# Tenable Local Testing Setup Guide

## Overview

This guide provides instructions for testing Tenable API integration locally without requiring access to cloud.tenable.com. We’ll use PyTenable for initial testing and create mock data structures that match the real API responses.

## Prerequisites

* Python 3.6 or later
* Node.js 16 or later (for the main RAS-DASH integration)
* Local PostgreSQL database
* Network isolation or restricted environment

## Option 1: PyTenable with Mock Server

### Step 1: Install PyTenable

# Install PyTenable and dependencies  
pip install pytenable  
pip install flask  
pip install requests-mock

### Step 2: Create Mock Tenable Server

# mock\_tenable\_server.py  
from flask import Flask, jsonify, request  
import json  
import uuid  
from datetime import datetime, timedelta  
import random  
  
app = Flask(\_\_name\_\_)  
  
# Mock data storage  
mock\_assets = []  
mock\_vulnerabilities = []  
mock\_scans = []  
  
# Generate realistic mock data  
def generate\_mock\_assets(count=50):  
 assets = []  
 for i in range(count):  
 asset = {  
 "id": str(uuid.uuid4()),  
 "hostname": [f"server-{i+1}.example.com"],  
 "ipv4": [f"192.168.1.{i+10}"],  
 "ipv6": [],  
 "fqdn": [f"server-{i+1}.example.com"],  
 "netbios\_name": f"SERVER{i+1}",  
 "operating\_system": [random.choice([  
 "Windows Server 2019",  
 "Ubuntu 20.04 LTS",  
 "CentOS 8",  
 "Red Hat Enterprise Linux 8"  
 ])],  
 "system\_type": [random.choice(["general-purpose", "database", "web-server"])],  
 "has\_agent": random.choice([True, False]),  
 "agent\_uuid": [str(uuid.uuid4())] if random.choice([True, False]) else [],  
 "exposure\_score": random.randint(0, 1000),  
 "acr\_score": random.randint(1, 10),  
 "criticality\_rating": random.choice(["low", "medium", "high", "critical"]),  
 "first\_seen": (datetime.now() - timedelta(days=random.randint(1, 365))).isoformat(),  
 "last\_seen": datetime.now().isoformat(),  
 "first\_scan\_time": (datetime.now() - timedelta(days=random.randint(1, 30))).isoformat(),  
 "last\_scan\_time": datetime.now().isoformat(),  
 "last\_authenticated\_scan\_date": datetime.now().isoformat(),  
   
 # AWS metadata (for some assets)  
 "aws\_ec2\_instance\_id": f"i-{random.randint(100000000000, 999999999999):012x}" if random.choice([True, False]) else None,  
 "aws\_ec2\_instance\_ami\_id": f"ami-{random.randint(100000000000, 999999999999):012x}" if random.choice([True, False]) else None,  
 "aws\_owner\_id": "123456789012" if random.choice([True, False]) else None,  
 "aws\_availability\_zone": random.choice(["us-east-1a", "us-east-1b", "us-west-2a"]) if random.choice([True, False]) else None,  
 "aws\_region": "us-east-1" if random.choice([True, False]) else None,  
   
 # Network information  
 "mac\_address": [f"00:1B:44:11:3A:{i+10:02X}"],  
 "network\_id": str(uuid.uuid4()),  
 "network\_name": "Corporate Network",  
   
 "indexed": datetime.now().isoformat()  
 }  
 assets.append(asset)  
 return assets  
  
def generate\_mock\_vulnerabilities(asset\_count=50, vuln\_per\_asset=5):  
 vulnerabilities = []  
 plugin\_families = ["Windows", "Ubuntu Local Security Checks", "Web Servers", "Databases", "Network Security"]  
 severities = ["critical", "high", "medium", "low", "info"]  
   
 for asset\_idx in range(asset\_count):  
 for vuln\_idx in range(random.randint(1, vuln\_per\_asset)):  
 severity = random.choice(severities)  
 severity\_id = {"critical": 4, "high": 3, "medium": 2, "low": 1, "info": 0}[severity]  
   
 vuln = {  
 "asset": {  
 "id": str(uuid.uuid4()),  
 "hostname": f"server-{asset\_idx+1}.example.com",  
 "ipv4": f"192.168.1.{asset\_idx+10}"  
 },  
 "plugin": {  
 "id": 10000 + vuln\_idx,  
 "name": f"Sample Vulnerability {vuln\_idx+1}",  
 "family": random.choice(plugin\_families),  
 "modification\_date": (datetime.now() - timedelta(days=random.randint(1, 90))).date().isoformat(),  
 "publication\_date": (datetime.now() - timedelta(days=random.randint(90, 365))).date().isoformat(),  
 "risk\_factor": severity.title(),  
 "solution": "Apply the latest security patches and updates.",  
 "synopsis": f"A {severity} severity vulnerability was detected.",  
 "description": f"This is a sample {severity} vulnerability for testing purposes.",  
 "version": "1.0"  
 },  
 "scan": {  
 "id": random.randint(1000, 9999),  
 "uuid": str(uuid.uuid4()),  
 "started\_at": (datetime.now() - timedelta(hours=random.randint(1, 24))).isoformat(),  
 "completed\_at": datetime.now().isoformat()  
 },  
 "severity": severity,  
 "severity\_id": severity\_id,  
 "severity\_default\_id": severity\_id,  
 "state": random.choice(["open", "reopened", "fixed"]),  
 "first\_found": (datetime.now() - timedelta(days=random.randint(1, 30))).isoformat(),  
 "last\_found": datetime.now().isoformat(),  
 "cvss\_base\_score": round(random.uniform(0.0, 10.0), 1),  
 "cvss\_temporal\_score": round(random.uniform(0.0, 10.0), 1),  
 "cvss\_vector": "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",  
 "vpr\_score": round(random.uniform(0.0, 10.0), 1),  
 "output": f"Sample vulnerability output for testing - {severity} level finding",  
 "port": random.choice([80, 443, 22, 3389, 21, 25, 53]),  
 "protocol": random.choice(["tcp", "udp"]),  
 "indexed": datetime.now().isoformat()  
 }  
 vulnerabilities.append(vuln)  
 return vulnerabilities  
  
# Initialize mock data  
mock\_assets = generate\_mock\_assets(100)  
mock\_vulnerabilities = generate\_mock\_vulnerabilities(100, 10)  
  
# API Endpoints  
@app.route('/session', methods=['GET'])  
def get\_session():  
 return jsonify({  
 "id": 12345,  
 "username": "test@example.com",  
 "email": "test@example.com",  
 "name": "Test User",  
 "type": "local",  
 "permissions": 64,  
 "login\_fail\_count": 0,  
 "login\_fail\_total": 0,  
 "enabled": True,  
 "lastlogin": int(datetime.now().timestamp()),  
 "uuid": str(uuid.uuid4())  
 })  
  
@app.route('/scans', methods=['GET'])  
def get\_scans():  
 return jsonify({  
 "scans": [  
 {  
 "id": 1001,  
 "uuid": str(uuid.uuid4()),  
 "name": "Weekly Network Scan",  
 "status": "completed",  
 "starttime": int((datetime.now() - timedelta(hours=2)).timestamp()),  
 "endtime": int(datetime.now().timestamp()),  
 "folder\_id": 2,  
 "type": "public",  
 "scanner\_name": "Local Scanner"  
 },  
 {  
 "id": 1002,  
 "uuid": str(uuid.uuid4()),  
 "name": "Vulnerability Assessment",  
 "status": "running",  
 "starttime": int((datetime.now() - timedelta(minutes=30)).timestamp()),  
 "endtime": None,  
 "folder\_id": 2,  
 "type": "public",  
 "scanner\_name": "Local Scanner"  
 }  
 ]  
 })  
  
@app.route('/assets', methods=['GET'])  
def get\_assets():  
 page = int(request.args.get('page', 1))  
 per\_page = int(request.args.get('per\_page', 50))  
   
 start\_idx = (page - 1) \* per\_page  
 end\_idx = start\_idx + per\_page  
   
 return jsonify({  
 "assets": mock\_assets[start\_idx:end\_idx],  
 "total": len(mock\_assets),  
 "page": page,  
 "per\_page": per\_page  
 })  
  
@app.route('/assets/export', methods=['POST'])  
def export\_assets():  
 export\_uuid = str(uuid.uuid4())  
 return jsonify({  
 "export\_uuid": export\_uuid  
 })  
  
@app.route('/assets/export/<export\_uuid>/status', methods=['GET'])  
def get\_asset\_export\_status(export\_uuid):  
 return jsonify({  
 "status": "FINISHED"  
 })  
  
@app.route('/assets/export/<export\_uuid>/chunks/<int:chunk\_id>', methods=['GET'])  
def get\_asset\_export\_chunk(export\_uuid, chunk\_id):  
 if chunk\_id == 1:  
 return jsonify(mock\_assets[:50])  
 elif chunk\_id == 2:  
 return jsonify(mock\_assets[50:])  
 else:  
 return jsonify([])  
  
@app.route('/workbenches/vulnerabilities', methods=['GET'])  
def get\_vulnerability\_workbench():  
 severity\_counts = {"critical": 0, "high": 0, "medium": 0, "low": 0, "info": 0}  
   
 for vuln in mock\_vulnerabilities:  
 severity\_counts[vuln["severity"]] += 1  
   
 vulnerabilities = []  
 for severity, count in severity\_counts.items():  
 if count > 0:  
 vulnerabilities.append({  
 "severity\_name": severity,  
 "count": count,  
 "plugin\_family": "Mixed"  
 })  
   
 return jsonify({  
 "vulnerabilities": vulnerabilities  
 })  
  
@app.route('/vulns/export', methods=['POST'])  
def export\_vulnerabilities():  
 export\_uuid = str(uuid.uuid4())  
 return jsonify({  
 "export\_uuid": export\_uuid  
 })  
  
@app.route('/vulns/export/<export\_uuid>/status', methods=['GET'])  
def get\_vuln\_export\_status(export\_uuid):  
 return jsonify({  
 "status": "FINISHED"  
 })  
  
@app.route('/vulns/export/<export\_uuid>/chunks/<int:chunk\_id>', methods=['GET'])  
def get\_vuln\_export\_chunk(export\_uuid, chunk\_id):  
 chunk\_size = 100  
 start\_idx = (chunk\_id - 1) \* chunk\_size  
 end\_idx = start\_idx + chunk\_size  
   
 if start\_idx < len(mock\_vulnerabilities):  
 return jsonify(mock\_vulnerabilities[start\_idx:end\_idx])  
 else:  
 return jsonify([])  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 print("Starting Mock Tenable Server...")  
 print("Access at: http://localhost:5001")  
 print("Available endpoints:")  
 print(" GET /session")  
 print(" GET /scans")  
 print(" GET /assets")  
 print(" POST /assets/export")  
 print(" GET /workbenches/vulnerabilities")  
 print(" POST /vulns/export")  
 app.run(host='0.0.0.0', port=5001, debug=True)

### Step 3: Create PyTenable Test Client

# test\_pytenable\_local.py  
from tenable.io import TenableIO  
import os  
from datetime import datetime  
  
# Configure to use local mock server  
class LocalTenableIO(TenableIO):  
 def \_\_init\_\_(self):  
 # Override the base URL to point to local server  
 super().\_\_init\_\_(  
 access\_key='mock\_access\_key',  
 secret\_key='mock\_secret\_key',  
 url='http://localhost:5001'  
 )  
  
def test\_local\_connection():  
 """Test connection to local mock server"""  
 try:  
 tio = LocalTenableIO()  
   
 # Test session endpoint  
 user\_info = tio.session.details()  
 print(f"✅ Connected successfully as: {user\_info['email']}")  
   
 # Test scans  
 scans = tio.scans.list()  
 print(f"📊 Available scans: {len(scans['scans'])}")  
   
 return True  
   
 except Exception as e:  
 print(f"❌ Connection failed: {str(e)}")  
 return False  
  
def test\_asset\_retrieval():  
 """Test asset data retrieval"""  
 try:  
 tio = LocalTenableIO()  
   
 # Get asset list  
 assets = tio.assets.list()  
 print(f"Found {len(assets)} assets")  
   
 # Show sample assets  
 for asset in assets[:3]:  
 hostname = asset.get('hostname', ['Unknown'])[0]  
 ip = asset.get('ipv4', ['N/A'])[0] if asset.get('ipv4') else 'N/A'  
 exposure = asset.get('exposure\_score', 0)  
 print(f" Asset: {hostname} ({ip}) - Exposure: {exposure}")  
   
 return assets  
   
 except Exception as e:  
 print(f"❌ Asset retrieval failed: {str(e)}")  
 return []  
  
def test\_vulnerability\_data():  
 """Test vulnerability data retrieval"""  
 try:  
 tio = LocalTenableIO()  
   
 # Get vulnerability workbench  
 vulns = tio.workbenches.vulnerabilities()  
   
 print("Vulnerability Summary:")  
 for vuln in vulns['vulnerabilities']:  
 severity = vuln['severity\_name']  
 count = vuln['count']  
 print(f" {severity.title()}: {count}")  
   
 return vulns  
   
 except Exception as e:  
 print(f"❌ Vulnerability retrieval failed: {str(e)}")  
 return {}  
  
def test\_export\_functionality():  
 """Test export functionality"""  
 try:  
 tio = LocalTenableIO()  
   
 # Test asset export  
 print("Testing asset export...")  
 export\_uuid = tio.exports.assets()  
 print(f"Asset export started: {export\_uuid}")  
   
 # Download chunks (mock server returns data immediately)  
 all\_assets = []  
 for chunk in tio.exports.download\_chunks(export\_uuid):  
 all\_assets.extend(chunk)  
 print(f"Downloaded chunk with {len(chunk)} assets")  
   
 print(f"Total assets exported: {len(all\_assets)}")  
   
 return True  
   
 except Exception as e:  
 print(f"❌ Export test failed: {str(e)}")  
 return False  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("=== PyTenable Local Testing ===")  
 print("Make sure mock\_tenable\_server.py is running on port 5001")  
 print()  
   
 # Run tests  
 if test\_local\_connection():  
 print("\n=== Testing Asset Retrieval ===")  
 assets = test\_asset\_retrieval()  
   
 print("\n=== Testing Vulnerability Data ===")  
 vulns = test\_vulnerability\_data()  
   
 print("\n=== Testing Export Functionality ===")  
 test\_export\_functionality()  
   
 print("\n✅ All tests completed successfully!")  
 else:  
 print("❌ Connection test failed - check if mock server is running")

## Option 2: Node.js with Mock Data

### Step 1: Create Node.js Test Environment

// test\_tenable\_nodejs.ts  
import axios, { AxiosInstance } from 'axios';  
  
class MockTenableClient {  
 private baseURL: string;  
 private client: AxiosInstance;  
   
 constructor() {  
 this.baseURL = 'http://localhost:5001';  
 this.client = axios.create({  
 baseURL: this.baseURL,  
 headers: {  
 'Content-Type': 'application/json',  
 'User-Agent': 'RAS-DASH-Test/1.0.0'  
 }  
 });  
 }  
   
 async testConnection() {  
 try {  
 const response = await this.client.get('/session');  
 return response.data;  
 } catch (error) {  
 throw new Error(`Connection failed: ${error.message}`);  
 }  
 }  
   
 async getAssets() {  
 try {  
 const response = await this.client.get('/assets');  
 return response.data;  
 } catch (error) {  
 throw new Error(`Failed to get assets: ${error.message}`);  
 }  
 }  
   
 async getVulnerabilities() {  
 try {  
 const response = await this.client.get('/workbenches/vulnerabilities');  
 return response.data;  
 } catch (error) {  
 throw new Error(`Failed to get vulnerabilities: ${error.message}`);  
 }  
 }  
   
 async exportAssets() {  
 try {  
 // Start export  
 const exportResponse = await this.client.post('/assets/export', {});  
 const exportUuid = exportResponse.data.export\_uuid;  
   
 console.log(`Asset export started: ${exportUuid}`);  
   
 // Get export status  
 const statusResponse = await this.client.get(`/assets/export/${exportUuid}/status`);  
 console.log(`Export status: ${statusResponse.data.status}`);  
   
 // Download chunks  
 const allAssets = [];  
 let chunkId = 1;  
   
 while (true) {  
 try {  
 const chunkResponse = await this.client.get(`/assets/export/${exportUuid}/chunks/${chunkId}`);  
 const chunk = chunkResponse.data;  
   
 if (!chunk || chunk.length === 0) {  
 break;  
 }  
   
 allAssets.push(...chunk);  
 console.log(`Downloaded chunk ${chunkId} with ${chunk.length} assets`);  
 chunkId++;  
   
 } catch (error) {  
 if (error.response?.status === 404) {  
 // No more chunks  
 break;  
 }  
 throw error;  
 }  
 }  
   
 return allAssets;  
   
 } catch (error) {  
 throw new Error(`Export failed: ${error.message}`);  
 }  
 }  
}  
  
async function runTests() {  
 try {  
 const client = new MockTenableClient();  
   
 console.log("=== Testing Connection ===");  
 const userInfo = await client.testConnection();  
 console.log(`✅ Connected as: ${userInfo.email}`);  
   
 console.log("\n=== Testing Asset Retrieval ===");  
 const assets = await client.getAssets();  
 console.log(`✅ Retrieved ${assets.assets.length} assets`);  
   
 console.log("\n=== Testing Vulnerability Data ===");  
 const vulns = await client.getVulnerabilities();  
 console.log(`✅ Retrieved vulnerability data for ${vulns.vulnerabilities.length} categories`);  
   
 console.log("\n=== Testing Asset Export ===");  
 const exportedAssets = await client.exportAssets();  
 console.log(`✅ Exported ${exportedAssets.length} total assets`);  
   
 console.log("\n🎉 All tests passed!");  
   
 } catch (error) {  
 console.error(`❌ Test failed: ${error.message}`);  
 }  
}  
  
// Run tests if this file is executed directly  
if (require.main === module) {  
 runTests();  
}  
  
export { MockTenableClient };

## Quick Start Instructions

### 1. Start Mock Server

# Terminal 1: Start the mock Tenable server  
python mock\_tenable\_server.py

### 2. Test with PyTenable

# Terminal 2: Test PyTenable integration  
python test\_pytenable\_local.py

### 3. Test with Node.js

# Terminal 3: Test Node.js integration  
npx tsx test\_tenable\_nodejs.ts

## Environment Setup for Local Testing

### Update .env for Local Testing

# Local Testing Configuration  
TENABLE\_ACCESS\_KEY=mock\_access\_key  
TENABLE\_SECRET\_KEY=mock\_secret\_key  
TENABLE\_BASE\_URL=http://localhost:5001  
TENABLE\_SYNC\_ENABLED=true  
  
# Use smaller intervals for testing  
TENABLE\_SYNC\_INTERVAL\_HOURS=1  
TENABLE\_ASSET\_CHUNK\_SIZE=50  
TENABLE\_VULN\_CHUNK\_SIZE=100

## Benefits of Local Testing

1. **No Network Dependencies**: Works in isolated environments
2. **Predictable Data**: Consistent test data for development
3. **Fast Iteration**: No API rate limits or delays
4. **Realistic Responses**: Mock data matches real Tenable API structure
5. **Full Coverage**: Test all integration scenarios including error cases

## Next Steps

1. **Start Mock Server**: Run python mock\_tenable\_server.py
2. **Test PyTenable**: Verify PyTenable works with mock server
3. **Test Node.js**: Confirm Node.js client works with mock data
4. **Integrate with RAS-DASH**: Use mock server for development
5. **Switch to Production**: Update base URL when Tenable access is available

This local testing environment provides a complete Tenable API simulation that matches the real API structure and responses, allowing full development and testing of the integration without requiring cloud access.