Tenable and AWS Secrets Manager Integration Guide

Last Revised: July 23, 2026

Table of Contents

Welcome to Tenable for AWS Secrets Manager

This document provides information and steps for integrating Tenable One Vulnerability Management or Tenable Nessus with AWS Secrets Manager.

AWS Secrets Manager is an AWS service that allows you to store, rotate, and retrieve secrets such as database credentials, application credentials, OAuth tokens, API keys, usernames, and passwords. The Tenable integration with the AWS Secrets Manager API allows you to retrieve target login credentials directly from AWS at scan time, and eliminates the need to store scan credentials in Tenable. The integration acts as a PAM (Privileged Access Management) integration: Tenable One Vulnerability Management (or Tenable Security Center) passes the credential configuration to Tenable Nessus. The scanner then calls the AWS Secrets Manager API to fetch the secret value and uses the returned credentials (username, password, and/or SSH key) to authenticate to the scan target.

Tenable provides this integration as an authentication method for the following credential types:

- Host > Windows (SMB / WMI)
- Host > SSH
- Database (PostgreSQL, MongoDB, Cassandra, DB2, MySQL, SQL Server, Oracle)
- Miscellaneous > VMware ESX SOAP API
- Miscellaneous > VMware vCenter API
- Miscellaneous > Nutanix Prism Central

For more information about the related Tenable products, refer to the following user guides:

- Tenable One Vulnerability Management
- Tenable Nessus

What information does the AWS Secrets Manager integration collect?

The AWS Secrets Manager integration collects configuration values that you supply in the Tenable credential UI, and retrieves the secret based on those values from AWS at scan time.

AWS Secrets Manager-related configurations:

- **AWS Region** (required, default **us-east-1**) – selects the regional AWS Secrets Manager endpoint, `secretsmanager.<region>.amazonaws.com`, and is used in the AWS Signature V4 credential scope.
- **AWS Access Key ID** (required) – the IAM access key ID used to authenticate to AWS.
- **AWS Secret Access Key** (required) – the IAM secret access key used as the seed for the AWS4-HMAC-SHA256 signing key. Never sent on the wire.
- **AWS Session Token** (optional) – included as the `X-Amz-Security-Token` header when using AWS STS temporary credentials. Leave blank for long-lived IAM user keys.
- **Credential ID** (required) – the secret name or full ARN passed as `SecretId` in the `GetSecretValue` API call.

Data retrieved from AWS Secrets Manager at scan time

The AWS Secrets Manager integration retrieves a single secret per unique Credential ID per scan chunk. Each retrieved secret value is parsed as JSON; if the secret is plain text, the entire string is treated as the password. The integration extracts the following fields from the parsed secret value (when present):

- **username** – the login name used to authenticate to the target. Overridden by the **Username** field of the credential when set.

- **password** – the login password. May also carry an SSH private key – see the *password vs. ssh_key* note below.
- **ssh_key** – an SSH private key (PEM-encoded). When present, it is used in place of **password** for SSH authentication.
- **passphrase** – the passphrase that decrypts an encrypted SSH private key. Only consulted when an SSH key is supplied (either via the `ssh_key` field or via a `password` field that contains PEM key material).
- **domain** – the Windows or Active Directory domain. Used only when **Fetch Domain** is enabled in the credential.

> **Note: password vs. ssh_key**: If `ssh_key` is present in the secret, it always takes precedence and is used as the SSH private key. If `ssh_key` is absent but the `password` value contains PEM markers (the literal text `-----BEGIN` or `PRIVATE KEY`), the integration treats the `password` field as an SSH private key rather than as a password. In every other case, the `password` field is used as a plain password. The `passphrase` field is only applied when one of those two SSH-key paths is taken.

If privilege escalation is configured for an SSH credential, the integration retrieves a second secret (the **Escalation Credential ID**) and uses its `password` field as the sudo/escalation password.

API requests per target

The integration makes one POST to `secretsmanager.<region>.amazonaws.com` per unique secret per scan chunk. Results are cached for the duration of the chunk by the MD5 of the secret ID, so multiple targets that share a secret only require one API call. When privilege escalation uses a separate Escalation Credential ID, one additional secret retrieval is performed.

Credentialed scans

Credentialed scans allow the Tenable scanner to log in to the target system directly and perform a deeper assessment than is possible without credentials. This includes checking installed software versions, configuration settings, patch levels, and compliance status.

What the AWS Secrets Manager integration does not collect

This integration does not support the following Tenable credential types as an authentication method. For these types, you must use a different PAM integration or configure credentials within the scan policy:

- F5 Networks
- PAN-OS
- SNMPv3

The integration does not perform dynamic scanning or auto-discovery. AWS Secrets Manager stores secrets only and does not expose target host names or IP addresses. Scan targets must be supplied through the normal target list or asset source.

Compliance audit credentials are not separately fetched by this integration; the integration returns a single login credential and that same credential is used for compliance checks when enabled in the scan template.

AWS Secrets Manager integration limitations

AWS Secrets Manager allows up to 10,000 API requests per second per AWS account, which is well above the request rate generated by Tenable scans. The integration further reduces request volume by caching each secret per scan chunk.

Each retrieved secret incurs the standard AWS Secrets Manager pricing. Refer to Amazon for current AWS pricing.

Configure AWS Secrets Manager Permissions

To use this integration, you must provide AWS IAM credentials (access key ID and secret access key, optionally with a session token) for an IAM identity that is permitted to read the target secret(s).

Before you begin:

Ensure that your IAM identity is permitted to read the following:

- `secretsmanager:GetSecretValue` — required for every secret the integration reads.
- `kms:Decrypt` — required only if the secret is encrypted with a customer-managed AWS KMS key (not required when the secret uses the default AWS-managed `aws/secretsmanager` key).

To configure the required AWS IAM permissions:

1. **Create a dedicated IAM identity.** In the AWS IAM console, create an IAM user (or assumable role) dedicated to Tenable scans.

   > **Note**: Tenable recommends a dedicated identity so that integration activity is auditable and can be revoked independently.

2. **Attach a least-privilege policy.** Attach an IAM policy granting `secretsmanager:GetSecretValue` on the ARN of each secret to be read.

   > **Note**: Use ARN wildcards (for example, `arn:aws:secretsmanager:us-east-1:123456789012:secret:tenable/*`) if you want to grant access to a group of secrets without listing them individually. If the secret uses a customer-managed AWS KMS key, also grant `kms:Decrypt` on that key's ARN.

3. **Generate access credentials.** Generate an access key ID and secret access key for the IAM user, or use AWS STS (`aws sts get-session-token` or `aws sts assume-role`) to obtain temporary credentials and supply the resulting session token in the **AWS Session Token** field

of the credential.

> **Note**: Refer to the AWS Identity and Access Management documentation for full details.

Configure Tenable One Vulnerability Management Scans

Required User Role: Standard, Scan Manager, or Administrator

To configure scans with the AWS Secrets Manager integration in Tenable One Vulnerability Management, complete the following steps:

1. Log into your Tenable user interface.
2. In the upper-left corner, click the **Menu** button.

   The left navigation plane appears.
3. In the left navigation plane, click **Scans**.

   The **Scans** page appears.
4. In the upper-right corner of the page, click the **Create Scan** button.

   The **Select a Scan Template** page appears.
5. Select a scan template.

   The **Scan Configuration** page appears.
6. In the **Name** box, type a name for the scan.
7. In the **Targets** box, type an IP address, hostname, or range of IP addresses for the scan target (s).
8. (Optional) Add a description, folder location, scanner location, and specify target groups.
9. Click the **Credentials** tab.

   The **Credentials** pane appears.

10. Click the credential category that matches your target – **Host** (for SSH or Windows), **Database** (for PostgreSQL, MongoDB, Cassandra, DB2, MySQL, SQL Server, or Oracle), or **Miscellaneous** (for VMware ESX SOAP API, VMware vCenter API, or Nutanix Prism Central).

11. Within that category, select the credential type, and from the authentication method dropdown, choose **AWS Secrets Manager**.

    The AWS Secrets Manager options appear.

12. Configure the options described in the following tables.

    > **Note:** You can configure multiple AWS Secrets Manager credentials in a single scan policy – for example, one credential for SSH targets and a separate credential for Windows targets, each pointing to a different secret.

    The following options apply to every credential type that supports AWS Secrets Manager.

Option	Description	Required
Username	Username to log in to the target. Optional – when blank, the username stored in the AWS secret is used.	No
AWS Region	The AWS region where the secret is stored (for example, **us-east-1**, **us-west-2**, **eu-central-1**). The default is **us-east-1**.	Yes
AWS Access Key ID	The AWS IAM access key ID with permission to read the secret.	Yes
AWS Secret Access Key	The AWS IAM secret access key paired with the access key ID.	Yes
AWS Session Token	Optional session token for temporary AWS	No

	credentials issued by AWS STS. Leave blank when using long-lived IAM user credentials.	
Credential ID	The ID, name, or ARN of the secret in AWS Secrets Manager that contains the login credentials for the scan target.	Yes

Additional options for Host > Windows

Option	Description	Required
Domain	Domain to use to log in to the target. Optional.	No
Use Kerberos KDC	When enabled, Kerberos authentication is used to log in to the Windows target. Reveals additional Kerberos KDC fields (KDC host, port, transport, domain).	No
Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret instead of the Domain field above. Default Off.	No

Additional options for Host > SSH

Option	Description	Required
Use Kerberos KDC	When enabled, Kerberos authentication is used. Reveals a KDC Domain field.	No

Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret. Default Off.	No
Elevate privileges with	Privilege escalation method (**.k5login**, **Cisco 'enable'**, **dzdo**, **pbrun**, **su**, **su+sudo**, **sudo**, or **Nothing**). When set, an Escalation Credential ID field appears.	No
Escalation Credential ID	The ID, name, or ARN of an additional secret containing the escalation or sudo password. If blank, the integration falls back to the secret named in Credential ID.	Yes, when Elevate privileges is set

Additional options for Database

Option	Description	Required
Database Port	Port the target database listens on. Defaults vary by database type (PostgreSQL 5432, MySQL 3306, MongoDB 27017, Cassandra 9042, DB2 50000, SQL Server 1433, Oracle 1521).	Yes
Database Name / Instance name	Name of the database (or, for SQL Server, the instance name) to connect to. Required for DB2; optional for PostgreSQL, MongoDB, and SQL Server.	Varies
Auth type	Database-specific auth type. SQL Server: **Windows** or **SQL**. Oracle: **SYSDBA**,	Yes (SQL Server,

	SYSOPER, or **NORMAL**.	Oracle)
Service type / Service	Oracle only: the service identifier type (**SID** or **Service Name**) and the value.	Yes (Oracle)

Additional options for VMware ESX SOAP API, VMware vCenter API, and Nutanix Prism Central

Option	Description	Required
Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret. Available for VMware ESX SOAP and VMware vCenter; not present on Nutanix Prism Central. Default Off.	No

13. Save the credential and launch the scan.

Configure Tenable Nessus Scans

Required User Role: Standard, Scan Manager, or Administrator

To configure scans with the AWS Secrets Manager integration in Tenable Nessus, complete the following steps:

1. Log in to your Tenable user interface.
2. In the upper-left corner, click the **Menu** button.

   The left navigation plane appears.
3. In the left navigation plane, click **Scans**.

   The **Scans** page appears.
4. In the upper-right corner of the page, click the **Create Scan** button. The **Select a Scan Template** page appears.
5. Select a scan template.

   The **Scan Configuration** page appears.
6. In the **Name** box, type a name for the scan.
7. In the **Targets** box, type an IP address, hostname, or range of IP addresses for the scan target (s).
8. (Optional) Add a description, folder location, scanner location, and specify target groups.
9. Click the **Credentials** tab.

   The **Credentials** pane appears.

10. Click the credential category that matches your target – **Host** (for SSH or Windows), **Database** (for PostgreSQL, MongoDB, Cassandra, DB2, MySQL, SQL Server, or Oracle), or **Miscellaneous** (for VMware ESX SOAP API, VMware vCenter API, or Nutanix Prism Central).
11. Within that category, select the credential type, and from the authentication method dropdown choose **AWS Secrets Manager**. The AWS Secrets Manager options appear.
12. Configure the options described in the following tables.

> **Note:** You can configure multiple AWS Secrets Manager credentials in a single scan policy – for example, one credential for SSH targets and a separate credential for Windows targets, each pointing to a different secret.

The following options apply to every credential type that supports AWS Secrets Manager.

Option	Description	Required
Username	Username to log in to the target. Optional – when blank, the username stored in the AWS secret is used.	No
AWS Region	The AWS region where the secret is stored (for example, **us-east-1**, **us-west-2**, **eu-central-1**). Default is **us-east-1**.	Yes
AWS Access Key ID	The AWS IAM access key ID with permission to read the secret.	Yes
AWS Secret Access Key	The AWS IAM secret access key paired with the access key ID.	Yes
AWS Session Token	Optional session token for temporary AWS credentials issued by AWS STS. Leave	No

	blank when using long-lived IAM user credentials.	
Credential ID	The ID, name, or ARN of the secret in AWS Secrets Manager that contains the login credentials for the scan target.	Yes

Additional options for Host > Windows

Option	Description	Required
Domain	Domain to use to log in to the target. Optional.	No
Use Kerberos KDC	When enabled, Kerberos authentication is used to log in to the Windows target. Reveals additional Kerberos KDC fields (KDC host, port, transport, domain).	No
Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret instead of the Domain field above. Default Off.	No

Additional options for Host > SSH

Option	Description	Required
Use Kerberos KDC	When enabled, Kerberos authentication is used. Reveals a KDC Domain field.	No

Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret. Default Off.	No
Elevate privileges with	Privilege escalation method (**.k5login**, **Cisco 'enable'**, **dzdo**, **pbrun**, **su**, **su+sudo**, **sudo**, or **Nothing**). When set, an Escalation Credential ID field appears.	No
Escalation Credential ID	The ID, name, or ARN of an additional secret containing the escalation or sudo password. If blank, the integration falls back to the secret named in Credential ID.	Yes, when Elevate privileges is set

Additional options for Database

Option	Description	Required
Database Port	Port the target database listens on. Defaults vary by database type (PostgreSQL 5432, MySQL 3306, MongoDB 27017, Cassandra 9042, DB2 50000, SQL Server 1433, Oracle 1521).	Yes
Database Name / Instance name	Name of the database (or, for SQL Server, the instance name) to connect to. Required for DB2; optional for PostgreSQL, MongoDB, and SQL Server.	Varies
Auth type	Database-specific auth type. SQL Server: **Windows** or **SQL**. Oracle: **SYSDBA**,	Yes (SQL Server,

	SYSOPER, or **NORMAL**.	Oracle)
Service type / Service	Oracle only: the service identifier type (**SID** or **Service Name**) and the value.	Yes (Oracle)

Additional options for VMware ESX SOAP API, VMware vCenter API, and Nutanix Prism Central

Option	Description	Required
Fetch Domain	When enabled, the integration uses the `domain` field stored in the AWS secret. Available for VMware ESX SOAP and VMware vCenter; not present on Nutanix Prism Central. Default Off.	No

13. Save the credential and launch the scan.

Debug Log Reporting

Plugin # 84239: **Debugging Log Report**. Logs generated by other plugins are reported by this plugin. Plugin debugging must be enabled in the scan policy for this plugin to run.

When logging is enabled, it writes a per-scan debug log file. The filename follows the pattern `<calling-plugin>~AWS Secrets Manager`. One log file is created per Settings plugin that invokes the integration:

- `logins.nasl~AWS Secrets Manager` – Windows / SMB credential lookups.
- `ssh_settings.nasl~AWS Secrets Manager` – SSH credential lookups.

- `database_settings.nasl~AWS Secrets Manager` – Database credential lookups (PostgreSQL, MongoDB, Cassandra, DB2, MySQL, SQL Server, Oracle).
- `vmware_soap_settings.nasl~AWS Secrets Manager` – VMware ESX / ESXi (SOAP API) credential lookups, processed by VMware SOAP API Settings (Plugin # 57395).
- `vmware_vcenter_settings.nasl~AWS Secrets Manager` – VMware vCenter (vCenter API) credential lookups, processed by VMware vCenter Settings (Plugin # 63060).
- `nutanix_settings.nasl~AWS Secrets Manager` – Nutanix Prism Central credential lookups.

What the logs contain

These log files contain the following information:

- The full integration configuration (region, host, port, SSL settings, Kerberos / domain handling, escalation settings). Secret values, SecretString payload contents, password fields, and SSH keys are scrubbed.
- AWS Signature Version 4 signing details (canonical headers, signed headers, credential scope, calculated signature).
- Outgoing API request headers and the high-level success or failure of the `GetSecretValue` call.
- Cache hits and misses keyed by the MD5 of the secret ID.
- Errors raised by the integration.

> **Note:** For authentication-issue troubleshooting, prefer the per-integration log above over the top-level scanner log – the integration log isolates AWS-specific signing and request data and applies the `aws_secretsmanager::scrubber` function so that secret values are redacted from the logged output.

Additional Information

Helpful Tips

The overall process of the AWS Secrets Manager integration is like other PAM integrations, as follows:

- Tenable One Vulnerability Management or Tenable Security Center pass the policy and credential values down to Tenable Nessus. This includes the AWS region, AWS Access Key ID, AWS Secret Access Key, optional Session Token, and the Credential ID (secret name or ARN).
- The Tenable Nessus scanner communicates with the AWS Secrets Manager API using AWS Signature Version 4, and the API returns the secret payload containing the username, password, and/or SSH key required for target authentication.
- The scanner uses these values for target authentication.

Testing Integration Connectivity

To verify that the IAM identity and the secret are reachable from outside Tenable, install the AWS CLI on the same network segment as the Tenable Nessus scanner and run:

```
aws secretsmanager get-secret-value --region <region> --secret-id <credential-id>
```

A successful response includes the secret's ARN, Name, VersionId, CreatedDate, and a SecretString field containing the JSON payload that the integration parses. If the AWS CLI command fails for the IAM identity used in the credential, the Tenable integration also fails with an equivalent error code.

The integration calls the same API (`secretsmanager.GetSecretValue`) as the AWS CLI command above. If the AWS CLI returns the secret successfully but the integration does not, capture the per-integration log (see Debug Log Reporting) and contact Tenable Technical Support.

API Authentication

All API requests are sent over HTTPS to the AWS Secrets Manager regional endpoint, `secretsmanager.<region>.amazonaws.com`. The integration signs every request using AWS Signature Version 4 (AWS4-HMAC-SHA256).

To ensure successful authentication, verify the following:

- **Region match** – the region configured in the credential matches the region the secret was created in.
- **System clock** – the scanner's clock is within 15 minutes of the AWS server clock; AWS Signature V4 rejects requests outside that window.
- **Outbound network access** – the scanner can reach `secretsmanager.<region>.amazonaws.com` on TCP port 443.
- **Endpoint TLS** – TLS is required; plain HTTP is rejected.
- **Header set** – the integration sends `Content-Type`, `Host`, `X-Amz-Date`, `X-Amz-Target: secretsmanager.GetSecretValue`, an `Authorization` header with the AWS4-HMAC-SHA256 signature, and `X-Amz-Security-Token` when a session token is configured.

Example HTTPS request:

```
curl -s -X POST "https://secretsmanager.us-east-2.amazonaws.com/" \
  -H "Content-Type: application/x-amz-json-1.1" \
  -H "X-Amz-Date: 20260626T192121Z" \
  -H "X-Amz-Security-Token: <redacted>" \
  -H "X-Amz-Target: secretsmanager.GetSecretValue" \
  -H "Authorization: AWS4-HMAC-SHA256 Credential=<access-key-id>/20260626/us-east-2/secretsmanager/aws4_request,
SignedHeaders=content-type;host;x-amz-date;x-amz-security-token;x-amz-target, Signature=<redacted>" \
  -d '{"SecretId":"<credential-id>"}'
```

AWS Secrets Manager Credential Fields, Usage, and Limitations

Required Fields

AWS Region

The AWS region where the secret is stored — for example, `us-east-1`, `us-west-2`, or `eu-central-1`. The integration constructs the API host as `secretsmanager.<region>.amazonaws.com` and signs requests using this region in the AWS Signature V4 credential scope. The default value is `us-east-1`; if your secret lives in a different region, you must change this value or AWS will return a ResourceNotFoundException even when the credentials are otherwise valid.

AWS Access Key ID

The IAM access key ID for the identity that reads the secret. This value is sent in the `Authorization` header as part of the AWS4 credential string and is visible in the debug log.

AWS Secret Access Key

The IAM secret access key paired with the access key ID. The secret access key is used as the seed for the AWS Signature V4 HMAC-SHA256 signing key and is never sent on the wire.

AWS Session Token

Optional. Supply this field when you are using AWS STS temporary credentials (for example, those returned by `aws sts assume-role` or by an EC2 instance profile). When set, the integration sends the token in the `X-Amz-Security-Token` header and includes it in the canonical headers used for signing. Leave this field blank when using long-lived IAM user access keys.

Credential ID

The identifier of the secret to read. AWS Secrets Manager accepts either the secret's friendly name (for example, `tenable/scan/web-server`) or the full ARN (for example, `arn:aws:secretsmanager:us-east-1:123456789012:secret:tenable/scan/web-server-AbCd12`).

The expected payload of the secret is a JSON object containing one or more of `username`, `password`, `ssh_key`, `passphrase`, and `domain`. If the secret is stored as plain text rather than JSON, the entire string is treated as the password and no username, SSH key, passphrase, or domain is extracted.

When the target uses an SSH private key, the key must be embedded in the JSON payload as a single string with line breaks encoded as the `\n` escape sequence. The simplest reliable approach is to construct the secret using a JSON-aware tool, for example:

```
json.dumps({"username": "svc", "passphrase": "...", "ssh_key": open("id_rsa").read()})
```

Then pass the resulting file to:

```
aws secretsmanager create-secret --secret-string file://secret.json
```

Pasting raw, unescaped key text into the AWS console Plaintext editor will produce invalid JSON and the integration will fall back to treating the entire payload as a password – SSH authentication will then fail.

Supported SSH private key formats: The integration forwards the `ssh_key` (or PEM-bearing password) string to the scanner's SSH stack, which accepts RSA, DSA (legacy), ECDSA (nistp256, nistp384, nistp521), and Ed25519 key types. Supported encodings are PEM/PKCS#1, PEM/PKCS#8, and OpenSSH format. PuTTY .ppk keys are not supported – convert them with `puttygen` before storing them in the secret. Encrypted keys require the matching passphrase to be supplied via the `passphrase` field of the same secret.

Optional Fields

Username

When set, this value overrides the `username` field stored in the AWS secret. Use this field when the same secret stores only a password and is reused across multiple targets that each have a different login name.

Domain (Windows only)

When set, this value is used as the Windows or Active Directory domain for SMB authentication. Combined with Fetch Domain, the domain resolution order is: KDC Domain (when Use Kerberos KDC is on) > Domain (this field) > `domain` field from the AWS secret (when Fetch Domain is on). The first non-empty value in that order is used.

Use Kerberos KDC

Available on Windows and SSH credentials. When enabled, Kerberos authentication is used and additional Key Distribution Center (KDC) fields appear (KDC host, KDC port – default 88, KDC transport – TCP or UDP, and KDC Domain). The integration still fetches the username and password from AWS Secrets Manager, but the scanner uses Kerberos rather than NTLM (or password authentication for SSH) at the target.

Fetch Domain

Available on Windows, SSH, VMware ESX SOAP, and VMware vCenter credentials. Default Off. When set to On, the integration uses the `domain` field stored in the AWS secret JSON payload as the Windows / AD domain.

Elevate privileges with (SSH only)

Selects the privilege escalation mechanism for SSH credentialed scanning. Supported values are **.k5login**, **Cisco 'enable'**, **dzdo**, **pbrun**, **su**, **su+sudo**, **sudo**, and **Nothing** (the default). When set to anything other than Nothing, an Escalation Credential ID field appears.

Escalation Credential ID (SSH only)

The ID, name, or ARN of an additional secret containing the password used for privilege escalation (for example, the sudo password or the Cisco enable password). The integration retrieves this secret with a separate `GetSecretValue` call and uses its `password` field. If Elevate privileges with is set but Escalation Credential ID is left blank, the integration falls back to the secret named in Credential ID.

Database Port, Database Name, Auth type, Service type, Service (Database only)

Standard database connection options. The default port varies by database type (see the Scan Configuration table). Database Name is required for DB2 and optional elsewhere. Auth type is required for SQL Server (Windows or SQL) and Oracle (SYSDBA, SYSOPER, or NORMAL). Service type and Service are required for Oracle and select between SID and Service Name.

Potential Issues when First Enabling the AWS Secrets Manager Integration

Incorrect AWS region

If the AWS Region field does not match the region the secret was created in, AWS returns ResourceNotFoundException even when the IAM credentials are otherwise valid. This appears in the integration debug log as a failure on GetSecretValue with HTTP 400 and an AWS error code of ResourceNotFoundException. Confirm the secret's region in the AWS console and update the credential.

Missing IAM permission to read the secret

If the IAM identity does not have `secretsmanager:GetSecretValue` on the secret's ARN, AWS returns AccessDeniedException (HTTP 400). The integration logs the failure and the calling Settings plugin logs `Failed to retrieve AWS Secrets Manager PAM <type> credentials`. Add the missing permission to the IAM policy attached to the identity, or use a wildcard ARN to grant access to a group of secrets.

Customer-managed KMS key with no kms:Decrypt permission

Secrets encrypted with a customer-managed AWS KMS key require the IAM identity to have `kms:Decrypt` on that key in addition to `secretsmanager:GetSecretValue` on the secret. Without it, AWS returns AccessDeniedException referencing the KMS key. Secrets that use the default `aws/secretsmanager` AWS-managed key do not require an explicit KMS grant.

Expired AWS session token

When the credential uses a session token from AWS STS, the token has a maximum lifetime (15 minutes to 36 hours, depending on how it was issued). After expiration, the API returns ExpiredTokenException. Either issue a longer-lived token or switch to long-lived IAM user access keys. Tenable does not refresh session tokens automatically.

Secret payload missing username, password, or ssh_key

If the secret value is a JSON object that does not contain `password` or `ssh_key`, the integration returns NULL_ACCOUNT and logs `Password or SSH key missing from the secret`. Open the secret in the AWS console and confirm the JSON keys match the integration's expected schema (`username` / `password` / `ssh_key` / `passphrase` / `domain`). Plain-text secrets are accepted and treated as a password-only secret.

SSH private key stored without escaped newlines

PEM-encoded SSH private keys are multi-line, but the AWS Secrets Manager SecretString is a single JSON string. Newlines in the key must be encoded as `\n` inside the JSON value. When the key is pasted into the AWS console Plaintext editor with raw line breaks, the resulting payload is invalid JSON; the integration then falls back to treating the whole SecretString as a password and SSH authentication fails on the target. Construct the secret with a JSON-aware tool, for example:

```
secret = json.dumps({"username": "svc", "passphrase": "...", "ssh_key": open("id_rsa").read()})
aws secretsmanager create-secret --name tenable/scan/host --secret-string file://secret.json
```

If you must use the AWS console, paste the key into a JSON value with each line break replaced by `\n`, for example: `{"ssh_key": "-----BEGIN OPENSSH PRIVATE KEY-----\nb3BlbnNza...\n-----END OPENSSH PRIVATE KEY-----\n"}`.

Encrypted SSH key without a passphrase field

Encrypted PEM keys require the matching passphrase to be supplied in the `passphrase` field of the same secret. If the passphrase field is missing or wrong, the SSH stack on the scanner cannot decrypt the key and the SSH credentialed check logs a key-decoding or authentication failure. Add the `passphrase` key to the secret payload alongside `ssh_key` (or use an unencrypted key).

System clock skew (Signature V4)

AWS Signature Version 4 rejects requests where the X-Amz-Date timestamp is more than 15 minutes from the AWS server clock. If the Tenable Nessus scanner's clock has drifted, AWS returns SignatureDoesNotMatch or RequestTimeTooSkewed. Configure NTP on the scanner host.