response = http_get("http://localhost:8000/query", {
  'question' => 'How do I secure SSH configuration?',
  'rhel_version' => $facts['os']['release']['major']
})
```

Scaling for Production

Load Balancing

bash

```
# Using nginx for load balancing
upstream stig_backend {
    server localhost:8000;
    server localhost:8001;
    server localhost:8002;
}
```

High Availability

yaml

```
# docker-swarm.yml
version: '3.8'
services:
  stig-rag:
    image: rhel-stig-rag
    replicas: 3
    networks:
      - stig-network
```

Maintenance

Regular Updates

Update Dependencies

```bash
# Activate virtual environment
source stig_rag_env/bin/activate

# Update packages
pip list --outdated
pip install --upgrade package_name

# Update all packages
pip install --upgrade -r requirements.txt
```

Update STIG Documents

```bash
# Download latest STIGs
python3 stig_data_collector.py

# Reload updated documents
python3 stig_client.py load stig_data/latest-rhel9-stig.xml
```

Database Maintenance

Clean Vector Store

```bash
# Backup current database
cp -r stig_chroma_db stig_chroma_db.backup

# Clean and rebuild
rm -rf stig_chroma_db/*
python3 rebuild_vector_store.py
```

Optimize Performance

```bash
# Check database size
du -sh stig_chroma_db/

# Compact database (if supported)
python3 compact_database.py
```

Monitoring

System Health

```bash
# Monitor service status
curl -s http://localhost:8000/health | jq

# Check resource usage
top -p $(pgrep -f rhel_stig_rag.py)

# Monitor logs
tail -f logs/stig_rag.log | grep ERROR
```

Performance Metrics

```bash
# Query response times
curl -w "@curl-format.txt" -X POST http://localhost:8000/query \
  -H "Content-Type: application/json" \
  -d '{"question": "test"}'

# Memory usage
ps -o pid,vsz,rss,comm -p $(pgrep -f rhel_stig_rag.py)
```

Backup and Recovery

Backup Strategy

bash

```bash
#!/bin/bash
# backup.sh
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="backups/backup_$DATE"

mkdir -p "$BACKUP_DIR"
cp -r stig_chroma_db "$BACKUP_DIR/"
cp -r stig_data "$BACKUP_DIR/"
cp .env "$BACKUP_DIR/"
tar -czf "$BACKUP_DIR.tar.gz" "$BACKUP_DIR"
rm -rf "$BACKUP_DIR"
```

Recovery Process

bash

```bash
# Extract backup
tar -xzf backup_20241201_120000.tar.gz

# Restore files
cp -r backup_20241201_120000/stig_chroma_db ./
cp -r backup_20241201_120000/stig_data ./
cp backup_20241201_120000/.env ./

# Restart service
./setup.sh --no-test
```

Uninstallation

Stopping Services

bash

```bash
# Stop the service
kill $(cat .app_pid) && rm .app_pid

# Or if using Docker
docker-compose down
```

Cleaning Up

```bash
# Remove virtual environment
rm -rf stig_rag_env/

# Remove data directories
rm -rf stig_chroma_db/
rm -rf stig_data/
rm -rf logs/

# Remove application files
rm -rf rhel-stig-rag/
```

Docker Cleanup

```bash
# Stop and remove containers
docker-compose down --volumes

# Remove images
docker rmi rhel-stig-rag

# Clean up unused resources
docker system prune -a
```

Support and Resources

Documentation

- **API Documentation**: `http://localhost:8000/docs`
- **Configuration Reference**: See `.env` file comments
- **STIG Resources**: DISA STIG Library

Getting Help

1. **Check the logs**: `tail -f logs/stig_rag.log`
2. **Run health checks**: `python3 stig_client.py health`
3. **Verify configuration**: Review `.env` file settings
4. **Test components**: Use the troubleshooting commands above

Contributing

To contribute to the project:

1. Fork the repository
2. Create a feature branch
3. Make your changes
4. Add tests
5. Submit a pull request

Note: This installation guide covers the basic setup and common configurations. For advanced use cases or custom deployments, you may need to modify the configuration based on your specific requirements.