# Tenable API Integration Guide for RAS-DASH

## Complete Setup and Data Consumption Guide

### Table of Contents

1. [Getting Started](#getting-started)
2. [API Authentication Setup](#api-authentication-setup)
3. [Development Environment Setup](#development-environment-setup)
4. [Asset Data Consumption](#asset-data-consumption)
5. [Vulnerability Data Consumption](#vulnerability-data-consumption)
6. [Data Volume Management](#data-volume-management)
7. [Best Practices](#best-practices)
8. [Integration with RAS-DASH](#integration-with-ras-dash)
9. [Troubleshooting](#troubleshooting)
10. [Production Considerations](#production-considerations)

## Getting Started

### Overview

This guide provides complete instructions for integrating Tenable’s Vulnerability Management API into the RAS-DASH platform. Tenable’s API provides access to comprehensive asset inventory, vulnerability data, scan results, and security metrics that will enhance RAS-DASH’s cybersecurity intelligence capabilities.

### Prerequisites

* Node.js 16 or later
* Network access to cloud.tenable.com
* Basic understanding of REST APIs
* RAS-DASH development environment

### What You’ll Get Access To

* **Asset Inventory**: Complete asset discovery and inventory data
* **Vulnerability Data**: Detailed vulnerability information with CVSS scores
* **Scan Results**: Historical and real-time scan data
* **Compliance Data**: Regulatory compliance status and findings
* **Risk Metrics**: Risk scoring and prioritization data

## API Authentication Setup

### Step 1: Create Tenable Account

1. **Visit** [cloud.tenable.com](https://cloud.tenable.com)
2. **Sign up** for the 60-day free trial
3. **Complete** account verification
4. **Log in** to your new account

### Step 2: Generate API Keys

1. **Click** your user profile icon (upper-right corner)
2. **Navigate** to “My Account” → “API Keys”
3. **Click** “Generate” button
4. **Copy** both ACCESS KEY and SECRET KEY immediately
5. **Store** keys securely (you cannot retrieve them later)

**Important Notes:** - Each account can only have one API key pair at a time - Generating new keys invalidates existing keys - Treat these keys like passwords - store them securely

### Step 3: Test API Connection

# Test connection using curl  
curl -H "X-ApiKeys: accesskey=YOUR\_ACCESS\_KEY; secretkey=YOUR\_SECRET\_KEY" \  
 -H "Content-Type: application/json" \  
 https://cloud.tenable.com/session

Expected response:

{  
 "id": "user-id",  
 "username": "your-email@domain.com",  
 "email": "your-email@domain.com",  
 "name": "Your Name",  
 "type": "local",  
 "container\_uuid": "container-uuid",  
 "permissions": 64,  
 "user\_name": "your-email@domain.com",  
 "login\_fail\_count": 0,  
 "login\_fail\_total": 0,  
 "enabled": true,  
 "lastlogin": 1547152640  
}

## Development Environment Setup

### Step 1: Install Required Packages

# Install HTTP client and utilities  
npm install axios dotenv  
npm install --save-dev @types/node typescript  
  
# Verify installation  
node -e "console.log('Node.js packages installed successfully')"

### Step 2: Environment Configuration

Create a .env file for your API keys:

# .env file  
TENABLE\_ACCESS\_KEY=your\_access\_key\_here  
TENABLE\_SECRET\_KEY=your\_secret\_key\_here

### Step 3: Basic Connection Test

// test\_tenable\_connection.ts  
import axios, { AxiosInstance } from 'axios';  
import \* as dotenv from 'dotenv';  
  
// Load environment variables  
dotenv.config();  
  
class TenableClient {  
 private client: AxiosInstance;  
   
 constructor(accessKey: string, secretKey: string) {  
 this.client = axios.create({  
 baseURL: 'https://cloud.tenable.com',  
 headers: {  
 'X-ApiKeys': `accesskey=${accessKey}; secretkey=${secretKey}`,  
 'Content-Type': 'application/json',  
 'User-Agent': 'RAS-DASH-Cybersecurity-Platform/1.0.0'  
 }  
 });  
 }  
   
 async testConnection() {  
 try {  
 const response = await this.client.get('/session');  
 return response.data;  
 } catch (error) {  
 throw new Error(`Connection failed: ${error.response?.data?.error || error.message}`);  
 }  
 }  
   
 async getScans() {  
 try {  
 const response = await this.client.get('/scans');  
 return response.data;  
 } catch (error) {  
 throw new Error(`Failed to get scans: ${error.response?.data?.error || error.message}`);  
 }  
 }  
}  
  
async function testTenableConnection() {  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Test connection  
 const userInfo = await client.testConnection();  
 console.log(`✅ Connected successfully as: ${userInfo.email}`);  
   
 // Get basic statistics  
 const scans = await client.getScans();  
 console.log(`📊 Available scans: ${scans.scans.length}`);  
   
 return true;  
   
 } catch (error) {  
 console.error(`❌ Connection failed: ${error.message}`);  
 return false;  
 }  
}  
  
// Run test if this file is executed directly  
if (require.main === module) {  
 testTenableConnection();  
}  
  
export { TenableClient, testTenableConnection };

## Asset Data Consumption

### Understanding Asset Data Structure

Tenable assets contain comprehensive information about discovered systems:

# Asset data structure example  
{  
 "id": "asset-uuid",  
 "has\_agent": true,  
 "created\_at": "2023-01-01T00:00:00.000Z",  
 "updated\_at": "2025-01-15T12:00:00.000Z",  
 "first\_seen": "2023-01-01T00:00:00.000Z",  
 "last\_seen": "2025-01-15T12:00:00.000Z",  
 "first\_scan\_time": "2023-01-01T00:00:00.000Z",  
 "last\_scan\_time": "2025-01-15T12:00:00.000Z",  
 "last\_authenticated\_scan\_date": "2025-01-15T12:00:00.000Z",  
 "last\_licensed\_scan\_date": "2025-01-15T12:00:00.000Z",  
 "azure\_vm\_id": null,  
 "azure\_resource\_id": null,  
 "aws\_ec2\_instance\_ami\_id": "ami-12345678",  
 "aws\_ec2\_instance\_id": "i-1234567890abcdef0",  
 "aws\_owner\_id": "123456789012",  
 "aws\_availability\_zone": "us-east-1a",  
 "aws\_region": "us-east-1",  
 "aws\_vpc\_id": "vpc-12345678",  
 "aws\_ec2\_instance\_group\_name": "production-web-servers",  
 "aws\_ec2\_instance\_state\_name": "running",  
 "aws\_ec2\_instance\_type": "t3.medium",  
 "aws\_subnet\_id": "subnet-12345678",  
 "aws\_ec2\_product\_code": null,  
 "aws\_ec2\_name": "web-server-01",  
 "bios\_uuid": "12345678-1234-1234-1234-123456789abc",  
 "network\_id": "network-uuid",  
 "network\_name": "Corporate Network",  
 "exposure\_score": 750,  
 "acr\_score": 8,  
 "criticality\_rating": "high",  
 "ipv4": ["192.168.1.100"],  
 "ipv6": [],  
 "fqdn": ["web-server-01.company.com"],  
 "netbios\_name": "WEB-SERVER-01",  
 "operating\_system": ["Microsoft Windows Server 2019"],  
 "system\_type": ["general-purpose"],  
 "hostname": ["web-server-01"],  
 "agent\_uuid": ["agent-uuid"],  
 "last\_authenticated\_results": ["2025-01-15T12:00:00.000Z"],  
 "last\_unauthenticated\_results": ["2025-01-15T12:00:00.000Z"],  
 "mac\_address": ["00:0c:29:12:34:56"],  
 "manufacturer\_tpm\_id": ["tpm-id"],  
 "qualys\_asset\_id": null,  
 "qualys\_host\_id": null,  
 "servicenow\_sysid": null,  
 "sources": [  
 {  
 "name": "NESSUS\_AGENT",  
 "first\_seen": "2023-01-01T00:00:00.000Z",  
 "last\_seen": "2025-01-15T12:00:00.000Z"  
 }  
 ],  
 "tags": [  
 {  
 "key": "Environment",  
 "value": "Production",  
 "added\_by": "user-uuid",  
 "added\_at": "2023-01-01T00:00:00.000Z"  
 }  
 ],  
 "network\_interfaces": [  
 {  
 "name": "Ethernet0",  
 "virtual": false,  
 "ipv4": ["192.168.1.100"],  
 "ipv6": [],  
 "mac\_address": ["00:0c:29:12:34:56"]  
 }  
 ]  
}

### Retrieving Asset Data

#### Method 1: Simple Asset List (Small Datasets)

async function getAssetList(): Promise<any[]> {  
 /\*\*  
 \* Get basic asset list - good for small environments  
 \*/  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Get asset list  
 const response = await client.client.get('/assets');  
 const assets = response.data.assets;  
   
 console.log(`Found ${assets.length} assets`);  
   
 // Show first 5 assets  
 assets.slice(0, 5).forEach((asset: any) => {  
 const hostname = asset.hostname?.[0] || 'Unknown';  
 const ip = asset.ipv4?.[0] || 'N/A';  
 console.log(`Asset: ${hostname} - IP: ${ip}`);  
 });  
   
 return assets;  
   
 } catch (error) {  
 console.error(`Error retrieving assets: ${error.message}`);  
 return [];  
 }  
}

#### Method 2: Asset Export (Large Datasets - Recommended)

async function exportAssets(): Promise<any[]> {  
 /\*\*  
 \* Export assets using chunked approach - recommended for production  
 \*/  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Request asset export  
 const exportRequest = {  
 chunk\_size: 4000, // Optimal chunk size  
 filters: {  
 updated\_since: '2024-01-01' // Only get recent updates  
 }  
 };  
   
 const exportResponse = await client.client.post('/assets/export', exportRequest);  
 const exportUuid = exportResponse.data.export\_uuid;  
   
 console.log(`Asset export started: ${exportUuid}`);  
   
 // Wait for export to complete  
 await waitForExportCompletion(client, exportUuid);  
   
 // Download chunks  
 const allAssets = await downloadExportChunks(client, exportUuid);  
   
 console.log(`Total assets exported: ${allAssets.length}`);  
 return allAssets;  
   
 } catch (error) {  
 console.error(`Error exporting assets: ${error.message}`);  
 return [];  
 }  
}  
  
async function waitForExportCompletion(client: TenableClient, exportUuid: string): Promise<void> {  
 const maxWaitTime = 600000; // 10 minutes  
 const pollInterval = 5000; // 5 seconds  
 const startTime = Date.now();  
   
 while (Date.now() - startTime < maxWaitTime) {  
 try {  
 const statusResponse = await client.client.get(`/assets/export/${exportUuid}/status`);  
 const status = statusResponse.data.status;  
   
 console.log(`Export status: ${status}`);  
   
 if (status === 'FINISHED') {  
 return;  
 } else if (status === 'ERROR') {  
 throw new Error('Export failed');  
 }  
   
 await new Promise(resolve => setTimeout(resolve, pollInterval));  
   
 } catch (error) {  
 throw new Error(`Error checking export status: ${error.message}`);  
 }  
 }  
   
 throw new Error('Export timeout');  
}  
  
async function downloadExportChunks(client: TenableClient, exportUuid: string): Promise<any[]> {  
 const allAssets: any[] = [];  
 let chunkId = 1;  
   
 while (true) {  
 try {  
 const chunkResponse = await client.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 new Error(`Error downloading chunk ${chunkId}: ${error.message}`);  
 }  
 }  
   
 return allAssets;  
}

#### Method 3: Filtered Asset Export

def export\_filtered\_assets():  
 """Export assets with specific filters"""  
 try:  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Define filters  
 filters = {  
 'has\_agent': True, # Only assets with agents  
 'operating\_systems': ['Windows', 'Linux'], # Specific OS types  
 'tags': ['Environment:Production'], # Production assets only  
 'updated\_since': '2025-01-01' # Recent updates only  
 }  
   
 # Request filtered export  
 export\_uuid = tio.exports.assets(  
 chunk\_size=2000,  
 filters=filters  
 )  
   
 assets = []  
 for chunk in tio.exports.download\_chunks(export\_uuid):  
 assets.extend(chunk)  
   
 return assets  
   
 except Exception as e:  
 print(f"Error with filtered export: {str(e)}")  
 return []

## Vulnerability Data Consumption

### Understanding Vulnerability Data Structure

Vulnerability data provides detailed security findings:

# Vulnerability data structure example  
{  
 "asset": {  
 "id": "asset-uuid",  
 "uuid": "asset-uuid",  
 "hostname": "web-server-01.company.com",  
 "fqdn": "web-server-01.company.com",  
 "ipv4": "192.168.1.100",  
 "ipv6": null,  
 "netbios\_name": "WEB-SERVER-01",  
 "operating\_system": "Microsoft Windows Server 2019",  
 "agent\_uuid": "agent-uuid",  
 "last\_authenticated\_results": "2025-01-15T12:00:00.000Z"  
 },  
 "plugin": {  
 "id": 19506,  
 "name": "Nessus Scan Information",  
 "family": "Settings",  
 "modification\_date": "2025-01-01",  
 "publication\_date": "1999-01-01",  
 "risk\_factor": "None",  
 "solution": "n/a",  
 "synopsis": "This plugin displays information about the Nessus scan.",  
 "description": "This plugin displays, for each target, information about how the scan was performed, such as:\n\n- Which credential types were used\n- The port range that was scanned\n- The duration of the scan\n- etc.",  
 "see\_also": [],  
 "version": "1.0"  
 },  
 "scan": {  
 "id": 12345,  
 "uuid": "scan-uuid",  
 "schedule\_id": "schedule-uuid",  
 "started\_at": "2025-01-15T10:00:00.000Z",  
 "completed\_at": "2025-01-15T12:00:00.000Z"  
 },  
 "output": "Nessus version : 10.4.2 (#68)\nPlugin feed version : 202501150947\n...",  
 "severity": "info",  
 "severity\_id": 0,  
 "severity\_default\_id": 0,  
 "severity\_modification\_type": "none",  
 "first\_found": "2023-01-01T00:00:00.000Z",  
 "last\_found": "2025-01-15T12:00:00.000Z",  
 "state": "open",  
 "indexed": "2025-01-15T12:30:00.000Z"  
}

### Retrieving Vulnerability Data

#### Method 1: Workbench Vulnerabilities (Small Datasets)

async function getVulnerabilitySummary(): Promise<any> {  
 /\*\*  
 \* Get vulnerability summary from workbench - good for dashboards  
 \*/  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Get vulnerability summary  
 const response = await client.client.get('/workbenches/vulnerabilities');  
 const vulns = response.data;  
   
 // Organize by severity  
 const severityCounts: { [key: string]: number } = {};  
   
 vulns.vulnerabilities.forEach((vuln: any) => {  
 const severity = vuln.severity\_name;  
 severityCounts[severity] = (severityCounts[severity] || 0) + vuln.count;  
 });  
   
 console.log("Vulnerability Summary:");  
 Object.entries(severityCounts).forEach(([severity, count]) => {  
 console.log(` ${severity.charAt(0).toUpperCase() + severity.slice(1)}: ${count}`);  
 });  
   
 return vulns;  
   
 } catch (error) {  
 console.error(`Error retrieving vulnerability summary: ${error.message}`);  
 return {};  
 }  
}

#### Method 2: Vulnerability Export (Large Datasets - Recommended)

async function exportVulnerabilities(): Promise<any[]> {  
 /\*\*  
 \* Export detailed vulnerability data - recommended for production  
 \*/  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Request vulnerability export  
 const exportRequest = {  
 filters: {  
 severity: ['critical', 'high', 'medium'], // Filter by severity  
 state: ['open'], // Only open vulnerabilities  
 since: 7 // Last 7 days  
 }  
 };  
   
 const exportResponse = await client.client.post('/vulns/export', exportRequest);  
 const exportUuid = exportResponse.data.export\_uuid;  
   
 console.log(`Vulnerability export started: ${exportUuid}`);  
   
 // Wait for export to complete  
 await waitForVulnExportCompletion(client, exportUuid);  
   
 // Download vulnerability data  
 const allVulns = await downloadVulnExportChunks(client, exportUuid);  
   
 console.log(`Total vulnerabilities exported: ${allVulns.length}`);  
 return allVulns;  
   
 } catch (error) {  
 console.error(`Error exporting vulnerabilities: ${error.message}`);  
 return [];  
 }  
}  
  
async function waitForVulnExportCompletion(client: TenableClient, exportUuid: string): Promise<void> {  
 const maxWaitTime = 600000; // 10 minutes  
 const pollInterval = 5000; // 5 seconds  
 const startTime = Date.now();  
   
 while (Date.now() - startTime < maxWaitTime) {  
 try {  
 const statusResponse = await client.client.get(`/vulns/export/${exportUuid}/status`);  
 const status = statusResponse.data.status;  
   
 console.log(`Vulnerability export status: ${status}`);  
   
 if (status === 'FINISHED') {  
 return;  
 } else if (status === 'ERROR') {  
 throw new Error('Vulnerability export failed');  
 }  
   
 await new Promise(resolve => setTimeout(resolve, pollInterval));  
   
 } catch (error) {  
 throw new Error(`Error checking vulnerability export status: ${error.message}`);  
 }  
 }  
   
 throw new Error('Vulnerability export timeout');  
}  
  
async function downloadVulnExportChunks(client: TenableClient, exportUuid: string): Promise<any[]> {  
 const allVulns: any[] = [];  
 let chunkId = 1;  
   
 while (true) {  
 try {  
 const chunkResponse = await client.client.get(`/vulns/export/${exportUuid}/chunks/${chunkId}`);  
 const chunk = chunkResponse.data;  
   
 if (!chunk || chunk.length === 0) {  
 break;  
 }  
   
 allVulns.push(...chunk);  
 console.log(`Downloaded chunk ${chunkId} with ${chunk.length} vulnerabilities`);  
 chunkId++;  
   
 } catch (error) {  
 if (error.response?.status === 404) {  
 // No more chunks  
 break;  
 }  
 throw new Error(`Error downloading vulnerability chunk ${chunkId}: ${error.message}`);  
 }  
 }  
   
 return allVulns;  
}

#### Method 3: Asset-Specific Vulnerabilities

def get\_asset\_vulnerabilities(asset\_id):  
 """Get vulnerabilities for a specific asset"""  
 try:  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Get vulnerabilities for specific asset  
 vulns = tio.workbenches.asset\_vulnerabilities(asset\_id)  
   
 print(f"Asset {asset\_id} has {len(vulns['vulnerabilities'])} vulnerabilities")  
   
 # Group by severity  
 critical = [v for v in vulns['vulnerabilities'] if v['severity'] == 4]  
 high = [v for v in vulns['vulnerabilities'] if v['severity'] == 3]  
 medium = [v for v in vulns['vulnerabilities'] if v['severity'] == 2]  
   
 print(f" Critical: {len(critical)}")  
 print(f" High: {len(high)}")  
 print(f" Medium: {len(medium)}")  
   
 return vulns  
   
 except Exception as e:  
 print(f"Error retrieving asset vulnerabilities: {str(e)}")  
 return {}

## Data Volume Management

### Understanding Data Volumes

#### Typical Enterprise Volumes

* **Small Organization**: 100-1,000 assets, 1,000-10,000 vulnerabilities
* **Medium Organization**: 1,000-10,000 assets, 10,000-100,000 vulnerabilities
* **Large Organization**: 10,000+ assets, 100,000+ vulnerabilities

#### API Limits

* **Asset Export**: 100-10,000 assets per chunk (recommended: 4,000-5,000)
* **Vulnerability Export**: 50-5,000 assets per chunk (recommended: 1,000-3,000)
* **Workbench Limit**: Maximum 5,000 records per request
* **Concurrency**: Maximum 10 concurrent exports
* **Rate Limiting**: Dynamic based on load

### Efficient Data Processing

#### Chunked Processing Strategy

def process\_large\_dataset():  
 """Efficiently process large datasets with chunking"""  
 try:  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Start export with optimal chunk size  
 export\_uuid = tio.exports.assets(  
 chunk\_size=4000,  
 filters={'updated\_since': '2024-01-01'}  
 )  
   
 # Process chunks as they become available  
 total\_processed = 0  
   
 for chunk\_num, chunk in enumerate(tio.exports.download\_chunks(export\_uuid)):  
 print(f"Processing chunk {chunk\_num + 1} with {len(chunk)} records")  
   
 # Process chunk data  
 processed\_chunk = process\_asset\_chunk(chunk)  
   
 # Store or transmit processed data  
 store\_processed\_data(processed\_chunk)  
   
 total\_processed += len(chunk)  
 print(f"Total processed: {total\_processed}")  
   
 print(f"✅ Successfully processed {total\_processed} assets")  
   
 except Exception as e:  
 print(f"❌ Error in bulk processing: {str(e)}")  
  
def process\_asset\_chunk(assets):  
 """Process a chunk of assets"""  
 processed = []  
   
 for asset in assets:  
 # Transform data for RAS-DASH format  
 processed\_asset = {  
 'id': asset.get('id'),  
 'hostname': asset.get('hostname', ['Unknown'])[0] if asset.get('hostname') else 'Unknown',  
 'ip\_address': asset.get('ipv4', ['Unknown'])[0] if asset.get('ipv4') else 'Unknown',  
 'operating\_system': asset.get('operating\_system', ['Unknown'])[0] if asset.get('operating\_system') else 'Unknown',  
 'exposure\_score': asset.get('exposure\_score', 0),  
 'criticality': asset.get('criticality\_rating', 'unknown'),  
 'last\_seen': asset.get('last\_seen'),  
 'has\_agent': asset.get('has\_agent', False),  
 'cloud\_provider': 'aws' if asset.get('aws\_ec2\_instance\_id') else 'unknown',  
 'tags': [f"{tag['key']}:{tag['value']}" for tag in asset.get('tags', [])]  
 }  
 processed.append(processed\_asset)  
   
 return processed  
  
def store\_processed\_data(data):  
 """Store processed data in RAS-DASH database"""  
 # Implementation depends on your database setup  
 print(f"Storing {len(data)} processed records")

#### Delta Synchronization

def delta\_sync():  
 """Perform incremental data synchronization"""  
 try:  
 # Get last sync timestamp from database  
 last\_sync = get\_last\_sync\_timestamp() # Implement based on your DB  
   
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Export only updated data  
 export\_uuid = tio.exports.assets(  
 filters={'updated\_since': last\_sync.strftime('%Y-%m-%d')}  
 )  
   
 updated\_count = 0  
 for chunk in tio.exports.download\_chunks(export\_uuid):  
 # Process and store updates  
 process\_asset\_updates(chunk)  
 updated\_count += len(chunk)  
   
 # Update sync timestamp  
 update\_last\_sync\_timestamp() # Implement based on your DB  
   
 print(f"✅ Delta sync completed: {updated\_count} updates")  
   
 except Exception as e:  
 print(f"❌ Delta sync failed: {str(e)}")

## Best Practices

### 1. Authentication Security

// ✅ Good: Use environment variables  
const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
);  
  
// ❌ Bad: Hardcoded credentials  
const client = new TenableClient(  
 'your\_key\_here',  
 'your\_secret\_here'  
);

### 2. Error Handling

async function robustApiCall(): Promise<any> {  
 /\*\*  
 \* Implement robust error handling  
 \*/  
 const maxRetries = 3;  
 const retryDelay = 5000; // 5 seconds  
   
 for (let attempt = 0; attempt < maxRetries; attempt++) {  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 // Make API call  
 const result = await client.getScans();  
 return result;  
   
 } catch (error: any) {  
 if (attempt < maxRetries - 1) {  
 console.log(`Attempt ${attempt + 1} failed: ${error.message}`);  
 console.log(`Retrying in ${retryDelay / 1000} seconds...`);  
 await new Promise(resolve => setTimeout(resolve, retryDelay));  
 } else {  
 console.log(`All ${maxRetries} attempts failed`);  
 throw error;  
 }  
 }  
 }  
}

### 3. Rate Limit Handling

async function handleRateLimits(): Promise<any> {  
 /\*\*  
 \* Handle rate limiting gracefully  
 \*/  
 try {  
 const client = new TenableClient(  
 process.env.TENABLE\_ACCESS\_KEY!,  
 process.env.TENABLE\_SECRET\_KEY!  
 );  
   
 const result = await exportAssets();  
 return result;  
   
 } catch (error: any) {  
 if (error.response?.status === 429) { // Rate limited  
 const retryAfter = parseInt(error.response.headers['retry-after']) || 60;  
 console.log(`Rate limited. Waiting ${retryAfter} seconds...`);  
 await new Promise(resolve => setTimeout(resolve, retryAfter \* 1000));  
 // Retry the request  
 return handleRateLimits();  
 } else {  
 throw error;  
 }  
 }  
}

### 4. Data Validation

function validateAssetData(asset: any): boolean {  
 /\*\*  
 \* Validate asset data before processing  
 \*/  
 const requiredFields = ['id', 'hostname', 'ipv4'];  
   
 for (const field of requiredFields) {  
 if (!(field in asset) || !asset[field]) {  
 console.log(`Warning: Asset missing required field: ${field}`);  
 return false;  
 }  
 }  
   
 // Validate IP format  
 const ipAddress = asset.ipv4?.[0];  
 if (ipAddress && !isValidIp(ipAddress)) {  
 console.log(`Warning: Invalid IP address: ${ipAddress}`);  
 return false;  
 }  
   
 return true;  
}  
  
function isValidIp(ip: string): boolean {  
 /\*\*  
 \* Validate IP address format  
 \*/  
 const ipv4Regex = /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/;  
 const ipv6Regex = /^(([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}|([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}|([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}|([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}|[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})|:((:[0-9a-fA-F]{1,4}){1,7}|:)|fe80:(:[0-9a-fA-F]{0,4}){0,4}%[0-9a-zA-Z]{1,}|::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])|([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9]))$/;  
   
 return ipv4Regex.test(ip) || ipv6Regex.test(ip);  
}

## Integration with RAS-DASH

### Database Schema Integration

# Example integration with RAS-DASH database schema  
def integrate\_tenable\_assets():  
 """Integrate Tenable assets with RAS-DASH database"""  
 try:  
 # Get Tenable data  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 export\_uuid = tio.exports.assets()  
   
 # Process and integrate with RAS-DASH  
 for chunk in tio.exports.download\_chunks(export\_uuid):  
 for asset in chunk:  
 # Map to RAS-DASH asset format  
 ras\_asset = {  
 'external\_id': asset['id'],  
 'source': 'tenable',  
 'hostname': asset.get('hostname', ['Unknown'])[0] if asset.get('hostname') else 'Unknown',  
 'ip\_address': asset.get('ipv4', ['Unknown'])[0] if asset.get('ipv4') else 'Unknown',  
 'operating\_system': asset.get('operating\_system', ['Unknown'])[0] if asset.get('operating\_system') else 'Unknown',  
 'risk\_score': asset.get('exposure\_score', 0),  
 'criticality': map\_criticality(asset.get('criticality\_rating')),  
 'last\_updated': asset.get('updated\_at'),  
 'cloud\_provider': detect\_cloud\_provider(asset),  
 'agent\_installed': asset.get('has\_agent', False),  
 'tags': extract\_tags(asset.get('tags', [])),  
 'metadata': {  
 'aws\_instance\_id': asset.get('aws\_ec2\_instance\_id'),  
 'aws\_region': asset.get('aws\_region'),  
 'network\_interfaces': asset.get('network\_interfaces', [])  
 }  
 }  
   
 # Insert or update in RAS-DASH database  
 upsert\_asset(ras\_asset)  
   
 print("✅ Tenable asset integration completed")  
   
 except Exception as e:  
 print(f"❌ Integration failed: {str(e)}")  
  
def map\_criticality(tenable\_criticality):  
 """Map Tenable criticality to RAS-DASH format"""  
 mapping = {  
 'low': 'Low',  
 'medium': 'Medium',  
 'high': 'High',  
 'critical': 'Critical'  
 }  
 return mapping.get(tenable\_criticality, 'Unknown')  
  
def detect\_cloud\_provider(asset):  
 """Detect cloud provider from asset data"""  
 if asset.get('aws\_ec2\_instance\_id'):  
 return 'AWS'  
 elif asset.get('azure\_vm\_id'):  
 return 'Azure'  
 elif asset.get('gcp\_instance\_id'):  
 return 'GCP'  
 else:  
 return 'On-Premise'  
  
def extract\_tags(tenable\_tags):  
 """Extract and format tags for RAS-DASH"""  
 return [f"{tag['key']}:{tag['value']}" for tag in tenable\_tags]

### Scheduled Synchronization

def setup\_scheduled\_sync():  
 """Setup scheduled synchronization with Tenable"""  
 import schedule  
 import time  
   
 # Schedule full sync weekly  
 schedule.every().sunday.at("02:00").do(full\_sync)  
   
 # Schedule delta sync every 4 hours  
 schedule.every(4).hours.do(delta\_sync)  
   
 # Schedule vulnerability sync every 2 hours  
 schedule.every(2).hours.do(sync\_vulnerabilities)  
   
 print("Scheduled synchronization configured")  
   
 # Keep running  
 while True:  
 schedule.run\_pending()  
 time.sleep(60) # Check every minute  
  
def full\_sync():  
 """Perform full synchronization"""  
 print("Starting full synchronization...")  
 integrate\_tenable\_assets()  
 integrate\_tenable\_vulnerabilities()  
 print("Full synchronization completed")  
  
def sync\_vulnerabilities():  
 """Synchronize vulnerability data"""  
 print("Starting vulnerability synchronization...")  
 # Implementation similar to asset sync  
 print("Vulnerability synchronization completed")

## Troubleshooting

### Common Issues and Solutions

#### 1. Authentication Errors

Error: 401 Unauthorized

**Solutions:** - Verify API keys are correct - Check if keys have expired or been regenerated - Ensure proper header format: X-ApiKeys: accesskey=XXX; secretkey=YYY

#### 2. Rate Limiting

Error: 429 Too Many Requests

**Solutions:** - Implement exponential backoff - Reduce request frequency - Use chunked exports instead of frequent API calls

#### 3. Export Timeout

Error: Export expired or not found

**Solutions:** - Download export chunks within 24 hours - Check export status before downloading - Implement automatic retry logic

#### 4. Data Volume Issues

Error: Request timeout or memory issues

**Solutions:** - Use smaller chunk sizes - Implement streaming processing - Process data in batches

### Debugging Tools

def debug\_api\_call():  
 """Debug API calls with detailed logging"""  
 import logging  
   
 # Enable debug logging  
 logging.basicConfig(level=logging.DEBUG)  
   
 try:  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Enable debug mode  
 tio.\_session.debug = True  
   
 # Make test call  
 result = tio.session.details()  
 print(f"Debug result: {result}")  
   
 except Exception as e:  
 print(f"Debug error: {str(e)}")  
 import traceback  
 traceback.print\_exc()

## Production Considerations

### 1. Security Best Practices

* Store API keys in secure environment variables or key vault
* Use dedicated service accounts for API access
* Implement API key rotation procedures
* Monitor API usage for anomalies

### 2. Performance Optimization

* Use export APIs for large datasets
* Implement connection pooling
* Cache frequently accessed data
* Use compression for data transfer

### 3. Monitoring and Alerting

def setup\_monitoring():  
 """Setup monitoring for Tenable integration"""  
 import logging  
   
 # Configure logging  
 logging.basicConfig(  
 level=logging.INFO,  
 format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  
 handlers=[  
 logging.FileHandler('tenable\_integration.log'),  
 logging.StreamHandler()  
 ]  
 )  
   
 # Log API usage  
 logger = logging.getLogger('tenable\_integration')  
 logger.info("Tenable integration monitoring started")  
   
 return logger  
  
def monitor\_api\_health():  
 """Monitor API connectivity and performance"""  
 import time  
   
 start\_time = time.time()  
   
 try:  
 tio = TenableIO(  
 access\_key=os.getenv('TENABLE\_ACCESS\_KEY'),  
 secret\_key=os.getenv('TENABLE\_SECRET\_KEY')  
 )  
   
 # Test API connectivity  
 tio.session.details()  
   
 response\_time = time.time() - start\_time  
   
 if response\_time > 5.0:  
 print(f"⚠️ Slow API response: {response\_time:.2f}s")  
 else:  
 print(f"✅ API healthy: {response\_time:.2f}s")  
   
 except Exception as e:  
 print(f"❌ API health check failed: {str(e)}")

### 4. Data Backup and Recovery

def backup\_sync\_state():  
 """Backup synchronization state"""  
 import json  
 from datetime import datetime  
   
 sync\_state = {  
 'last\_asset\_sync': get\_last\_sync\_timestamp('assets'),  
 'last\_vuln\_sync': get\_last\_sync\_timestamp('vulnerabilities'),  
 'total\_assets': get\_asset\_count(),  
 'total\_vulnerabilities': get\_vulnerability\_count(),  
 'backup\_timestamp': datetime.now().isoformat()  
 }  
   
 # Save backup  
 with open(f'tenable\_sync\_backup\_{datetime.now().strftime("%Y%m%d\_%H%M%S")}.json', 'w') as f:  
 json.dump(sync\_state, f, indent=2)  
   
 print("Sync state backed up successfully")

### ✅ Getting Started Checklist

1. **Sign up** for Tenable 60-day free trial
2. **Generate** API access and secret keys
3. **Install** PyTenable library
4. **Test** basic connectivity
5. **Implement** asset and vulnerability data consumption
6. **Setup** delta synchronization
7. **Monitor** integration performance

### 📊 Expected Data Volumes

* **Assets**: Thousands to tens of thousands per organization
* **Vulnerabilities**: Tens of thousands to hundreds of thousands
* **API Calls**: Multiple requests needed for complete data export
* **Processing**: Chunked approach required for large datasets

### 🔧 Integration Benefits for RAS-DASH

* **Comprehensive Asset Discovery**: Detailed inventory from Tenable agents and scans
* **Vulnerability Intelligence**: Real-time vulnerability data with CVSS scoring
* **Risk Prioritization**: Exposure scores and criticality ratings
* **Cloud Asset Management**: AWS, Azure, GCP asset identification
* **Compliance Integration**: Regulatory compliance status and findings

### 🚀 Next Steps

1. **Implement** basic integration following this guide
2. **Test** with sample data from trial account
3. **Develop** RAS-DASH specific data transformations
4. **Setup** automated synchronization workflows
5. **Monitor** and optimize performance

The integration will significantly enhance RAS-DASH’s cybersecurity intelligence capabilities by providing comprehensive, real-time asset and vulnerability data from Tenable’s industry-leading platform.

## Technical Implementation Details for Developer Handoff

### 1. Database Schema Requirements

#### New Tables to Create

##### Tenable Integration Configuration

-- Tenable API configuration and credentials  
CREATE TABLE tenable\_config (  
 id SERIAL PRIMARY KEY,  
 name VARCHAR(255) NOT NULL UNIQUE,  
 access\_key\_encrypted TEXT NOT NULL,  
 secret\_key\_encrypted TEXT NOT NULL,  
 base\_url VARCHAR(255) DEFAULT 'https://cloud.tenable.com',  
 is\_active BOOLEAN DEFAULT true,  
 last\_sync\_at TIMESTAMP WITH TIME ZONE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Tenable sync jobs and status tracking  
CREATE TABLE tenable\_sync\_jobs (  
 id SERIAL PRIMARY KEY,  
 config\_id INTEGER REFERENCES tenable\_config(id),  
 job\_type VARCHAR(50) NOT NULL, -- 'assets', 'vulnerabilities', 'scans'  
 status VARCHAR(50) NOT NULL, -- 'pending', 'running', 'completed', 'failed'  
 export\_uuid VARCHAR(255),  
 total\_records INTEGER DEFAULT 0,  
 processed\_records INTEGER DEFAULT 0,  
 error\_message TEXT,  
 started\_at TIMESTAMP WITH TIME ZONE,  
 completed\_at TIMESTAMP WITH TIME ZONE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

##### Tenable Asset Data

-- Tenable assets with full metadata  
CREATE TABLE tenable\_assets (  
 id SERIAL PRIMARY KEY,  
 tenable\_id VARCHAR(255) NOT NULL UNIQUE,  
 config\_id INTEGER REFERENCES tenable\_config(id),  
 hostname VARCHAR(255)[],  
 ipv4 VARCHAR(15)[],  
 ipv6 VARCHAR(45)[],  
 fqdn VARCHAR(255)[],  
 netbios\_name VARCHAR(255),  
 operating\_system VARCHAR(255)[],  
 system\_type VARCHAR(255)[],  
 has\_agent BOOLEAN DEFAULT false,  
 agent\_uuid VARCHAR(255)[],  
 exposure\_score INTEGER DEFAULT 0,  
 acr\_score INTEGER DEFAULT 0,  
 criticality\_rating VARCHAR(50),  
 first\_seen TIMESTAMP WITH TIME ZONE,  
 last\_seen TIMESTAMP WITH TIME ZONE,  
 first\_scan\_time TIMESTAMP WITH TIME ZONE,  
 last\_scan\_time TIMESTAMP WITH TIME ZONE,  
 last\_authenticated\_scan\_date TIMESTAMP WITH TIME ZONE,  
   
 -- Cloud metadata  
 aws\_ec2\_instance\_id VARCHAR(255),  
 aws\_ec2\_instance\_ami\_id VARCHAR(255),  
 aws\_owner\_id VARCHAR(255),  
 aws\_availability\_zone VARCHAR(255),  
 aws\_region VARCHAR(255),  
 aws\_vpc\_id VARCHAR(255),  
 aws\_subnet\_id VARCHAR(255),  
 aws\_ec2\_instance\_type VARCHAR(255),  
 aws\_ec2\_instance\_state\_name VARCHAR(255),  
 aws\_ec2\_name VARCHAR(255),  
   
 azure\_vm\_id VARCHAR(255),  
 azure\_resource\_id VARCHAR(255),  
   
 -- Network information  
 bios\_uuid VARCHAR(255),  
 network\_id VARCHAR(255),  
 network\_name VARCHAR(255),  
 mac\_address VARCHAR(17)[],  
 manufacturer\_tpm\_id VARCHAR(255)[],  
   
 -- Integration tracking  
 last\_updated\_tenable TIMESTAMP WITH TIME ZONE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Tenable asset tags  
CREATE TABLE tenable\_asset\_tags (  
 id SERIAL PRIMARY KEY,  
 asset\_id INTEGER REFERENCES tenable\_assets(id),  
 key VARCHAR(255) NOT NULL,  
 value VARCHAR(255) NOT NULL,  
 added\_by VARCHAR(255),  
 added\_at TIMESTAMP WITH TIME ZONE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);  
  
-- Tenable asset network interfaces  
CREATE TABLE tenable\_asset\_network\_interfaces (  
 id SERIAL PRIMARY KEY,  
 asset\_id INTEGER REFERENCES tenable\_assets(id),  
 name VARCHAR(255),  
 virtual BOOLEAN DEFAULT false,  
 ipv4 VARCHAR(15)[],  
 ipv6 VARCHAR(45)[],  
 mac\_address VARCHAR(17)[],  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

##### Tenable Vulnerability Data

-- Tenable vulnerabilities  
CREATE TABLE tenable\_vulnerabilities (  
 id SERIAL PRIMARY KEY,  
 asset\_id INTEGER REFERENCES tenable\_assets(id),  
 config\_id INTEGER REFERENCES tenable\_config(id),  
 plugin\_id INTEGER NOT NULL,  
 plugin\_name VARCHAR(500),  
 plugin\_family VARCHAR(255),  
 plugin\_modification\_date DATE,  
 plugin\_publication\_date DATE,  
 plugin\_risk\_factor VARCHAR(50),  
 plugin\_solution TEXT,  
 plugin\_synopsis TEXT,  
 plugin\_description TEXT,  
 plugin\_version VARCHAR(50),  
   
 -- Scan information  
 scan\_id INTEGER,  
 scan\_uuid VARCHAR(255),  
 scan\_schedule\_id VARCHAR(255),  
 scan\_started\_at TIMESTAMP WITH TIME ZONE,  
 scan\_completed\_at TIMESTAMP WITH TIME ZONE,  
   
 -- Vulnerability details  
 severity VARCHAR(50),  
 severity\_id INTEGER,  
 severity\_default\_id INTEGER,  
 severity\_modification\_type VARCHAR(50),  
 state VARCHAR(50), -- 'open', 'reopened', 'fixed'  
 first\_found TIMESTAMP WITH TIME ZONE,  
 last\_found TIMESTAMP WITH TIME ZONE,  
   
 -- CVSS and scoring  
 cvss\_base\_score DECIMAL(3,1),  
 cvss\_temporal\_score DECIMAL(3,1),  
 cvss\_environmental\_score DECIMAL(3,1),  
 cvss\_vector VARCHAR(255),  
 vpr\_score DECIMAL(4,1),  
   
 -- Output and evidence  
 output TEXT,  
 port INTEGER,  
 protocol VARCHAR(10),  
   
 indexed\_at TIMESTAMP WITH TIME ZONE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
   
 UNIQUE(asset\_id, plugin\_id, scan\_id)  
);

##### Tenable Scans

-- Tenable scan information  
CREATE TABLE tenable\_scans (  
 id SERIAL PRIMARY KEY,  
 tenable\_scan\_id INTEGER NOT NULL,  
 config\_id INTEGER REFERENCES tenable\_config(id),  
 scan\_uuid VARCHAR(255) UNIQUE,  
 name VARCHAR(500),  
 description TEXT,  
 policy\_id INTEGER,  
 scanner\_id INTEGER,  
 scanner\_name VARCHAR(255),  
 folder\_id INTEGER,  
 type VARCHAR(50),  
 status VARCHAR(50),  
 starttime TIMESTAMP WITH TIME ZONE,  
 endtime TIMESTAMP WITH TIME ZONE,  
 timezone VARCHAR(100),  
   
 -- Scan targets and settings  
 text\_targets TEXT,  
 target\_network\_uuid VARCHAR(255),  
 scanner\_start TIMESTAMP WITH TIME ZONE,  
 scanner\_end TIMESTAMP WITH TIME ZONE,  
   
 -- Scan statistics  
 hosts\_total INTEGER DEFAULT 0,  
 hosts\_scanned INTEGER DEFAULT 0,  
 vulnerabilities\_total INTEGER DEFAULT 0,  
   
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()  
);

#### Extend Existing Tables

-- Add Tenable integration fields to existing assets table  
ALTER TABLE assets ADD COLUMN tenable\_asset\_id INTEGER REFERENCES tenable\_assets(id);  
ALTER TABLE assets ADD COLUMN last\_tenable\_sync TIMESTAMP WITH TIME ZONE;  
  
-- Add Tenable integration fields to existing vulnerabilities table  
ALTER TABLE vulnerabilities ADD COLUMN tenable\_vulnerability\_id INTEGER REFERENCES tenable\_vulnerabilities(id);  
ALTER TABLE vulnerabilities ADD COLUMN tenable\_plugin\_id INTEGER;  
ALTER TABLE vulnerabilities ADD COLUMN last\_tenable\_sync TIMESTAMP WITH TIME ZONE;  
  
-- Create indexes for performance  
CREATE INDEX idx\_tenable\_assets\_tenable\_id ON tenable\_assets(tenable\_id);  
CREATE INDEX idx\_tenable\_assets\_config\_id ON tenable\_assets(config\_id);  
CREATE INDEX idx\_tenable\_assets\_last\_seen ON tenable\_assets(last\_seen);  
CREATE INDEX idx\_tenable\_vulnerabilities\_asset\_id ON tenable\_vulnerabilities(asset\_id);  
CREATE INDEX idx\_tenable\_vulnerabilities\_severity ON tenable\_vulnerabilities(severity);  
CREATE INDEX idx\_tenable\_vulnerabilities\_state ON tenable\_vulnerabilities(state);  
CREATE INDEX idx\_tenable\_sync\_jobs\_status ON tenable\_sync\_jobs(status);

### 2. Environment Configuration (.env Updates)

# Tenable API Configuration  
TENABLE\_ACCESS\_KEY=your\_tenable\_access\_key\_here  
TENABLE\_SECRET\_KEY=your\_tenable\_secret\_key\_here  
TENABLE\_BASE\_URL=https://cloud.tenable.com  
TENABLE\_SYNC\_ENABLED=true  
  
# Sync Configuration  
TENABLE\_SYNC\_INTERVAL\_HOURS=4  
TENABLE\_FULL\_SYNC\_INTERVAL\_HOURS=24  
TENABLE\_ASSET\_CHUNK\_SIZE=4000  
TENABLE\_VULN\_CHUNK\_SIZE=2000  
TENABLE\_MAX\_CONCURRENT\_EXPORTS=3  
TENABLE\_EXPORT\_TIMEOUT\_MINUTES=30  
  
# Rate Limiting  
TENABLE\_RATE\_LIMIT\_REQUESTS\_PER\_MINUTE=100  
TENABLE\_RATE\_LIMIT\_BURST=20  
TENABLE\_RETRY\_MAX\_ATTEMPTS=3  
TENABLE\_RETRY\_DELAY\_SECONDS=5  
  
# Encryption for stored credentials  
TENABLE\_ENCRYPTION\_KEY=your\_32\_character\_encryption\_key\_here  
  
# Monitoring and Logging  
TENABLE\_LOG\_LEVEL=info  
TENABLE\_ENABLE\_METRICS=true  
TENABLE\_WEBHOOK\_URL=optional\_webhook\_for\_notifications

### 3. Service Layer Implementation

#### TenableApiService

**Location**: server/services/tenableApiService.ts

class TenableApiService {  
 // Connection and authentication methods  
 async testConnection(): Promise<boolean>  
 // Tests API connectivity and credentials  
   
 async getAccountInfo(): Promise<TenableAccount>  
 // Retrieves account information and permissions  
   
 // Asset management methods  
 async getAssets(filters?: AssetFilters): Promise<TenableAsset[]>  
 // Retrieves asset list with optional filtering  
   
 async exportAssets(filters?: AssetFilters): Promise<string>  
 // Starts asset export job, returns export UUID  
   
 async getAssetDetails(assetId: string): Promise<TenableAssetDetails>  
 // Gets detailed information for specific asset  
   
 // Vulnerability management methods  
 async getVulnerabilities(filters?: VulnFilters): Promise<TenableVulnerability[]>  
 // Retrieves vulnerability list with filtering  
   
 async exportVulnerabilities(filters?: VulnFilters): Promise<string>  
 // Starts vulnerability export job, returns export UUID  
   
 async getAssetVulnerabilities(assetId: string): Promise<TenableVulnerability[]>  
 // Gets all vulnerabilities for specific asset  
   
 // Export management methods  
 async getExportStatus(exportUuid: string): Promise<ExportStatus>  
 // Checks status of export job  
   
 async downloadExportChunks(exportUuid: string): Promise<any[]>  
 // Downloads all chunks from completed export  
   
 // Scan management methods  
 async getScans(): Promise<TenableScan[]>  
 // Retrieves scan list  
   
 async getScanDetails(scanId: number): Promise<TenableScanDetails>  
 // Gets detailed scan information  
   
 async getScanResults(scanId: number): Promise<TenableScanResults>  
 // Downloads scan results  
}

#### TenableSyncService

**Location**: server/services/tenableSyncService.ts

class TenableSyncService {  
 // Synchronization orchestration  
 async performFullSync(): Promise<SyncResult>  
 // Performs complete data synchronization  
   
 async performDeltaSync(): Promise<SyncResult>  
 // Performs incremental synchronization  
   
 async syncAssets(since?: Date): Promise<AssetSyncResult>  
 // Synchronizes asset data with optional delta  
   
 async syncVulnerabilities(since?: Date): Promise<VulnSyncResult>  
 // Synchronizes vulnerability data with optional delta  
   
 async syncScans(since?: Date): Promise<ScanSyncResult>  
 // Synchronizes scan data with optional delta  
   
 // Job management  
 async createSyncJob(type: SyncJobType): Promise<TenableSyncJob>  
 // Creates new sync job record  
   
 async updateSyncJob(jobId: number, updates: Partial<TenableSyncJob>): Promise<void>  
 // Updates sync job status and progress  
   
 async getSyncJobs(filters?: SyncJobFilters): Promise<TenableSyncJob[]>  
 // Retrieves sync job history  
   
 // Data transformation  
 async transformAssetData(tenableAssets: TenableAsset[]): Promise<Asset[]>  
 // Converts Tenable asset format to RAS-DASH format  
   
 async transformVulnerabilityData(tenableVulns: TenableVulnerability[]): Promise<Vulnerability[]>  
 // Converts Tenable vulnerability format to RAS-DASH format  
   
 // Conflict resolution  
 async resolveAssetConflicts(existing: Asset, incoming: TenableAsset): Promise<Asset>  
 // Handles conflicts during asset updates  
   
 async resolveVulnerabilityConflicts(existing: Vulnerability, incoming: TenableVulnerability): Promise<Vulnerability>  
 // Handles conflicts during vulnerability updates  
}

#### TenableDataService

**Location**: server/services/tenableDataService.ts

class TenableDataService {  
 // Database operations for Tenable data  
 async saveAssets(assets: TenableAsset[]): Promise<void>  
 // Bulk insert/update Tenable assets  
   
 async saveVulnerabilities(vulnerabilities: TenableVulnerability[]): Promise<void>  
 // Bulk insert/update Tenable vulnerabilities  
   
 async saveScans(scans: TenableScan[]): Promise<void>  
 // Bulk insert/update Tenable scans  
   
 // Query methods  
 async getAssetByTenableId(tenableId: string): Promise<TenableAsset | null>  
 // Retrieves asset by Tenable ID  
   
 async getVulnerabilitiesByAsset(assetId: number): Promise<TenableVulnerability[]>  
 // Gets all vulnerabilities for asset  
   
 async getRecentAssetUpdates(since: Date): Promise<TenableAsset[]>  
 // Gets assets updated since date  
   
 async getRecentVulnerabilityUpdates(since: Date): Promise<TenableVulnerability[]>  
 // Gets vulnerabilities updated since date  
   
 // Statistics and reporting  
 async getAssetStatistics(): Promise<AssetStats>  
 // Returns asset count and distribution statistics  
   
 async getVulnerabilityStatistics(): Promise<VulnStats>  
 // Returns vulnerability count and severity distribution  
   
 async getSyncStatistics(): Promise<SyncStats>  
 // Returns synchronization performance metrics  
   
 // Data cleanup  
 async cleanupOrphanedData(): Promise<CleanupResult>  
 // Removes data for assets no longer in Tenable  
   
 async archiveOldData(retentionDays: number): Promise<ArchiveResult>  
 // Archives old vulnerability and scan data  
}

#### TenableSchedulerService

**Location**: server/services/tenableSchedulerService.ts

class TenableSchedulerService {  
 // Scheduler management  
 async startScheduler(): Promise<void>  
 // Starts the sync scheduler  
   
 async stopScheduler(): Promise<void>  
 // Stops the sync scheduler  
   
 async scheduleFullSync(): Promise<void>  
 // Schedules full synchronization job  
   
 async scheduleDeltaSync(): Promise<void>  
 // Schedules incremental synchronization job  
   
 // Job execution  
 async executeScheduledSync(jobType: SyncJobType): Promise<void>  
 // Executes scheduled sync job  
   
 async retryFailedJobs(): Promise<void>  
 // Retries failed sync jobs  
   
 // Monitoring  
 async getSchedulerStatus(): Promise<SchedulerStatus>  
 // Returns current scheduler status  
   
 async getUpcomingJobs(): Promise<ScheduledJob[]>  
 // Returns list of upcoming sync jobs  
   
 async getJobHistory(limit?: number): Promise<JobHistoryEntry[]>  
 // Returns execution history  
}

### 4. Controller Layer Implementation

#### TenableController

**Location**: server/controllers/tenableController.ts

class TenableController {  
 // Configuration endpoints  
 async getConfig(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/config - Get Tenable configuration  
   
 async updateConfig(req: Request, res: Response): Promise<void>  
 // PUT /api/tenable/config - Update API credentials and settings  
   
 async testConnection(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/test - Test API connectivity  
   
 // Asset endpoints  
 async getAssets(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/assets - Get paginated asset list with filters  
   
 async getAssetDetails(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/assets/:id - Get detailed asset information  
   
 async refreshAssets(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/assets/refresh - Trigger asset sync  
   
 // Vulnerability endpoints  
 async getVulnerabilities(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/vulnerabilities - Get paginated vulnerability list  
   
 async getAssetVulnerabilities(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/assets/:id/vulnerabilities - Get vulnerabilities for asset  
   
 async refreshVulnerabilities(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/vulnerabilities/refresh - Trigger vulnerability sync  
   
 // Sync management endpoints  
 async getSyncJobs(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/sync/jobs - Get sync job history  
   
 async createSyncJob(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/sync/jobs - Create new sync job  
   
 async getSyncJobStatus(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/sync/jobs/:id - Get sync job status  
   
 async cancelSyncJob(req: Request, res: Response): Promise<void>  
 // DELETE /api/tenable/sync/jobs/:id - Cancel running sync job  
   
 // Statistics and reporting  
 async getAssetStatistics(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/stats/assets - Get asset statistics  
   
 async getVulnerabilityStatistics(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/stats/vulnerabilities - Get vulnerability statistics  
   
 async getSyncStatistics(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/stats/sync - Get synchronization performance metrics  
   
 // Export endpoints  
 async exportAssetData(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/export/assets - Export asset data to CSV/JSON  
   
 async exportVulnerabilityData(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/export/vulnerabilities - Export vulnerability data  
}

#### TenableWebhookController

**Location**: server/controllers/tenableWebhookController.ts

class TenableWebhookController {  
 // Webhook management  
 async createWebhook(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/webhooks - Create new webhook subscription  
   
 async getWebhooks(req: Request, res: Response): Promise<void>  
 // GET /api/tenable/webhooks - List webhook subscriptions  
   
 async deleteWebhook(req: Request, res: Response): Promise<void>  
 // DELETE /api/tenable/webhooks/:id - Delete webhook subscription  
   
 // Webhook handlers  
 async handleAssetUpdate(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/webhooks/assets - Handle asset update notifications  
   
 async handleVulnerabilityUpdate(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/webhooks/vulnerabilities - Handle vulnerability updates  
   
 async handleScanComplete(req: Request, res: Response): Promise<void>  
 // POST /api/tenable/webhooks/scans - Handle scan completion notifications  
}

### 5. Frontend Implementation

#### Page 1: Tenable Integration Settings

**Route**: /settings/integrations/tenable **Purpose**: Configure Tenable API connection and sync settings

**UI Components**: - **Connection Configuration Panel** - Input fields for Access Key and Secret Key (masked) - Base URL field (defaulted to cloud.tenable.com) - “Test Connection” button with status indicator - Connection status badge (Connected/Disconnected/Error)

* **Sync Configuration Panel**
  + Toggle for enabling/disabling automatic sync
  + Dropdown for sync frequency (Every 2 hours, 4 hours, 8 hours, Daily)
  + Toggle for full sync schedule (Weekly/Monthly)
  + Chunk size sliders for assets and vulnerabilities
  + Max concurrent exports setting
* **Advanced Settings Panel**
  + Rate limiting configuration
  + Retry settings
  + Data retention policies
  + Webhook configuration

**Captured Data**: - API credentials (encrypted before storage) - Sync preferences and schedules - Performance tuning parameters - Notification settings

#### Page 2: Tenable Assets Dashboard

**Route**: /assets/tenable **Purpose**: View and manage Tenable asset inventory

**UI Components**: - **Header with Statistics Cards** - Total Assets count - Assets with Agents count - Cloud Assets count (AWS/Azure/GCP breakdown) - Last Sync timestamp

* **Filters and Search Panel**
  + Text search for hostname/IP
  + Dropdown filters for OS, Cloud Provider, Agent Status
  + Date range picker for last seen
  + Criticality level filter
  + Export buttons (CSV/JSON)
* **Asset Data Grid**
  + Columns: Hostname, IP Address, OS, Cloud Provider, Agent Status, Exposure Score, Last Seen
  + Sortable columns with pagination
  + Row actions: View Details, View Vulnerabilities, Sync Individual Asset
  + Bulk actions: Sync Selected, Export Selected
* **Asset Details Modal**
  + Comprehensive asset information
  + Network interfaces and IP addresses
  + Cloud metadata (instance IDs, regions, etc.)
  + Tags and custom attributes
  + Asset timeline and scan history

**Captured Data**: - Asset inventory from Tenable - User filter preferences - Asset selection for bulk operations - Export parameters

#### Page 3: Tenable Vulnerabilities Dashboard

**Route**: /vulnerabilities/tenable **Purpose**: View and analyze vulnerability data from Tenable

**UI Components**: - **Vulnerability Metrics Panel** - Donut chart showing severity distribution - Trend line for new vulnerabilities over time - MTTR (Mean Time to Remediation) metric - Critical/High vulnerability count

* **Advanced Filtering Panel**
  + Severity level checkboxes
  + CVSS score range slider
  + State filter (Open/Fixed/Reopened)
  + Plugin family dropdown
  + Date range for first/last found
  + Asset-specific filtering
* **Vulnerability List View**
  + Columns: Plugin Name, Severity, CVSS Score, Affected Assets, First Found, Last Found, State
  + Expandable rows showing vulnerability details
  + Inline remediation guidance
  + Links to asset details
* **Vulnerability Details Panel**
  + Complete vulnerability description
  + CVSS metrics and scoring
  + Affected asset list
  + Remediation steps and references
  + Historical tracking chart

**Captured Data**: - Vulnerability assessment data - User analysis filters - Remediation tracking information - Risk prioritization preferences

#### Page 4: Sync Management Dashboard

**Route**: /admin/tenable/sync **Purpose**: Monitor and manage Tenable synchronization jobs

**UI Components**: - **Sync Status Overview** - Current sync job status cards - Progress bars for active syncs - Next scheduled sync countdown - Sync health indicators

* **Job Management Panel**
  + “Start Full Sync” button
  + “Start Delta Sync” button
  + “Schedule Custom Sync” button
  + Job queue visualization
* **Sync History Table**
  + Columns: Job Type, Status, Started, Duration, Records Processed, Errors
  + Filter by job type and status
  + Detailed job logs modal
  + Retry failed jobs action
* **Performance Metrics**
  + Sync duration trends chart
  + Data volume processed over time
  + Error rate tracking
  + API rate limit utilization
* **Configuration Quick Settings**
  + Sync frequency adjustment
  + Enable/disable scheduled syncs
  + Emergency stop all syncs button

**Captured Data**: - Sync job configurations - Performance monitoring data - Error logs and diagnostics - Administrative preferences

#### Page 5: Tenable Analytics & Reporting

**Route**: /analytics/tenable **Purpose**: Business intelligence and trend analysis

**UI Components**: - **Executive Dashboard** - Risk posture overview - Asset growth trends - Vulnerability remediation metrics - Compliance status indicators

* **Interactive Charts**
  + Asset discovery timeline
  + Vulnerability aging analysis
  + Risk score distribution
  + Cloud asset allocation
* **Custom Report Builder**
  + Drag-and-drop report designer
  + Predefined report templates
  + Scheduled report generation
  + Export options (PDF, Excel, PowerPoint)
* **Comparative Analysis**
  + Before/after Tenable integration metrics
  + Benchmark comparisons
  + ROI calculations
  + Time-to-detection improvements

**Captured Data**: - Report configurations and templates - Analytics preferences - Scheduled report settings - Business metrics and KPIs

### 6. Integration Workflow

#### Initial Setup Process

1. **Configuration**: Admin configures API credentials in settings page
2. **Connection Test**: System validates credentials and permissions
3. **Initial Sync**: Full synchronization of assets and vulnerabilities
4. **Schedule Setup**: Automated sync jobs are configured and started
5. **Validation**: Data verification and quality checks

#### Ongoing Operations

1. **Scheduled Syncs**: Regular delta synchronization every 4 hours
2. **Real-time Updates**: Webhook notifications for immediate updates
3. **Monitoring**: Continuous sync job and data quality monitoring
4. **Reporting**: Automated generation of security and compliance reports
5. **Maintenance**: Periodic cleanup and data archival

This technical specification provides a complete blueprint for implementing Tenable API integration into the RAS-DASH platform, ensuring comprehensive asset and vulnerability data synchronization with proper monitoring and management capabilities.