User Guide

Vulnerability Database API

Table of Contents

---

1. Introduction

1.1 What is Vulnetix Vulnerability Database (VVD)?

The Vulnetix Vulnerability Database (VVD) is a comprehensive vulnerability intelligence platform that aggregates, enriches, and provides unified access to CVE metadata and vulnerability data from multiple authoritative sources. VVD enriches this data with AI-generated analyses, affected function identification, and cross-source correlation to provide comprehensive vulnerability intelligence.

VVD operates as a **GCVE (Global CVE) Numbering Authority**, contributing to the global vulnerability identification ecosystem.

1.2 Data Redistribution License

All data provided through the VVD API is licensed under the **MIT License**, allowing you to:

- Use the data commercially
- Modify and distribute the data
- Include the data in proprietary software
- Redistribute the data freely

The only requirement is to include the MIT license notice and copyright statement in all copies or substantial portions of the data.

1.3 Data Source Credits

VVD aggregates vulnerability data from the following authoritative sources:

Primary Sources:

- **MITRE (cve.org)** - CVE List and official CVE records in CVEListV5 format
- **NIST NVD** - National Vulnerability Database with enriched CVSS metrics and CPE data
- **CISA KEV** - Cybersecurity and Infrastructure Security Agency Known Exploited Vulnerabilities Catalog

Enhanced Intelligence Sources:

- **VulnCheck KEV** - Extended Known Exploited Vulnerabilities database
- **VulnCheck NVD++** - Enhanced NVD data with additional metadata
- **VulnCheck XDB** - Exploit database with proof-of-concept code
- **CrowdSec** - Real-world attack sightings and IP intelligence

Ecosystem-Specific Sources:

- **GitHub Security Advisories (GHSA)** - Security advisories for open source projects
- **OSV (Open Source Vulnerabilities)** - Distributed vulnerability database for open source
- **EUVD** - European Union Vulnerability Database

Risk Scoring:

- **FIRST EPSS** - Exploit Prediction Scoring System for probability of exploitation
- **Coalition CESS** - Cybersecurity Exploit Scoring System

---

2. Getting Started

2.1 Obtaining API Credentials

VVD uses an **Organization-based credential model** where your organization's UUID and secret key serve as your API credentials.

How to Request Credentials:

1. **Via Demo Request Form:**
    - Visit: https://www.vulnetix.com
    - Complete the demo request form
    - Credentials will be issued to your organization email

2. **Via Email:**
    - Email: sales@vulnetix.com
    - Subject: "VDB API Access Request"
    - Include: Company name, use case, and contact information

2.2 Understanding Your Credentials

Your credentials consist of two components:

Organization UUID (Access Key)

- Format: Standard UUID (e.g.,)
- Purpose: Uniquely identifies your organization
- Usage: Used as the AWS SigV4 access key for authentication

Organization Secret (Secret Key)

- Format: 64-character alphanumeric string (e.g.,)
- Purpose: Secret key for signing authentication requests
- Security: Never share this key or commit it to version control

Credential Storage Best Practices:

```
export VVD_ORG=""
export VVD_SECRET=""
export VVD_ACCESS_KEY="${VVD_ORG}"
```

2.3 API Base URLs

Production API:

https://api.vdb.vulnetix.com/v1

All API endpoints are prefixed with `/v1` to support future versioning.

---

3. Authentication

3.1 Authentication Architecture

The VDB API uses a client credential authentication system, with time-limited authorization tokens:

1. **Stage 1:** AWS Signature Version 4 (SigV4) with SHA-512 to exchange credentials for a JWT token
2. **Stage 2:** JWT Bearer token authentication for all subsequent API requests

This dual-stage approach provides:

- Industry-standard request signing (AWS SigV4)
- Leverages AWS maintained libraries in many programming languages
- Stateless, scalable authentication (JWT) for API requests
- Short-lived tokens (15-minute expiry) for enhanced security

3.2 AWS Signature Version 4 (SigV4)

What is AWS SigV4?

AWS Signature Version 4 is a cryptographic signing protocol that ensures:

- **Request Integrity:** Requests cannot be modified in transit
- **Authentication:** Proves you possess the secret key without transmitting it
- **Replay Protection:** Timestamps prevent request replay attacks

Why We Chose SigV4

1. **Industry Standard:** Proven security model used by AWS and other major platforms
2. **No Password Transmission:** Secret key never leaves your system
3. **Request Tampering Protection:** Any modification invalidates the signature
4. **Time-Based Security:** Authorization expires after 15 minutes
5. **Broad Library Support:** Client libraries available for all major languages

SigV4 Signature Components

The VDB API uses a SHA-512 variant of SigV4 (instead of the standard SHA-256) with these parameters:

- **Algorithm:** `AWS4-HMAC-SHA512`
- **Region:** `us-east-1`
- **Service:** `vdb`
- **Signed Headers:** `x-amz-date` only (browser-compatible)

Why only `x-amz-date` is signed:

Browsers block JavaScript from setting the `host` header, so we sign only `x-amz-date` to ensure consistent behavior across all clients (browser, server, mobile).

Using the Request Signing Script

A Node.js script is provided for generating SigV4 signatures:

#!/usr/bin/env node

```
const crypto = require('crypto');
const { URL } = require('url');

const AWS_ACCESS_KEY_ID = process.env.AWS_ACCESS_KEY_ID ||
'YOUR_ACCESS_KEY';
const AWS_SECRET_ACCESS_KEY = process.env.AWS_SECRET_ACCESS_KEY ||
'YOUR_SECRET_KEY';
const AWS_REGION = process.env.AWS_REGION || 'us-east-1';
const AWS_SERVICE = process.env.AWS_SERVICE || 'vdb';

const method = (process.argv[2] || 'GET').toUpperCase();
const urlString = process.argv[3] || 'http://localhost:8778/v1/auth/token';
const body = process.argv[4] || '';

function sign(key, msg) {
    return crypto.createHmac('sha512', key).update(msg).digest();
}

function getSignatureKey(key, dateStamp, regionName, serviceName) {
    const kDate = sign('AWS4' + key, dateStamp);
    const kRegion = sign(kDate, regionName);
    const kService = sign(kRegion, serviceName);
    const kSigning = sign(kService, 'aws4_request');
    return kSigning;
}

function hash(data) {
    return crypto.createHash('sha512').update(data).digest('hex');
}

const url = new URL(urlString);
const amzDate = new Date().toISOString().replace(/[:-]|\.\d{3}/g, '');
const dateStamp = amzDate.substring(0, 8);

const canonicalUri = url.pathname;
const canonicalQuerystring = url.search.substring(1);
const canonicalHeaders = `x-amz-date:${amzDate}\n`;
const signedHeaders = 'x-amz-date';
const payloadHash = hash(body);

const canonicalRequest =
`${method}\n${canonicalUri}\n${canonicalQuerystring}\n${canonicalHeaders}\n${signedHead
ers}\n${payloadHash}`;

const algorithm = 'AWS4-HMAC-SHA512';
const credentialScope =
`${dateStamp}/${AWS_REGION}/${AWS_SERVICE}/aws4_request`;
```

```
const stringToSign =
`${algorithm}\n${amzDate}\n${credentialScope}\n${hash(canonicalRequest)}`;

const signingKey = getSignatureKey(AWS_SECRET_ACCESS_KEY, dateStamp,
AWS_REGION, AWS_SERVICE);
const signature = crypto.createHmac('sha512',
signingKey).update(stringToSign).digest('hex');

const authorizationHeader = `${algorithm}
Credential=${AWS_ACCESS_KEY_ID}/${credentialScope},
SignedHeaders=${signedHeaders}, Signature=${signature}`;

console.log(JSON.stringify({
    'X-Amz-Date': amzDate,
    'Authorization': authorizationHeader
}, null, 2));
```

Usage:

```
export AWS_ACCESS_KEY_ID=""
export AWS_SECRET_ACCESS_KEY=""

node sign-aws-request.js GET https://api.vdb.vulnetix.com/v1/auth/token
```

Output:

```
{
  "X-Amz-Date": "20250116T123456Z",
  "Authorization": "AWS4-HMAC-SHA512
Credential=f7c11fc1-.../20250116/us-east-1/vdb/aws4_request, SignedHeaders=x-amz-date,
Signature=abc123..."
}
```

3.3 JWT Token Authentication

What is JWT?

JSON Web Token (JWT) is an industry-standard (RFC 7519) method for securely transmitting information between parties as a JSON object. JWTs are:

- **Self-contained:** All necessary information is in the token
- **Stateless:** No server-side session storage required
- **Signed:** Cryptographically signed to prevent tampering
- **Compact:** URL-safe and efficient for HTTP headers

Why We Chose JWT

1. **Stateless Authentication:** No database lookups on every request
2. **Scalability:** Works seamlessly across distributed systems
3. **Standard Format:** Widely supported by libraries and tools
4. **Security:** HMAC-SHA512 signing prevents token forgery
5. **Expiry:** Built-in expiration mechanism (15 minutes)

JWT Token Structure

A VDB JWT token consists of three parts separated by dots:

eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJ1cm46dnVsbmV0aXg6dmRiIiwic3ViIjoidXJuOnV1aWQ6ZjdjMTFmYzEtZDQyMi00MjQyLWEwNWUtZWEzZTc0N2IwN2JjIiwib3JnSWQiOiJmN2MxMWZjMS1kNDIyLTQyNDItYTA1ZS1lYTNlNzQ3YjA3YmMiLCJpYXQiOjE3MDQxMDk1MDAsImV4cCI6MTcwNDExMDQwMH0.signature

Header (Part 1):

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

Payload (Part 2):

```
{
  "iss": "urn:vulnetix:vdb",
  "sub": "urn:uuid:",
  "orgId": "",
  "iat": 1704109500,
  "exp": 1704110400
}
```

Signature (Part 3): HMAC-SHA512 hash of header + payload

JWT Claims Reference

Claim	Description	Example
`iss`	Issuer - always "urn:vulnetix:vdb"	`"urn:vulnetix:vdb"`
`sub`	Subject - your Organization UUID in URN format	`"urn:uuid:f7c11fc1-..."`
`orgId`	Your Organization UUID	`"f7c11fc1-d422-4242-..."`

`iat`	Issued At - Unix timestamp when token was created	`1704109500`
`exp`	Expiration - Unix timestamp when token expires (iat + 900 seconds)	`1704110400`

Validating JWT Claims

To validate a JWT token in your application:

1. **Verify Signature:** Ensure the token hasn't been tampered with
2. **Check Expiration:** Ensure `exp` is greater than current time
3. **Validate Issuer:** Ensure `iss` is "urn:vulnetix:vdb"
4. **Confirm Subject:** Ensure `sub` matches your organization UUID

Example Validation (Python):

```
import jwt
import time

def validate_jwt(token, organization_uuid):
    try:
        decoded = jwt.decode(
            token,
            options={"verify_signature": False}
        )

        current_time = int(time.time())

        if decoded.get('exp', 0) < current_time:
            raise ValueError("Token expired")

        if decoded.get('iss') != 'urn:vulnetix:vdb':
            raise ValueError("Invalid issuer")

        expected_sub = f"urn:uuid:{organization_uuid}"
        if decoded.get('sub') != expected_sub:
            raise ValueError("Invalid subject")

        return True

    except jwt.InvalidTokenError as e:
        raise ValueError(f"Invalid token: {e}")
```

Requesting a New JWT Token

To obtain a JWT token, make a SigV4-signed GET request to `/v1/auth/token`:

Request:

```
GET /v1/auth/token HTTP/1.1
Host: api.vdb.vulnetix.com
X-Amz-Date: 20250116T123456Z
Authorization: AWS4-HMAC-SHA512
Credential=f7c11fc1-.../20250116/us-east-1/vdb/aws4_request, SignedHeaders=x-amz-date,
Signature=abc123...
```

Response:

```
{
  "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9...",
  "iss": "urn:vulnetix:vdb",
  "sub": "urn:uuid:",
  "exp": 1704110400
}
```

Using JWT for API Requests

Once you have a JWT token, include it in the `Authorization` header as a Bearer token for all API requests:

Request Example:

```
GET /v1/info/CVE-2024-1234 HTTP/1.1
Host: api.vdb.vulnetix.com
Authorization: Bearer eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9...
Content-Type: application/json
```

Token Lifecycle Management

JWT tokens expire after **15 minutes**. Implement automatic token refresh:

Best Practice Pattern:

```
from datetime import datetime, timedelta

class VDBClient:
    def __init__(self, access_key, secret_key):
        self.access_key = access_key
        self.secret_key = secret_key
        self.token = None
        self.token_expiry = None

    def get_token(self):
```

```
        if self.token and self.token_expiry > datetime.now():
            return self.token

        response = self._request_token()
        self.token = response['token']
        self.token_expiry = datetime.fromtimestamp(response['exp'])
        return self.token

    def api_request(self, endpoint):
        token = self.get_token()
        return requests.get(
            f"https://api.vdb.vulnetix.com{endpoint}",
            headers={'Authorization': f'Bearer {token}'}
        )
```

---

4. API Tools and Resources

4.1 OpenAPI Specification (OAS)

The VDB API is fully documented using OpenAPI Specification 3.1.0.

Location:

- https://api.vdb.vulnetix.com/v1/spec save it as `vdb-v1-oas.json`
- Available upon request to support@vulnetix.com

Viewing the OAS:

Option 1: Swagger UI Online

Visit https://redocly.github.io/redoc/?url=https://api.vdb.vulnetix.com/v1/spec to get an interactive API documentation interface.

Option 2: VSCode Extension

Install the "OpenAPI (Swagger) Editor" extension in VSCode, then open `vdb-v1-oas.json` for inline documentation and validation.

4.2 Using Postman in VSCode

Postman provides a powerful HTTP client that integrates with VSCode for API testing.

Step 1: Install Postman Extension

1. Open VSCode
2. Go to Extensions (Ctrl+Shift+X)

3. Search for "Postman"
4. Install the official Postman extension

Step 2: Import Collection

1. Open the Postman panel in VSCode
2. Click "Import"
3. Select `postman-collection.json` (Available upon request to support@vulnetix.com)
4. The collection will appear in your workspace

Step 3: Configure AWS Credentials

The Postman collection includes AWS SigV4 authentication pre-configured for the `/auth/token` endpoint.

1. Click on the collection name
2. Go to **Variables** tab
3. Set these values:

Variable	Value	Type
`awsAccessKeyId`	Your Organization UUID	default
`awsSecretAccessKey`	Your Organization Secret	secret
`awsRegion`	`us-east-1`	default
`awsService`	`vdb`	default
`baseUrl`	`https://api.vdb.vulnetix.com/v1`	default

4. Click **Save**

Step 4: Test Authentication

1. Expand the collection and select **GET /auth/token**
2. Click **Send**
3. Postman automatically:
    - Generates `X-Amz-Date` header
    - Calculates AWS SigV4 signature
    - Adds `Authorization` header
    - Sends the signed request

4. You should receive a JWT token in the response

Step 5: Save JWT for Subsequent Requests

Add this to the **Tests** tab of the `/auth/token` request:

```
pm.collectionVariables.set("bearerToken", pm.response.json().token);
```

Now all subsequent requests will automatically use the JWT token.

Pre-Request Script (Automatic in Collection)

The collection includes a pre-request script that automatically generates AWS SigV4 signatures:

```
const CryptoJS = require('crypto-js');

const accessKey = pm.collectionVariables.get('awsAccessKeyId') ||
pm.environment.get('awsAccessKeyId');
const secretKey = pm.collectionVariables.get('awsSecretAccessKey') ||
pm.environment.get('awsSecretAccessKey');
const region = pm.collectionVariables.get('awsRegion') || 'us-east-1';
const service = pm.collectionVariables.get('awsService') || 'vdb';

if (!accessKey || !secretKey) {
    console.error('AWS credentials not configured. Set awsAccessKeyId and
awsSecretAccessKey in collection variables.');
    return;
}

const now = new Date();
const amzDate = now.toISOString().replace(/[:-]|\.\d{3}/g, '');
const dateStamp = amzDate.substring(0, 8);

pm.request.headers.add({
    key: 'X-Amz-Date',
    value: amzDate
});

const method = pm.request.method;
const url = pm.request.url;
const path = url.getPath();
const query = url.getQueryString() || '';
const body = pm.request.body ? pm.request.body.raw || '' : '';

const payloadHash = CryptoJS.SHA512(body).toString(CryptoJS.enc.Hex);
```

```
const canonicalHeaders = `x-amz-date:${amzDate}\n`;
const signedHeaders = 'x-amz-date';
const canonicalRequest = [
    method,
    path,
    query,
    canonicalHeaders,
    signedHeaders,
    payloadHash
].join('\n');

const algorithm = 'AWS4-HMAC-SHA512';
const credentialScope = `${dateStamp}/${region}/${service}/aws4_request`;
const canonicalRequestHash =
CryptoJS.SHA512(canonicalRequest).toString(CryptoJS.enc.Hex);
const stringToSign = [
    algorithm,
    amzDate,
    credentialScope,
    canonicalRequestHash
].join('\n');

const kDate = CryptoJS.HmacSHA512(dateStamp, 'AWS4' + secretKey);
const kRegion = CryptoJS.HmacSHA512(region, kDate);
const kService = CryptoJS.HmacSHA512(service, kRegion);
const kSigning = CryptoJS.HmacSHA512('aws4_request', kService);
const signature = CryptoJS.HmacSHA512(stringToSign,
kSigning).toString(CryptoJS.enc.Hex);

const authHeader = `${algorithm} Credential=${accessKey}/${credentialScope},
SignedHeaders=${signedHeaders}, Signature=${signature}`;

pm.request.headers.add({
    key: 'Authorization',
    value: authHeader
});

console.log('AWS SigV4 (SHA-512) authentication applied');
console.log('X-Amz-Date:', amzDate);
console.log('Authorization:', authHeader.substring(0, 80) + '...');
```

---

5. API Concepts

5.1 Rate Limiting

The VDB API implements **per-minute rate limiting** to ensure fair usage and system stability.

Rate Limit Headers

Every API response includes rate limit information in the headers:

Header	Description	Example
`RateLimit-MinuteLimit`	Maximum requests allowed per minute	`60`
`RateLimit-Remaining`	Requests remaining in current minute	`45`
`RateLimit-Reset`	Seconds until limit resets	`28`

Example Response Headers:

HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-MinuteLimit: 60
RateLimit-Remaining: 45
RateLimit-Reset: 28

Rate Limit Exceeded Response

When you exceed the rate limit, you'll receive a 429 status code:

```
{
  "success": false,
  "error": "Rate limit exceeded",
  "details": "Too many requests. Limit: 60 requests per minute. Try again in 28 seconds."
}
```

Handling Rate Limits

Approach 1: Monitor Headers

```
import time
import requests

def check_rate_limits(response):
    remaining = response.headers.get('RateLimit-Remaining')
    reset = response.headers.get('RateLimit-Reset')

    if remaining != 'unlimited':
        remaining_count = int(remaining)
```

```
        if remaining_count < 10:
            print(f"Warning: Only {remaining_count} requests remaining")
            print(f"Resets in {reset} seconds")

        if remaining_count < 5:
            time.sleep(2)
```

Approach 2: Exponential Backoff

```
async function apiRequest(url, options = {}) {
  const response = await fetch(url, {
    ...options,
    headers: {
      'Authorization': `Bearer ${token}`,
      ...options.headers
    }
  });

  const remaining = response.headers.get('RateLimit-Remaining');
  const reset = response.headers.get('RateLimit-Reset');

  if (response.status === 429) {
    const retryAfter = parseInt(reset) || 60;
    console.log(`Rate limited. Retrying in ${retryAfter} seconds...`);

    await new Promise(resolve => setTimeout(resolve, retryAfter * 1000));
    return apiRequest(url, options);
  }

  if (remaining !== 'unlimited' && parseInt(remaining) < 10) {
    console.warn(`Rate limit warning: ${remaining} requests remaining`);
  }

  return response;
}
```

5.2 Quota Management (Weekly Limits)

In addition to per-minute rate limits, the API enforces **weekly quotas** that reset every Sunday at 00:00 UTC.

Quota Headers

Header	Description	Example

`RateLimit-WeekLimit`	Maximum requests allowed per week	`10000`
`RateLimit-WeekRemaining`	Requests remaining this week	`8543`
`RateLimit-WeekReset`	Seconds until weekly quota resets	`172800`

Example Response Headers:

HTTP/1.1 200 OK
Content-Type: application/json
RateLimit-MinuteLimit: 60
RateLimit-WeekLimit: 10000
RateLimit-WeekRemaining: 8543
RateLimit-WeekReset: 172800

Weekly Quota Exceeded Response

```
{
  "success": false,
  "error": "Weekly rate limit exceeded",
  "details": "Weekly quota exceeded. Limit: 10000 requests per week. Resets in 48 hours."
}
```

Quota Configuration

Quota limits are configurable per organization:

- **Default:** 1000 requests per week
- **Upgrade Available:** Contact sales@vulnetix.com for higher quotas
- **Unlimited:** Set limit to `0` for unlimited access

5.3 Pagination

Several API endpoints support pagination for large result sets:

- `/v1/product/:name`
- `/v1/product/:name/:version`
- `/v1/:package/versions`
- `/v1/:package/vulns`

Pagination Parameters

Parameter	Type	Default	Max	Description
`limit`	integer	100	500	Results per page

`offset`	integer	0	-	Number of results to skip

Pagination Response

```
{
  "packageName": "express",
  "timestamp": 1700000000,
  "total": 523,
  "limit": 100,
  "offset": 0,
  "hasMore": true,
  "versions": [...]
}
```

Field	Description
`total`	Total number of results available
`limit`	Current page size
`offset`	Current offset
`hasMore`	Whether more results are available

Pagination Example

Request Page 1:

GET /v1/product/express?limit=100&offset=0

Request Page 2:

GET /v1/product/express?limit=100&offset=100

Request Page 3:

GET /v1/product/express?limit=100&offset=200

Iterating Through All Pages:

```
def get_all_results(package_name, jwt_token):
    base_url = "https://api.vdb.vulnetix.com/v1/product"
    headers = {"Authorization": f"Bearer {jwt_token}"}

    all_versions = []
```

```
    offset = 0
    limit = 100
    has_more = True

    while has_more:
        url = f"{base_url}/{package_name}?limit={limit}&offset={offset}"
        response = requests.get(url, headers=headers)
        data = response.json()

        all_versions.extend(data['versions'])

        has_more = data['hasMore']
        offset += limit

    return all_versions
```

---

6. Language-Specific Examples

6.1 Bash

```
#!/bin/bash

export VVD_ORG=""
export VVD_SECRET=""
export VVD_ACCESS_KEY="${VVD_ORG}"

AMZ_DATE=$(date -u +"%Y%m%dT%H%M%SZ")
DATE_STAMP=$(date -u +"%Y%m%d")
REGION="us-east-1"
SERVICE="vdb"

METHOD="GET"
CANONICAL_URI="/v1/auth/token"
CANONICAL_QUERYSTRING=""
CANONICAL_HEADERS="x-amz-date:${AMZ_DATE}\n"
SIGNED_HEADERS="x-amz-date"

PAYLOAD_HASH=$(echo -n "" | openssl dgst -sha512 -hex | sed "s/^.* //")

CANONICAL_REQUEST="${METHOD}\n${CANONICAL_URI}\n${CANONICAL_QUERYST
RING}\n${CANONICAL_HEADERS}\n${SIGNED_HEADERS}\n${PAYLOAD_HASH}"

ALGORITHM="AWS4-HMAC-SHA512"
CREDENTIAL_SCOPE="${DATE_STAMP}/${REGION}/${SERVICE}/aws4_request"
```

```
REQUEST_HASH=$(echo -n "${CANONICAL_REQUEST}" | openssl dgst -sha512 -hex |
sed "s/^.* //")
STRING_TO_SIGN="${ALGORITHM}\n${AMZ_DATE}\n${CREDENTIAL_SCOPE}\n${REQ
UEST_HASH}"

kDate=$(echo -n "${DATE_STAMP}" | openssl dgst -sha512 -hmac
"AWS4${VVD_SECRET}" -binary)
kRegion=$(echo -n "${REGION}" | openssl dgst -sha512 -hmac "${kDate}" -binary)
kService=$(echo -n "${SERVICE}" | openssl dgst -sha512 -hmac "${kRegion}" -binary)
kSigning=$(echo -n "aws4_request" | openssl dgst -sha512 -hmac "${kService}" -binary)
SIGNATURE=$(echo -n "${STRING_TO_SIGN}" | openssl dgst -sha512 -hmac "${kSigning}"
-hex | sed "s/^.* //")

AUTHORIZATION="${ALGORITHM}
Credential=${VVD_ACCESS_KEY}/${CREDENTIAL_SCOPE},
SignedHeaders=${SIGNED_HEADERS}, Signature=${SIGNATURE}"

echo "Requesting JWT token from /v1/auth/token"
JWT_RESPONSE=$(curl -s -X GET "https://api.vdb.vulnetix.com/v1/auth/token" \
  -H "Authorization: ${AUTHORIZATION}" \
  -H "X-Amz-Date: ${AMZ_DATE}")

export VVD_JWT=$(echo "${JWT_RESPONSE}" | jq -r '.token')
echo "JWT token obtained (expires in 15 minutes): ${VVD_JWT:0:50}..."

echo "Making GET request to /ecosystems..."
curl -X GET "https://api.vdb.vulnetix.com/v1/ecosystems" \
  -H "Authorization: Bearer ${VVD_JWT}" \
  -H "Content-Type: application/json" | jq
```

6.2 JavaScript (Browser)

```
const VVD_ORG = "";
const VVD_SECRET = "";
const VVD_ACCESS_KEY = VVD_ORG;

async function hmacSha512(key, data) {
    const encoder = new TextEncoder();
    const keyData = typeof key === "string" ? encoder.encode(key) : key;

    const cryptoKey = await crypto.subtle.importKey(
        "raw", keyData,
        { name: "HMAC", hash: "SHA-512" },
        false, ["sign"]
    );

    const signature = await crypto.subtle.sign("HMAC", cryptoKey, encoder.encode(data));
```

```
    return new Uint8Array(signature);
}

function toHex(bytes) {
    return Array.from(bytes)
        .map(b => b.toString(16).padStart(2, "0"))
        .join("");
}

async function sha512(data) {
    const encoder = new TextEncoder();
    const hashBuffer = await crypto.subtle.digest("SHA-512", encoder.encode(data));
    return toHex(new Uint8Array(hashBuffer));
}

async function getSignatureKey(key, dateStamp, region, service) {
    const kDate = await hmacSha512(`AWS4${key}`, dateStamp);
    const kRegion = await hmacSha512(kDate, region);
    const kService = await hmacSha512(kRegion, service);
    const kSigning = await hmacSha512(kService, "aws4_request");
    return kSigning;
}

async function signRequest(accessKey, secretKey, method, path, headers = {}, body = "") {
    const now = new Date();
    const amzDate = now.toISOString().replace(/[:-]|\.\d{3}/g, "");
    const dateStamp = amzDate.substring(0, 8);
    const region = "us-east-1";
    const service = "vdb";

    const allHeaders = { ...headers, "x-amz-date": amzDate };

    const payloadHash = await sha512(body);
    const signedHeaders = Object.keys(allHeaders).sort().join(";");
    const canonicalHeaders = Object.keys(allHeaders).sort()
        .map(k => `${k}:${allHeaders[k].trim()}\n`)
        .join("");

    const canonicalRequest = [
        method, path, "",
        canonicalHeaders, signedHeaders, payloadHash
    ].join("\n");

    const canonicalRequestHash = await sha512(canonicalRequest);
    const credentialScope = `${dateStamp}/${region}/${service}/aws4_request`;
    const stringToSign = [
        "AWS4-HMAC-SHA512", amzDate, credentialScope, canonicalRequestHash
    ].join("\n");
```

```
    const signingKey = await getSignatureKey(secretKey, dateStamp, region, service);
    const signature = toHex(await hmacSha512(signingKey, stringToSign));

    const authHeader = `AWS4-HMAC-SHA512 Credential=${accessKey}/${credentialScope},
SignedHeaders=${signedHeaders}, Signature=${signature}`;

    return {
        ...allHeaders,
        "Authorization": authHeader
    };
}

async function getJWTToken() {
    const method = "GET";
    const path = "/auth/token";

    const signedHeaders = await signRequest(VVD_ACCESS_KEY, VVD_SECRET, method,
path);

    const response = await fetch("https://api.vdb.vulnetix.com/v1" + path, {
        method: method,
        headers: signedHeaders
    });

    const data = await response.json();

    if (!data.token) {
        throw new Error("Failed to obtain JWT token");
    }

    console.log("JWT token obtained (expires in 15 minutes):", data.token.substring(0, 50) +
"...");
    console.log("Token details:", {
        iss: data.iss,
        sub: data.sub,
        exp: new Date(data.exp * 1000).toISOString()
    });

    return data.token;
}

async function makeAPIRequest() {
    const jwtToken = await getJWTToken();

    const response = await fetch("https://api.vdb.vulnetix.com/v1/ecosystems", {
        method: "GET",
        headers: {
```

```
            "Authorization": `Bearer ${jwtToken}`,
            "Content-Type": "application/json"
        }
    });

    const data = await response.json();
    console.log("API Response:", data);
    return data;
}

makeAPIRequest().catch(console.error);
```

6.3 Python

```
#!/usr/bin/env python3

import json
import requests
from datetime import datetime
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
import botocore.credentials

VVD_ORG = ""
VVD_SECRET = ""
VVD_ACCESS_KEY = VVD_ORG

REGION = "us-east-1"
SERVICE = "vdb"

BASE_URL = "https://api.vdb.vulnetix.com/v1"


def sign_request(method, url, headers=None, body=None):
    credentials = botocore.credentials.Credentials(
        access_key=VVD_ACCESS_KEY,
        secret_key=VVD_SECRET
    )

    request = AWSRequest(method=method, url=url, headers=headers, data=body)

    SigV4Auth(credentials, SERVICE, REGION).add_auth(request)

    return dict(request.headers)


def get_jwt_token():
```

```
    print("Requesting JWT token from /v1/auth/token...")

    url = f"{BASE_URL}/auth/token"

    signed_headers = sign_request("GET", url)

    response = requests.get(url, headers=signed_headers)
    response.raise_for_status()

    data = response.json()

    token = data.get("token")
    print(f"JWT token obtained (expires in 15 minutes): {token[:50]}...")

    return token


def make_api_request():
    jwt_token = get_jwt_token()

    print("Making GET request to /ecosystems...")

    url = f"{BASE_URL}/ecosystems"

    headers = {
        "Authorization": f"Bearer {jwt_token}",
        "Content-Type": "application/json"
    }

    response = requests.get(url, headers=headers)
    response.raise_for_status()

    data = response.json()
    print(json.dumps(data, indent=2))

    return data


if __name__ == "__main__":
    try:
        make_api_request()
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")
        if hasattr(e, "response") and e.response is not None:
            print(f"Response: {e.response.text}")
```

6.4 Ruby

```ruby
require 'aws-sigv4'
require 'net/http'
require 'uri'
require 'json'
require 'openssl'

VVD_ORG = ""
VVD_SECRET = ""
VVD_ACCESS_KEY = VVD_ORG

signer = Aws::Sigv4::Signer.new(
  service: 'vdb',
  region: 'us-east-1',
  access_key_id: VVD_ACCESS_KEY,
  secret_access_key: VVD_SECRET,
  uri_escape_path: true,
  apply_checksum_header: false
)

def get_jwt_token(signer, host)
  url = URI("https://#{host}/auth/token")
  http = Net::HTTP.new(url.host, url.port)
  http.use_ssl = true

  request = Net::HTTP::Get.new(url.request_uri)

  signature = signer.sign_request(
    http_method: 'GET',
    url: url.to_s,
    headers: {},
    body: ''
  )

  signature.headers.each do |key, value|
    request[key] = value
  end

  response = http.request(request)

  if response.code.to_i == 200
    data = JSON.parse(response.body)
    puts "JWT token obtained (expires in 15 minutes): #{data['token'][0..50]}..."
    puts "Token details:"
    puts "  iss: #{data['iss']}"
    puts "  sub: #{data['sub']}"
    puts "  exp: #{Time.at(data['exp']).utc}"
```

```
    return data['token']
  else
    raise "Failed to obtain JWT token: #{response.code} #{response.body}"
  end
end

def make_api_request(jwt_token, host, path)
  url = URI("https://#{host}#{path}")
  http = Net::HTTP.new(url.host, url.port)
  http.use_ssl = true

  request = Net::HTTP::Get.new(url.request_uri)
  request['Authorization'] = "Bearer #{jwt_token}"
  request['Content-Type'] = 'application/json'

  response = http.request(request)
  data = JSON.parse(response.body)

  puts "API Response:"
  puts JSON.pretty_generate(data)

  data
end

begin
  host = "api.vdb.vulnetix.com"

  jwt_token = get_jwt_token(signer, host)

  result = make_api_request(jwt_token, host, "/ecosystems")
rescue StandardError => e
  puts "Error: #{e.message}"
  puts e.backtrace
end
```

6.5 Go

```
package main

import (
    "bytes"
    "context"
    "crypto/sha512"
    "encoding/hex"
    "encoding/json"
    "fmt"
    "io"
```

```
    "net/http"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    v4 "github.com/aws/aws-sdk-go-v2/aws/signer/v4"
)

const (
    VVD_ORG    = ""
    VVD_SECRET = ""
    VVD_ACCESS_KEY = VVD_ORG
)

type staticCredentialsProvider struct {
    accessKey string
    secretKey string
}

func (s *staticCredentialsProvider) Retrieve(ctx context.Context) (aws.Credentials, error) {
    return aws.Credentials{
        AccessKeyID:     s.accessKey,
        SecretAccessKey: s.secretKey,
    }, nil
}

func sha512Hash(data []byte) string {
    hash := sha512.Sum512(data)
    return hex.EncodeToString(hash[:])
}

func getJWTToken() (string, error) {
    ctx := context.Background()
    url := "https://api.vdb.vulnetix.com/v1/auth/token"

    req, err := http.NewRequestWithContext(ctx, "GET", url, nil)
    if err != nil {
        return "", fmt.Errorf("failed to create request: %w", err)
    }

    credsProvider := &staticCredentialsProvider{
        accessKey: VVD_ACCESS_KEY,
        secretKey: VVD_SECRET,
    }

    creds, err := credsProvider.Retrieve(ctx)
    if err != nil {
        return "", fmt.Errorf("failed to retrieve credentials: %w", err)
    }
```

```
    signer := v4.NewSigner()

    payloadHash := sha512Hash([]byte(""))

    err = signer.SignHTTP(ctx, creds, req, payloadHash, "vdb", "us-east-1", time.Now())
    if err != nil {
        return "", fmt.Errorf("failed to sign request: %w", err)
    }

    client := &http.Client{Timeout: 30 * time.Second}
    resp, err := client.Do(req)
    if err != nil {
        return "", fmt.Errorf("failed to execute request: %w", err)
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return "", fmt.Errorf("failed to read response: %w", err)
    }

    if resp.StatusCode != http.StatusOK {
        return "", fmt.Errorf("request failed with status %d: %s", resp.StatusCode, string(body))
    }

    var result struct {
        Token string `json:"token"`
        Iss   string `json:"iss"`
        Sub   string `json:"sub"`
        Exp   int64  `json:"exp"`
    }

    if err := json.Unmarshal(body, &result); err != nil {
        return "", fmt.Errorf("failed to parse response: %w", err)
    }

    fmt.Printf("JWT token obtained (expires in 15 minutes): %s...\n", result.Token[:50])
    fmt.Printf("Token details:\n")
    fmt.Printf("  iss: %s\n", result.Iss)
    fmt.Printf("  sub: %s\n", result.Sub)
    fmt.Printf("  exp: %s\n", time.Unix(result.Exp, 0).Format(time.RFC3339))

    return result.Token, nil
}

func makeAPIRequest(jwtToken string) error {
    url := "https://api.vdb.vulnetix.com/v1/ecosystems"
```

```
    req, err := http.NewRequest("GET", url, nil)
    if err != nil {
        return fmt.Errorf("failed to create request: %w", err)
    }

    req.Header.Set("Authorization", "Bearer "+jwtToken)
    req.Header.Set("Content-Type", "application/json")

    client := &http.Client{Timeout: 30 * time.Second}
    resp, err := client.Do(req)
    if err != nil {
        return fmt.Errorf("failed to execute request: %w", err)
    }
    defer resp.Body.Close()

    body, err := io.ReadAll(resp.Body)
    if err != nil {
        return fmt.Errorf("failed to read response: %w", err)
    }

    var prettyJSON bytes.Buffer
    if err := json.Indent(&prettyJSON, body, "", "  "); err != nil {
        fmt.Printf("API Response (raw): %s\n", string(body))
    } else {
        fmt.Printf("API Response:\n%s\n", prettyJSON.String())
    }

    return nil
}

func main() {
    jwtToken, err := getJWTToken()
    if err != nil {
        fmt.Printf("Error getting JWT token: %v\n", err)
        return
    }

    if err := makeAPIRequest(jwtToken); err != nil {
        fmt.Printf("Error making API request: %v\n", err)
        return
    }
}
```

6.6 Kotlin

```
import software.amazon.awssdk.auth.credentials.AwsBasicCredentials
```

```
import software.amazon.awssdk.auth.credentials.StaticCredentialsProvider
import software.amazon.awssdk.auth.signer.Aws4Signer
import software.amazon.awssdk.auth.signer.params.Aws4SignerParams
import software.amazon.awssdk.http.SdkHttpFullRequest
import software.amazon.awssdk.http.SdkHttpMethod
import software.amazon.awssdk.regions.Region
import com.google.gson.Gson
import com.google.gson.JsonObject
import java.net.URI
import java.net.http.HttpClient
import java.net.http.HttpRequest
import java.net.http.HttpResponse
import java.security.MessageDigest

const val VVD_ORG = ""
const val VVD_SECRET = ""
const val VVD_ACCESS_KEY = VVD_ORG
const val REGION = "us-east-1"
const val SERVICE = "vdb"
const val BASE_URL = "https://api.vdb.vulnetix.com/v1"

fun sha512(data: String): String {
    val digest = MessageDigest.getInstance("SHA-512")
    val hashBytes = digest.digest(data.toByteArray())
    return hashBytes.joinToString("") { "%02x".format(it) }
}

fun signRequest(
    method: SdkHttpMethod,
    uri: URI,
    path: String,
    body: String = ""
): SdkHttpFullRequest {
    val credentials = AwsBasicCredentials.create(VVD_ACCESS_KEY, VVD_SECRET)
    val credentialsProvider = StaticCredentialsProvider.create(credentials)

    val requestBuilder = SdkHttpFullRequest.builder()
        .method(method)
        .uri(uri)
        .encodedPath(path)

    if (body.isNotEmpty()) {
        requestBuilder.contentStreamProvider { body.byteInputStream() }
    }

    val request = requestBuilder.build()

    val signerParams = Aws4SignerParams.builder()
```

```
        .awsCredentials(credentials)
        .signingName(SERVICE)
        .signingRegion(Region.of(REGION))
        .build()

    val signer = Aws4Signer.create()
    return signer.sign(request, signerParams)
}

data class JWTResponse(
    val token: String,
    val iss: String,
    val sub: String,
    val exp: Long
)

suspend fun getJWTToken(): String {
    val path = "/auth/token"
    val uri = URI.create("$BASE_URL$path")

    val signedRequest = signRequest(SdkHttpMethod.GET, uri, path)

    val httpRequestBuilder = HttpRequest.newBuilder()
        .uri(uri)
        .GET()

    signedRequest.headers().forEach { (name, values) ->
        values.forEach { value ->
            httpRequestBuilder.header(name, value)
        }
    }

    val httpRequest = httpRequestBuilder.build()

    val client = HttpClient.newHttpClient()
    val response = client.send(httpRequest, HttpResponse.BodyHandlers.ofString())

    val gson = Gson()
    val jwtResponse = gson.fromJson(response.body(), JWTResponse::class.java)

    println("JWT token obtained (expires in 15 minutes): ${jwtResponse.token.take(50)}...")
    println("Token details:")
    println("  Issuer: ${jwtResponse.iss}")
    println("  Subject: ${jwtResponse.sub}")
    println("  Expires: ${java.time.Instant.ofEpochSecond(jwtResponse.exp)}")

    return jwtResponse.token
}
```

```
suspend fun makeAPIRequest() {
    val jwtToken = getJWTToken()

    val apiPath = "/ecosystems"
    val apiUri = URI.create("$BASE_URL$apiPath")

    val httpRequest = HttpRequest.newBuilder()
        .uri(apiUri)
        .GET()
        .header("Authorization", "Bearer $jwtToken")
        .header("Content-Type", "application/json")
        .build()

    val client = HttpClient.newHttpClient()
    val response = client.send(httpRequest, HttpResponse.BodyHandlers.ofString())

    val gson = Gson()
    val jsonResponse = gson.fromJson(response.body(), JsonObject::class.java)
    println("API Response:")
    println(gson.toJson(jsonResponse))
}

suspend fun main() {
    try {
        makeAPIRequest()
    } catch (e: Exception) {
        println("Error: ${e.message}")
        e.printStackTrace()
    }
}
```

6.7 Swift (iOS)

```
import Foundation
import AWSCore
import CommonCrypto

let VVD_ORG = ""
let VVD_SECRET = ""
let VVD_ACCESS_KEY = VVD_ORG

extension String {
    func sha512() -> String {
        guard let data = self.data(using: .utf8) else { return "" }
        var hash = [UInt8](repeating: 0, count: Int(CC_SHA512_DIGEST_LENGTH))
        data.withUnsafeBytes {
```

```
            _ = CC_SHA512($0.baseAddress, CC_LONG(data.count), &hash)
        }
        return hash.map { String(format: "%02x", $0) }.joined()
    }

    func hmacSHA512(key: String) -> Data {
        let keyData = key.data(using: .utf8)!
        let messageData = self.data(using: .utf8)!
        var hash = [UInt8](repeating: 0, count: Int(CC_SHA512_DIGEST_LENGTH))
        keyData.withUnsafeBytes { keyBytes in
            messageData.withUnsafeBytes { messageBytes in
                CCHmac(CCHmacAlgorithm(kCCHmacAlgSHA512),
                       keyBytes.baseAddress, keyData.count,
                       messageBytes.baseAddress, messageData.count,
                       &hash)
            }
        }
        return Data(hash)
    }

    func hmacSHA512(key: Data) -> Data {
        let messageData = self.data(using: .utf8)!
        var hash = [UInt8](repeating: 0, count: Int(CC_SHA512_DIGEST_LENGTH))
        key.withUnsafeBytes { keyBytes in
            messageData.withUnsafeBytes { messageBytes in
                CCHmac(CCHmacAlgorithm(kCCHmacAlgSHA512),
                       keyBytes.baseAddress, key.count,
                       messageBytes.baseAddress, messageData.count,
                       &hash)
            }
        }
        return Data(hash)
    }
}

extension Data {
    func hexEncodedString() -> String {
        return map { String(format: "%02x", $0) }.joined()
    }
}

class AWSV4Signer {
    static func signRequest(
        accessKey: String,
        secretKey: String,
        method: String,
        path: String,
        headers: [String: String] = [:],
```

```
        body: String = ""
    ) -> [String: String] {
        let region = "us-east-1"
        let service = "vdb"

        let dateFormatter = DateFormatter()
        dateFormatter.timeZone = TimeZone(abbreviation: "UTC")
        dateFormatter.dateFormat = "yyyyMMdd'T'HHmmss'Z'"
        let amzDate = dateFormatter.string(from: Date())

        dateFormatter.dateFormat = "yyyyMMdd"
        let dateStamp = dateFormatter.string(from: Date())

        var allHeaders = headers
        allHeaders["x-amz-date"] = amzDate

        let payloadHash = body.sha512()
        let signedHeaders = allHeaders.keys.sorted().joined(separator: ";")
        let canonicalHeaders = allHeaders.keys.sorted()
            .map { "\($0):\(allHeaders[$0]!.trimmingCharacters(in: .whitespaces))\n" }
            .joined()

        let canonicalRequest = [
            method,
            path,
            "",
            canonicalHeaders,
            signedHeaders,
            payloadHash
        ].joined(separator: "\n")

        let canonicalRequestHash = canonicalRequest.sha512()
        let credentialScope = "\(dateStamp)/\(region)/\(service)/aws4_request"
        let stringToSign = [
            "AWS4-HMAC-SHA512",
            amzDate,
            credentialScope,
            canonicalRequestHash
        ].joined(separator: "\n")

        let kDate = dateStamp.hmacSHA512(key: "AWS4\(secretKey)")
        let kRegion = region.hmacSHA512(key: kDate)
        let kService = service.hmacSHA512(key: kRegion)
        let kSigning = "aws4_request".hmacSHA512(key: kService)
        let signature = stringToSign.hmacSHA512(key: kSigning).hexEncodedString()

        let authHeader = "AWS4-HMAC-SHA512 Credential=\(accessKey)/\(credentialScope),
SignedHeaders=\(signedHeaders), Signature=\(signature)"
```

```
        var signedHeaders = allHeaders
        signedHeaders["Authorization"] = authHeader

        return signedHeaders
    }
}

class VDBAPIClient {
    private let accessKey: String
    private let secretKey: String
    private let baseURL = "https://api.vdb.vulnetix.com/v1"

    init(accessKey: String, secretKey: String) {
        self.accessKey = accessKey
        self.secretKey = secretKey
    }

    func getJWTToken(completion: @escaping (Result<String, Error>) -> Void) {
        let method = "GET"
        let path = "/auth/token"

        let signedHeaders = AWSV4Signer.signRequest(
            accessKey: accessKey,
            secretKey: secretKey,
            method: method,
            path: path
        )

        guard let url = URL(string: "\(baseURL)\(path)") else {
            completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                userInfo: [NSLocalizedDescriptionKey: "Invalid URL"])))
            return
        }

        var request = URLRequest(url: url)
        request.httpMethod = method

        for (key, value) in signedHeaders {
            request.setValue(value, forHTTPHeaderField: key)
        }

        let task = URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }
```

```
            guard let data = data else {
                completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                    userInfo: [NSLocalizedDescriptionKey: "No data received"])))
                return
            }

            do {
                if let json = try JSONSerialization.jsonObject(with: data) as? [String: Any],
                   let token = json["token"] as? String {
                    print("JWT token obtained (expires in 15 minutes): \(token.prefix(50))...")
                    if let exp = json["exp"] as? Int {
                        let expiryDate = Date(timeIntervalSince1970: TimeInterval(exp))
                        print("Token expires at: \(expiryDate)")
                    }
                    completion(.success(token))
                } else {
                    completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                        userInfo: [NSLocalizedDescriptionKey: "Failed to obtain JWT token"])))
                }
            } catch {
                completion(.failure(error))
            }
        }
        task.resume()
    }

    func makeAPIRequest(jwtToken: String, completion: @escaping (Result<[String: Any],
Error>) -> Void) {
        let method = "GET"
        let path = "/ecosystems"

        guard let url = URL(string: "\(baseURL)\(path)") else {
            completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                userInfo: [NSLocalizedDescriptionKey: "Invalid URL"])))
            return
        }

        var request = URLRequest(url: url)
        request.httpMethod = method
        request.setValue("Bearer \(jwtToken)", forHTTPHeaderField: "Authorization")
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        let task = URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }
```

```
            guard let data = data else {
                completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                    userInfo: [NSLocalizedDescriptionKey: "No data received"])))
                return
            }

            do {
                if let json = try JSONSerialization.jsonObject(with: data) as? [String: Any] {
                    print("API Response: \(json)")
                    completion(.success(json))
                } else {
                    completion(.failure(NSError(domain: "VDBAPIClient", code: -1,
                        userInfo: [NSLocalizedDescriptionKey: "Invalid response format"])))
                }
            } catch {
                completion(.failure(error))
            }
        }
        task.resume()
    }
}

let client = VDBAPIClient(accessKey: VVD_ACCESS_KEY, secretKey: VVD_SECRET)

client.getJWTToken { result in
    switch result {
    case .success(let jwtToken):
        client.makeAPIRequest(jwtToken: jwtToken) { result in
            switch result {
            case .success(let response):
                print("Success! Response: \(response)")
            case .failure(let error):
                print("API request failed: \(error.localizedDescription)")
            }
        }
    case .failure(let error):
        print("Failed to obtain JWT token: \(error.localizedDescription)")
    }
}
```

6.8 Java (Android)

```
import com.amazonaws.DefaultRequest;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.AWS4Signer;
import com.amazonaws.http.HttpMethodName;
```

```
import com.google.gson.Gson;
import com.google.gson.JsonObject;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import java.io.ByteArrayInputStream;
import java.net.URI;
import java.util.Map;

public class VDBAPIClient {

    private static final String VVD_ORG = "";
    private static final String VVD_SECRET = "";
    private static final String VVD_ACCESS_KEY = VVD_ORG;

    private static final String BASE_URL = "https://api.vdb.vulnetix.com/v1";
    private static final String REGION = "us-east-1";
    private static final String SERVICE_NAME = "vdb";

    private final OkHttpClient httpClient;
    private final Gson gson;
    private String jwtToken;

    public VDBAPIClient() {
        this.httpClient = new OkHttpClient();
        this.gson = new Gson();
    }

    private Map<String, String> signRequest(String method, String path, String content) {
        try {
            AWSCredentials credentials = new BasicAWSCredentials(VVD_ACCESS_KEY,
VVD_SECRET);

            DefaultRequest<?> request = new DefaultRequest<>(SERVICE_NAME);
            request.setHttpMethod(HttpMethodName.valueOf(method));
            request.setEndpoint(URI.create(BASE_URL));
            request.setResourcePath(path);

            if (content != null && !content.isEmpty()) {
                request.setContent(new ByteArrayInputStream(content.getBytes("UTF-8")));
            }

            AWS4Signer signer = new AWS4Signer();
            signer.setRegionName(REGION);
            signer.setServiceName(SERVICE_NAME);
            signer.sign(request, credentials);

            return request.getHeaders();
```

```
    } catch (Exception e) {
        throw new RuntimeException("Failed to sign request", e);
    }
}

public String getJWTToken() throws Exception {
    String path = "/auth/token";
    String url = BASE_URL + path;

    Map<String, String> signedHeaders = signRequest("GET", path, "");

    Request.Builder requestBuilder = new Request.Builder().url(url);

    for (Map.Entry<String, String> entry : signedHeaders.entrySet()) {
        requestBuilder.addHeader(entry.getKey(), entry.getValue());
    }

    Request request = requestBuilder.build();

    try (Response response = httpClient.newCall(request).execute()) {
        if (!response.isSuccessful()) {
            throw new Exception("Failed to obtain JWT token: " + response.code());
        }

        String responseBody = response.body().string();
        JsonObject jsonResponse = gson.fromJson(responseBody, JsonObject.class);

        if (!jsonResponse.has("token")) {
            throw new Exception("No token in response");
        }

        this.jwtToken = jsonResponse.get("token").getAsString();

        System.out.println("JWT token obtained (expires in 15 minutes): " +
            this.jwtToken.substring(0, Math.min(50, this.jwtToken.length())) + "...");
        System.out.println("Token issuer: " + jsonResponse.get("iss").getAsString());
        System.out.println("Token subject: " + jsonResponse.get("sub").getAsString());
        System.out.println("Token expiry: " + jsonResponse.get("exp").getAsLong());

        return this.jwtToken;
    }
}

public JsonObject makeAPIRequest() throws Exception {
    if (this.jwtToken == null) {
        getJWTToken();
    }
```

```
        String path = "/ecosystems";
        String url = BASE_URL + path;

        System.out.println("Making GET request to " + path + "...");

        Request request = new Request.Builder()
            .url(url)
            .get()
            .addHeader("Authorization", "Bearer " + this.jwtToken)
            .addHeader("Content-Type", "application/json")
            .build();

        try (Response response = httpClient.newCall(request).execute()) {
            if (!response.isSuccessful()) {
                throw new Exception("API request failed: " + response.code());
            }

            String responseBody = response.body().string();
            JsonObject jsonResponse = gson.fromJson(responseBody, JsonObject.class);

            System.out.println("API Response: " + gson.toJson(jsonResponse));

            return jsonResponse;
        }
    }

    public static void main(String[] args) {
        try {
            VDBAPIClient client = new VDBAPIClient();

            client.getJWTToken();

            JsonObject result = client.makeAPIRequest();

            System.out.println("Request completed successfully!");
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

6.9 TypeScript (Node.js)

```
import fetch from 'node-fetch';
import crypto from 'crypto';
```

```
const VVD_ORG = "";
const VVD_SECRET = "";
const VVD_ACCESS_KEY = VVD_ORG;
const BASE_URL = "https://api.vdb.vulnetix.com/v1";

function hmacSha512(key: string | Buffer, data: string): Buffer {
    return crypto.createHmac('sha512', key).update(data).digest();
}

function sha512(data: string): string {
    return crypto.createHash('sha512').update(data).digest('hex');
}

function getSignatureKey(key: string, dateStamp: string, region: string, service: string):
Buffer {
    const kDate = hmacSha512(`AWS4${key}`, dateStamp);
    const kRegion = hmacSha512(kDate, region);
    const kService = hmacSha512(kRegion, service);
    const kSigning = hmacSha512(kService, 'aws4_request');
    return kSigning;
}

function signRequest(
    accessKey: string,
    secretKey: string,
    method: string,
    path: string,
    body: string = ''
): Record<string, string> {
    const now = new Date();
    const amzDate = now.toISOString().replace(/[:-]|\.\d{3}/g, '');
    const dateStamp = amzDate.substring(0, 8);
    const region = 'us-east-1';
    const service = 'vdb';

    const payloadHash = sha512(body);
    const canonicalHeaders = `x-amz-date:${amzDate}\n`;
    const signedHeaders = 'x-amz-date';

    const canonicalRequest = [
        method,
        path,
        '',
        canonicalHeaders,
        signedHeaders,
        payloadHash
    ].join('\n');
```

```
    const algorithm = 'AWS4-HMAC-SHA512';
    const credentialScope = `${dateStamp}/${region}/${service}/aws4_request`;
    const stringToSign = [
        algorithm,
        amzDate,
        credentialScope,
        sha512(canonicalRequest)
    ].join('\n');

    const signingKey = getSignatureKey(secretKey, dateStamp, region, service);
    const signature = hmacSha512(signingKey, stringToSign).toString('hex');

    const authHeader = `${algorithm} Credential=${accessKey}/${credentialScope},
SignedHeaders=${signedHeaders}, Signature=${signature}`;

    return {
        'X-Amz-Date': amzDate,
        'Authorization': authHeader
    };
}

async function getJWTToken(): Promise<string> {
    const path = '/auth/token';
    const signedHeaders = signRequest(VVD_ACCESS_KEY, VVD_SECRET, 'GET', path);

    const response = await fetch(`${BASE_URL}${path}`, {
        method: 'GET',
        headers: signedHeaders
    });

    if (!response.ok) {
        throw new Error(`Failed to obtain JWT token: ${response.status}`);
    }

    const data = await response.json() as {
        token: string;
        iss: string;
        sub: string;
        exp: number;
    };

    console.log(`JWT token obtained (expires in 15 minutes): ${data.token.substring(0,
50)}...`);
    console.log('Token details:', {
        iss: data.iss,
        sub: data.sub,
        exp: new Date(data.exp * 1000).toISOString()
    });
```

```
    return data.token;
}

async function makeAPIRequest(): Promise<void> {
    const jwtToken = await getJWTToken();

    const response = await fetch(`${BASE_URL}/ecosystems`, {
        method: 'GET',
        headers: {
            'Authorization': `Bearer ${jwtToken}`,
            'Content-Type': 'application/json'
        }
    });

    if (!response.ok) {
        throw new Error(`API request failed: ${response.status}`);
    }

    const data = await response.json();
    console.log('API Response:', JSON.stringify(data, null, 2));
}

makeAPIRequest().catch(console.error);
```

---

7. Error Handling

7.1 Error Response Format

All API errors return a consistent JSON structure:

```
{
  "success": false,
  "error": "Error category or message",
  "details": "Additional context (optional)"
}
```

7.2 HTTP Status Codes

Code	Meaning	Common Causes
400	Bad Request	Missing or invalid parameters
401	Unauthorized	Missing, invalid, or expired JWT token

403	Forbidden	Organization inactive or blocked
404	Not Found	Resource doesn't exist
429	Too Many Requests	Rate limit or quota exceeded
500	Internal Server Error	Unexpected server error

7.3 Authentication Errors (401)

Missing Token:

```
{
  "success": false,
  "error": "Missing Authorization header. Please provide a Bearer token."
}
```

Invalid Token Format:

```
{
  "success": false,
  "error": "Invalid Authorization header format. Expected \"Bearer <token>\"."
}
```

Expired Token:

```
{
  "success": false,
  "error": "Token has expired. Please obtain a new token from /v1/auth/token."
}
```

Invalid Signature (SigV4):

```
{
  "success": false,
  "error": "Invalid signature"
}
```

7.4 Rate Limit Errors (429)

```
{
  "success": false,
  "error": "Rate limit exceeded",
  "details": "Too many requests. Limit: 60 requests per minute. Try again in 42 seconds."
}
```

Headers:

RateLimit-MinuteLimit: 60
RateLimit-WeekLimit: 10000
RateLimit-Remaining: 0
RateLimit-Reset: 42

7.5 Resource Not Found (404)


```
{
  "error": "Vulnerability not found",
  "identifier": "CVE-2024-99999"
}
```


7.6 Error Handling Best Practices

Retry with Exponential Backoff:

```
import time
import requests

def retry_with_backoff(func, max_retries=3, base_delay=1):
    for attempt in range(max_retries):
        try:
            response = func()

            if 400 <= response.status_code < 500:
                return response

            if response.status_code >= 500:
                if attempt < max_retries - 1:
                    delay = base_delay * (2 ** attempt)
                    print(f"Server error. Retrying in {delay}s...")
                    time.sleep(delay)
                    continue

            return response

        except requests.exceptions.RequestException as e:
            if attempt < max_retries - 1:
                delay = base_delay * (2 ** attempt)
                print(f"Request failed: {e}. Retrying in {delay}s...")
                time.sleep(delay)
            else:
                raise
```

```
    raise Exception("Max retries exceeded")
```

Token Refresh on Expiry:

```
class VDBClient {
  constructor(accessKey, secretKey) {
    this.accessKey = accessKey;
    this.secretKey = secretKey;
    this.token = null;
    this.tokenExpiry = 0;
  }

  async getValidToken() {
    if (!this.token || Date.now() / 1000 > this.tokenExpiry - 60) {
      await this.refreshToken();
    }
    return this.token;
  }

  async refreshToken() {
    const response = await this.requestTokenWithSigV4();
    const data = await response.json();

    this.token = data.token;
    this.tokenExpiry = data.exp;
  }

  async apiRequest(endpoint) {
    const token = await this.getValidToken();

    const response = await fetch(`https://api.vdb.vulnetix.com${endpoint}`, {
      headers: { 'Authorization': `Bearer ${token}` }
    });

    if (response.status === 401) {
      const error = await response.json();
      if (error.error.includes('expired')) {
        await this.refreshToken();
        return this.apiRequest(endpoint);
      }
    }

    return response;
  }
}
```

Rate Limit Handler:

```
func handleRateLimit(resp *http.Response) error {
    if resp.StatusCode != 429 {
        return nil
    }

    resetHeader := resp.Header.Get("RateLimit-Reset")
    if resetHeader == "" {
        return fmt.Errorf("rate limited but no reset header")
    }

    resetSeconds, err := strconv.Atoi(resetHeader)
    if err != nil {
        return fmt.Errorf("invalid reset header: %w", err)
    }

    fmt.Printf("Rate limited. Waiting %d seconds...\n", resetSeconds)
    time.Sleep(time.Duration(resetSeconds) * time.Second)

    return nil
}
```

Support

This user guide provides comprehensive coverage of the VDB Manager API, from initial setup through advanced error handling. For additional support:

- **Email:** sales@vulnetix.com
- **API Specification:** https://redocly.github.io/redoc/?url=https://api.vdb.vulnetix.com/v1/spec

Always ensure you're using the latest version of the API specification.