Red Hat Enterprise Linux 9

Working with vaults in Identity Management

Storing and managing sensitive data in IdM

Last Updated: 2024-07-30

Red Hat Enterprise Linux 9 Working with vaults in Identity Management

Storing and managing sensitive data in IdM

Legal Notice

Abstract

A vault is a secure location in Red Hat Identity Management (IdM) to store, retrieve, and share sensitive data, such as authentication credentials for services. You can manage vaults using the command line or Ansible Playbooks.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Submitting feedback through Jira (account required)

1. Log in to the Jira website.
2. Click **Create** in the top navigation bar
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Create** at the bottom of the dialogue.

CHAPTER 1. VAULTS IN IDM

This chapter describes vaults in Identity Management (IdM). It introduces the following topics:

- The concept of the vault.
- The different roles associated with a vault .
- The different types of vaults available in IdM based on the level of security and access control .
- The different types of vaults available in IdM based on ownership .
- The concept of vault containers.
- The basic commands for managing vaults in IdM .
- Installing the key recovery authority (KRA), which is a prerequisite for using vaults in IdM .

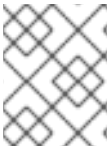
1.1. VAULTS AND THEIR BENEFITS

A vault is a useful feature for those Identity Management (IdM) users who want to keep all their sensitive data stored securely but conveniently in one place. There are various types of vaults and you should choose which vault to use based on your requirements.

A vault is a secure location in (IdM) for storing, retrieving, sharing, and recovering a secret. A secret is security-sensitive data, usually authentication credentials, that only a limited group of people or entities can access. For example, secrets include:

- Passwords
- PINs
- Private SSH keys

A vault is comparable to a password manager. Just like a password manager, a vault typically requires a user to generate and remember one primary password to unlock and access any information stored in the vault. However, a user can also decide to have a standard vault. A standard vault does not require the user to enter any password to access the secrets stored in the vault.



NOTE

The purpose of vaults in IdM is to store authentication credentials that allow you to authenticate to external, non-IdM-related services.

Other important characteristics of the IdM vaults are:

- Vaults are only accessible to the vault owner and those IdM users that the vault owner selects to be the vault members. In addition, the IdM administrator has access to the vault.
- If a user does not have sufficient privileges to create a vault, an IdM administrator can create the vault and set the user as its owner.
- Users and services can access the secrets stored in a vault from any machine enrolled in the IdM domain.

- One vault can only contain one secret, for example, one file. However, the file itself can contain multiple secrets such as passwords, keytabs or certificates.

NOTE

Vault is only available from the IdM command line (CLI), not from the IdM Web UI.

1.2. VAULT OWNERS, MEMBERS, AND ADMINISTRATORS

Identity Management (IdM) distinguishes the following vault user types:

Vault owner

A vault owner is a user or service with basic management privileges on the vault. For example, a vault owner can modify the properties of the vault or add new vault members.
Each vault must have at least one owner. A vault can also have multiple owners.

Vault member

A vault member is a user or service that can access a vault created by another user or service.

Vault administrator

Vault administrators have unrestricted access to all vaults and are allowed to perform all vault operations.

NOTE

Symmetric and asymmetric vaults are protected with a password or key and apply special access control rules (see Vault types). The administrator must meet these rules to:

- Access secrets in symmetric and asymmetric vaults.
- Change or reset the vault password or key.

A vault administrator is any user with the **Vault Administrators** privilege. In the context of the role-based access control (RBAC) in IdM, a privilege is a group of permissions that you can apply to a role.

Vault User

The vault user represents the user in whose container the vault is located. The **Vault user** information is displayed in the output of specific commands, such as **ipa vault-show**:

```
$ ipa vault-show my_vault
  Vault name: my_vault
  Type: standard
  Owner users: user
  Vault user: user
```

For details on vault containers and user vaults, see Vault containers.

Additional resources

- See Standard, symmetric and asymmetric vaults for details on vault types.

1.3. STANDARD, SYMMETRIC, AND ASYMMETRIC VAULTS

Based on the level of security and access control, IdM classifies vaults into the following types:

Standard vaults

Vault owners and vault members can archive and retrieve the secrets without having to use a password or key.

Symmetric vaults

Secrets in the vault are protected with a symmetric key. Vault owners and members can archive and retrieve the secrets, but they must provide the vault password.

Asymmetric vaults

Secrets in the vault are protected with an asymmetric key. Users archive the secret using a public key and retrieve it using a private key. Vault members can only archive secrets, while vault owners can do both, archive and retrieve secrets.

1.4. USER, SERVICE, AND SHARED VAULTS

Based on ownership, IdM classifies vaults into several types. The table below contains information about each type, its owner and use.

Table 1.1. IdM vaults based on ownership

Type	Description	Owner	Note
User vault	A private vault for a user	A single user	Any user can own one or more user vaults if allowed by IdM administrator
Service vault	A private vault for a service	A single service	Any service can own one or more user vaults if allowed by IdM administrator
Shared vault	A vault shared by multiple users and services	The vault administrator who created the vault	Users and services can own one or more user vaults if allowed by IdM administrator. The vault administrators other than the one that created the vault also have full access to the vault.

1.5. VAULT CONTAINERS

A vault container is a collection of vaults. The table below lists the default vault containers that Identity Management (IdM) provides.

Table 1.2. Default vault containers in IdM

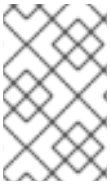
Type	Description	Purpose
User container	A private container for a user	Stores user vaults for a particular user
Service container	A private container for a service	Stores service vaults for a particular service

Type	Description	Purpose
Shared container	A container for multiple users and services	Stores vaults that can be shared by multiple users or services

IdM creates user and service containers for each user or service automatically when the first private vault for the user or service is created. After the user or service is deleted, IdM removes the container and its contents.

1.6. BASIC IDM VAULT COMMANDS

You can use the basic commands outlined below to manage Identity Management (IdM) vaults. The table below contains a list of **ipa vault-*** commands with the explanation of their purpose.



NOTE

Before running any **ipa vault-*** command, install the Key Recovery Authority (KRA) certificate system component on one or more of the servers in your IdM domain. For details, see Installing the Key Recovery Authority in IdM .

Table 1.3. Basic IdM vault commands with explanations

Command	Purpose
ipa help vault	Displays conceptual information about IdM vaults and sample vault commands.
ipa vault-add --help, **ipa vault-find --help**	Adding the **--help** option to a specific **ipa vault-*** command displays the options and detailed help available for that command.

ipa vault-show user_vault --user idm_user	When accessing a vault as a vault member, you must specify the vault owner. If you do not specify the vault owner, IdM informs you that it did not find the vault:
	`[admin@server ~]$ ipa vault-show user_vault`
	`ipa: ERROR: user_vault: vault not found`

ipa vault-show shared_vault --shared	When accessing a shared vault, you must specify that the vault you want to access is a shared vault. Otherwise, IdM informs you it did not find the vault:
	`[admin@server ~]$ ipa vault-show shared_vault`
	`ipa: ERROR: shared_vault: vault not found`

1.7. INSTALLING THE KEY RECOVERY AUTHORITY IN IDM

Follow this procedure to enable vaults in Identity Management (IdM) by installing the Key Recovery Authority (KRA) Certificate System (CS) component on a specific IdM server.

Prerequisites

- You are logged in as **root** on the IdM server.

- An IdM certificate authority is installed on the IdM server.
- You have the **Directory Manager** credentials.

Procedure

- Install the KRA:

```
# ipa-kra-install
```

IMPORTANT

You can install the first KRA of an IdM cluster on a hidden replica. However, installing additional KRAs requires temporarily activating the hidden replica before you install the KRA clone on a non-hidden replica. Then you can hide the originally hidden replica again.

NOTE

To make the vault service highly available and resilient, install the KRA on two IdM servers or more. Maintaining multiple KRA servers prevents data loss.

Additional resources

- See Demoting or promoting hidden replicas.
- See The hidden replica mode.

CHAPTER 2. USING IDM USER VAULTS: STORING AND RETRIEVING SECRETS

This chapter describes how to use user vaults in Identity Management. Specifically, it describes how a user can store a secret in an IdM vault, and how the user can retrieve it. The user can do the storing and the retrieving from two different IdM clients.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see Installing the Key Recovery Authority in IdM.

2.1. STORING A SECRET IN A USER VAULT

Follow this procedure to create a vault container with one or more private vaults to securely store files with sensitive information. In the example used in the procedure below, the **idm_user** user creates a vault of the standard type. The standard vault type ensures that **idm_user** will not be required to authenticate when accessing the file. **idm_user** will be able to retrieve the file from any IdM client to which the user is logged in.

In the procedure:

- **idm_user** is the user who wants to create the vault.
- **my_vault** is the vault used to store the user's certificate.
- The vault type is **standard**, so that accessing the archived certificate does not require the user to provide a vault password.
- **secret.txt** is the file containing the certificate that the user wants to store in the vault.

Prerequisites

- You know the password of **idm_user**.
- You are logged in to a host that is an IdM client.

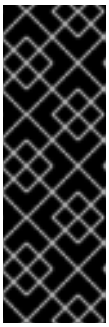
Procedure

1. Obtain the Kerberos ticket granting ticket (TGT) for **idm_user**:

```
$ kinit idm_user
```

2. Use the **ipa vault-add** command with the **--type standard** option to create a standard vault:

```
$ ipa vault-add my_vault --type standard
----------------------
Added vault "my_vault"
----------------------
  Vault name: my_vault
  Type: standard
  Owner users: idm_user
  Vault user: idm_user
```



IMPORTANT

Make sure the first user vault for a user is created by the same user. Creating the first vault for a user also creates the user's vault container. The agent of the creation becomes the owner of the vault container.

For example, if another user, such as **admin**, creates the first user vault for **user1**, the owner of the user's vault container will also be **admin**, and **user1** will be unable to access the user vault or create new user vaults.

3. Use the **ipa vault-archive** command with the **--in** option to archive the **secret.txt** file into the vault:

```
$ ipa vault-archive my_vault --in secret.txt
----------------------------------
Archived data into vault "my_vault"
----------------------------------
```

2.2. RETRIEVING A SECRET FROM A USER VAULT

As an Identity Management (IdM), you can retrieve a secret from your user private vault onto any IdM client to which you are logged in.

Follow this procedure to retrieve, as an IdM user named **idm_user**, a secret from the user private vault named **my_vault** onto **idm_client.idm.example.com**.

Prerequisites

- **idm_user** is the owner of **my_vault**.
- **idm_user** has archived a secret in the vault .
- **my_vault** is a standard vault, which means that **idm_user** does not have to enter any password to access the contents of the vault.

Procedure

1. SSH to **idm_client** as **idm_user**:

```
$ ssh idm_user@idm_client.idm.example.com
```

2. Log in as **idm_user**:

```
$ kinit user
```

3. Use the **ipa vault-retrieve --out** command with the **--out** option to retrieve the contents of the vault and save them into the **secret_exported.txt** file.

```
$ ipa vault-retrieve my_vault --out secret_exported.txt
-------------------------------------
Retrieved data from vault "my_vault"
-------------------------------------
```

2.3. ADDITIONAL RESOURCES

- See Using Ansible to manage IdM service vaults: storing and retrieving secrets .

CHAPTER 3. USING ANSIBLE TO MANAGE IDM USER VAULTS: STORING AND RETRIEVING SECRETS

This chapter describes how to manage user vaults in Identity Management using the Ansible **vault** module. Specifically, it describes how a user can use Ansible playbooks to perform the following three consecutive actions:

- Create a user vault in IdM .
- Store a secret in the vault .
- Retrieve a secret from the vault .

The user can do the storing and the retrieving from two different IdM clients.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see Installing the Key Recovery Authority in IdM.

3.1. ENSURING THE PRESENCE OF A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to use an Ansible playbook to create a vault container with one or more private vaults to securely store sensitive information. In the example used in the procedure below, the **idm_user** user creates a vault of the standard type named **my_vault**. The standard vault type ensures that **idm_user** will not be required to authenticate when accessing the file. **idm_user** will be able to retrieve the file from any IdM client to which the user is logged in.

Prerequisites

- You have installed the ansible-freeipa package on the Ansible controller, that is the host on which you execute the steps in the procedure.
- You know the password of **idm_user**.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Create an inventory file, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open **inventory.file** and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

4. Make a copy of the **ensure-standard-vault-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-standard-vault-is-present.yml ensure-standard-vault-is-present-copy.yml
```

5. Open the **ensure-standard-vault-is-present-copy.yml** file for editing.

6. Adapt the file by setting the following variables in the **ipavault** task section:

   - Set the **ipaadmin_principal** variable to **idm_user**.
   - Set the **ipaadmin_password** variable to the password of **idm_user**.
   - Set the **user** variable to **idm_user**.
   - Set the **name** variable to **my_vault**.
   - Set the **vault_type** variable to **standard**.
     This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_principal: idm_user
      ipaadmin_password: idm_user_password
      user: idm_user
      name: my_vault
      vault_type: standard
```

7. Save the file.

8. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-standard-vault-is-present-copy.yml
```

3.2. ARCHIVING A SECRET IN A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to use an Ansible playbook to store sensitive information in a personal vault. In the example used, the **idm_user** user archives a file with sensitive information named **password.txt** in a vault named **my_vault**.

Prerequisites

- You have installed the ansible-freeipa package on the Ansible controller, that is the host on which you execute the steps in the procedure.
- You know the password of **idm_user**.

- **idm_user** is the owner, or at least a member user of **my_vault**.
- You have access to **password.txt**, the secret that you want to archive in **my_vault**.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Open your inventory file and make sure that the IdM server that you want to configure is listed in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

3. Make a copy of the **data-archive-in-symmetric-vault.yml** Ansible playbook file but replace "symmetric" by "standard". For example:

```
$ cp data-archive-in-symmetric-vault.yml data-archive-in-standard-vault-copy.yml
```

4. Open the **data-archive-in-standard-vault-copy.yml** file for editing.
5. Adapt the file by setting the following variables in the **ipavault** task section:
   - Set the **ipaadmin_principal** variable to **idm_user**.
   - Set the **ipaadmin_password** variable to the password of **idm_user**.
   - Set the **user** variable to **idm_user**.
   - Set the **name** variable to **my_vault**.
   - Set the **in** variable to the full path to the file with sensitive information.
   - Set the **action** variable to **member**.
     This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_principal: idm_user
      ipaadmin_password: idm_user_password
      user: idm_user
      name: my_vault
      in: /usr/share/doc/ansible-freeipa/playbooks/vault/password.txt
      action: member
```

6. Save the file.

7. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file data-archive-in-standard-vault-copy.yml
```

3.3. RETRIEVING A SECRET FROM A STANDARD USER VAULT IN IDM USING ANSIBLE

Follow this procedure to use an Ansible playbook to retrieve a secret from the user personal vault. In the example used in the procedure below, the **idm_user** user retrieves a file with sensitive data from a vault of the standard type named **my_vault** onto an IdM client named **host01**. **idm_user** does not have to authenticate when accessing the file. **idm_user** can use Ansible to retrieve the file from any IdM client on which Ansible is installed.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.
  - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the password of **idm_user**.
- **idm_user** is the owner of **my_vault**.
- **idm_user** has stored a secret in **my_vault**.
- Ansible can write into the directory on the IdM host into which you want to retrieve the secret.
- **idm_user** can read from the directory on the IdM host into which you want to retrieve the secret.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Open your inventory file and mention, in a clearly defined section, the IdM client onto which you want to retrieve the secret. For example, to instruct Ansible to retrieve the secret onto **host01.idm.example.com**, enter:

```
[ipahost]
host01.idm.example.com
```

3. Make a copy of the **retrive-data-symmetric-vault.yml** Ansible playbook file. Replace "symmetric" with "standard". For example:

```
$ cp retrive-data-symmetric-vault.yml retrieve-data-standard-vault.yml-copy.yml
```

4. Open the **retrieve-data-standard-vault.yml-copy.yml** file for editing.

5. Adapt the file by setting the **hosts** variable to **ipahost**.

6. Adapt the file by setting the following variables in the **ipavault** task section:

   - Set the **ipaadmin_principal** variable to **idm_user**.
   - Set the **ipaadmin_password** variable to the password of **idm_user**.
   - Set the **user** variable to **idm_user**.
   - Set the **name** variable to **my_vault**.
   - Set the **out** variable to the full path of the file into which you want to export the secret.
   - Set the **state** variable to **retrieved**.
     This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipahost
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_principal: idm_user
      ipaadmin_password: idm_user_password
      user: idm_user
      name: my_vault
      out: /tmp/password_exported.txt
      state: retrieved
```

7. Save the file.

8. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file retrieve-data-standard-vault.yml-copy.yml
```

Verification steps

1. **SSH** to **host01** as **user01**:

```
$ ssh user01@host01.idm.example.com
```

2. View the file specified by the **out** variable in the Ansible playbook file:

```
$ vim /tmp/password_exported.txt
```

You can now see the exported secret.

- For more information about using Ansible to manage IdM vaults and user secrets and about playbook variables, see the README-vault.md Markdown file available in the **/usr/share/doc/ansible-freeipa/** directory and the sample playbooks available in the **/usr/share/doc/ansible-freeipa/playbooks/vault/** directory.

CHAPTER 4. MANAGING IDM SERVICE SECRETS: STORING AND RETRIEVING SECRETS

This section shows how an administrator can use a service vault in Identity Management (IdM) to securely store a service secret in a centralized location. The vault used in the example is asymmetric, which means that to use it, the administrator needs to perform the following steps:

1. Generate a private key using, for example, the **openssl** utility.
2. Generate a public key based on the private key.

The service secret is encrypted with the public key when an administrator archives it into the vault. Afterwards, a service instance hosted on a specific machine in the domain retrieves the secret using the private key. Only the service and the administrator are allowed to access the secret.

If the secret is compromised, the administrator can replace it in the service vault and then redistribute it to those individual service instances that have not been compromised.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see Installing the Key Recovery Authority in IdM.

This section includes these procedure

1. Storing an IdM service secret in an asymmetric vault
2. Retrieving a service secret for an IdM service instance
3. Changing an IdM service vault secret when compromised

Terminology used

In the procedures:

- **admin** is the administrator who manages the service password.
- **private-key-to-an-externally-signed-certificate.pem** is the file containing the service secret, in this case a private key to an externally signed certificate. Do not confuse this private key with the private key used to retrieve the secret from the vault.
- **secret_vault** is the vault created for the service.
- **HTTP/webserver.idm.example.com** is the service whose secret is being archived.
- **service-public.pem** is the service public key used to encrypt the password stored in **password_vault**.
- **service-private.pem** is the service private key used to decrypt the password stored in **secret_vault**.

4.1. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT

Follow this procedure to create an asymmetric vault and use it to archive a service secret.

Prerequisites

- You know the IdM administrator password.

Procedure

1. Log in as the administrator:

```
$ kinit admin
```

2. Obtain the public key of the service instance. For example, using the **openssl** utility:

   a. Generate the **service-private.pem** private key.

```
$ openssl genrsa -out service-private.pem 2048
Generating RSA private key, 2048 bit long modulus
.+++
.....................................................+++
e is 65537 (0x10001)
```

   b. Generate the **service-public.pem** public key based on the private key.

```
$ openssl rsa -in service-private.pem -out service-public.pem -pubout
writing RSA key
```

3. Create an asymmetric vault as the service instance vault, and provide the public key:

```
$ ipa vault-add secret_vault --service HTTP/webserver.idm.example.com --type
asymmetric --public-key-file service-public.pem
----------------------------
Added vault "secret_vault"
----------------------------
Vault name: secret_vault
Type: asymmetric
Public key: LS0tLS1C...S0tLS0tCg==
Owner users: admin
Vault service: HTTP/webserver.idm.example.com@IDM.EXAMPLE.COM
```

   The password archived into the vault will be protected with the key.

4. Archive the service secret into the service vault:

```
$ ipa vault-archive secret_vault --service HTTP/webserver.idm.example.com --in
private-key-to-an-externally-signed-certificate.pem
---------------------------------
Archived data into vault "secret_vault"
---------------------------------
```

   This encrypts the secret with the service instance public key.

Repeat these steps for every service instance that requires the secret. Create a new asymmetric vault for each service instance.

4.2. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE INSTANCE

Follow this procedure to use a service instance to retrieve the service vault secret using a locally-stored service private key.

Prerequisites

- You have access to the keytab of the service principal owning the service vault, for example HTTP/webserver.idm.example.com.
- You have created an asymmetric vault and archived a secret in the vault .
- You have access to the private key used to retrieve the service vault secret.

Procedure

1. Log in as the administrator:

```
$ kinit admin
```

2. Obtain a Kerberos ticket for the service:

```
# kinit HTTP/webserver.idm.example.com -k -t /etc/httpd/conf/ipa.keytab
```

3. Retrieve the service vault password:

```
$ ipa vault-retrieve secret_vault --service HTTP/webserver.idm.example.com --private-key-file service-private.pem --out secret.txt
----------------------------------
Retrieved data from vault "secret_vault"
----------------------------------
```

4.3. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED

Follow this procedure to isolate a compromised service instance by changing the service vault secret.

Prerequisites

- You know the **IdM administrator** password.
- You have created an asymmetric vault to store the service secret.
- You have generated the new secret and have access to it, for example in the **new-private-key-to-an-externally-signed-certificate.pem** file.

Procedure

1. Archive the new secret into the service instance vault:

```
$ ipa vault-archive secret_vault --service HTTP/webserver.idm.example.com --in new-private-key-to-an-externally-signed-certificate.pem
----------------------------------
Archived data into vault "secret_vault"
----------------------------------
```

This overwrites the current secret stored in the vault.

2. Retrieve the new secret on non-compromised service instances only. For details, see Retrieving a service secret for an IdM service instance.

4.4. ADDITIONAL RESOURCES

- See Using Ansible to manage IdM service vaults: storing and retrieving secrets .

CHAPTER 5. USING ANSIBLE TO MANAGE IDM SERVICE VAULTS: STORING AND RETRIEVING SECRETS

This section shows how an administrator can use the **ansible-freeipa vault** module to securely store a service secret in a centralized location. The vault used in the example is asymmetric, which means that to use it, the administrator needs to perform the following steps:

1. Generate a private key using, for example, the **openssl** utility.
2. Generate a public key based on the private key.

The service secret is encrypted with the public key when an administrator archives it into the vault. Afterwards, a service instance hosted on a specific machine in the domain retrieves the secret using the private key. Only the service and the administrator are allowed to access the secret.

If the secret is compromised, the administrator can replace it in the service vault and then redistribute it to those individual service instances that have not been compromised.

Prerequisites

- The Key Recovery Authority (KRA) Certificate System component has been installed on one or more of the servers in your IdM domain. For details, see Installing the Key Recovery Authority in IdM.

This section includes these procedures:

- Ensuring the presence of an asymmetric service vault in IdM using Ansible
- Storing an IdM service secret in an asymmetric vault using Ansible
- Retrieving a service secret for an IdM service using Ansible
- Changing an IdM service vault secret when compromised using Ansible

In the procedures:

- **admin** is the administrator who manages the service password.
- **private-key-to-an-externally-signed-certificate.pem** is the file containing the service secret, in this case a private key to an externally signed certificate. Do not confuse this private key with the private key used to retrieve the secret from the vault.
- **secret_vault** is the vault created to store the service secret.
- **HTTP/webserver1.idm.example.com** is the service that is the owner of the vault.
- **HTTP/webserver2.idm.example.com** and **HTTP/webserver3.idm.example.com** are the vault member services.
- **service-public.pem** is the service public key used to encrypt the password stored in **password_vault**.
- **service-private.pem** is the service private key used to decrypt the password stored in **secret_vault**.

5.1. ENSURING THE PRESENCE OF AN ASYMMETRIC SERVICE VAULT IN IDM USING ANSIBLE

Follow this procedure to use an Ansible playbook to create a service vault container with one or more private vaults to securely store sensitive information. In the example used in the procedure below, the administrator creates an asymmetric vault named **secret_vault**. This ensures that the vault members have to authenticate using a private key to retrieve the secret in the vault. The vault members will be able to retrieve the file from any IdM client.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.
  - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the **IdM administrator** password.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Obtain the public key of the service instance. For example, using the **openssl** utility:
   a. Generate the **service-private.pem** private key.

```
$ openssl genrsa -out service-private.pem 2048
Generating RSA private key, 2048 bit long modulus
.+++
.....................................................+++
e is 65537 (0x10001)
```

   b. Generate the **service-public.pem** public key based on the private key.

```
$ openssl rsa -in service-private.pem -out service-public.pem -pubout
writing RSA key
```

3. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

4. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

5. Make a copy of the **ensure-asymmetric-vault-is-present.yml** Ansible playbook file. For example:

```
$ cp ensure-asymmetric-vault-is-present.yml ensure-asymmetric-service-vault-is-present-copy.yml
```

6. Open the **ensure-asymmetric-vault-is-present-copy.yml** file for editing.

7. Add a task that copies the **service-public.pem** public key from the Ansible controller to the **server.idm.example.com** server.

8. Modify the rest of the file by setting the following variables in the **ipavault** task section:

   - Set the **ipaadmin_password** variable to the IdM administrator password.
   - Define the name of the vault using the **name** variable, for example **secret_vault**.
   - Set the **vault_type** variable to **asymmetric**.
   - Set the **service** variable to the principal of the service that owns the vault, for example **HTTP/webserver1.idm.example.com**.
   - Set the **public_key_file** to the location of your public key.
     This is the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false
  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Copy public key to ipaserver.
    copy:
      src: /path/to/service-public.pem
      dest: /usr/share/doc/ansible-freeipa/playbooks/vault/service-public.pem
      mode: 0600
  - name: Add data to vault, from a LOCAL file.
    ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      vault_type: asymmetric
      service: HTTP/webserver1.idm.example.com
      public_key_file: /usr/share/doc/ansible-freeipa/playbooks/vault/service-public.pem
```

9. Save the file.

10. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file ensure-asymmetric-service-vault-is-present-copy.yml
```

5.2. ADDING MEMBER SERVICES TO AN ASYMMETRIC VAULT USING ANSIBLE

Follow this procedure to use an Ansible playbook to add member services to a service vault so that they can all retrieve the secret stored in the vault. In the example used in the procedure below, the IdM administrator adds the **HTTP/webserver2.idm.example.com** and **HTTP/webserver3.idm.example.com** service principals to the **secret_vault** vault that is owned by **HTTP/webserver1.idm.example.com**.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.
  - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the **IdM administrator** password.
- You have created an asymmetric vault to store the service secret.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

4. Make a copy of the **data-archive-in-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp data-archive-in-asymmetric-vault.yml add-services-to-an-asymmetric-vault.yml
```

5. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.
6. Modify the file by setting the following variables in the **ipavault** task section:
   - Set the **ipaadmin_password** variable to the IdM administrator password.

- Set the **name** variable to the name of the vault, for example **secret_vault**.
- Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
- Define the services that you want to have access to the vault secret using the **services** variable.
- Set the **action** variable to **member**.
  This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      services:
      - HTTP/webserver2.idm.example.com
      - HTTP/webserver3.idm.example.com
      action: member
```

7. Save the file.
8. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file add-services-to-an-asymmetric-vault.yml
```

5.3. STORING AN IDM SERVICE SECRET IN AN ASYMMETRIC VAULT USING ANSIBLE

Follow this procedure to use an Ansible playbook to store a secret in a service vault so that it can be later retrieved by the service. In the example used in the procedure below, the administrator stores a **PEM** file with the secret in an asymmetric vault named **secret_vault**. This ensures that the service will have to authenticate using a private key to retrieve the secret from the vault. The vault members will be able to retrieve the file from any IdM client.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.

    - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.

- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the **IdM administrator** password.
- You have created an asymmetric vault to store the service secret.
- The secret is stored locally on the Ansible controller, for example in the **/usr/share/doc/ansible-freeipa/playbooks/vault/private-key-to-an-externally-signed-certificate.pem** file.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the IdM server that you want to configure in the **[ipaserver]** section. For example, to instruct Ansible to configure **server.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

4. Make a copy of the **data-archive-in-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp data-archive-in-asymmetric-vault.yml data-archive-in-asymmetric-vault-copy.yml
```

5. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.
6. Modify the file by setting the following variables in the **ipavault** task section:
    - Set the **ipaadmin_password** variable to the IdM administrator password.
    - Set the **name** variable to the name of the vault, for example **secret_vault**.
    - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
    - Set the **in** variable to **"{{ lookup('file', 'private-key-to-an-externally-signed-certificate.pem') | b64encode }}"**. This ensures that Ansible retrieves the file with the private key from the working directory on the Ansible controller rather than from the IdM server.
    - Set the **action** variable to **member**.
      This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
```

```
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      in: "{{ lookup('file', 'private-key-to-an-externally-signed-certificate.pem') | b64encode }}"
      action: member
```

7. Save the file.
8. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file data-archive-in-asymmetric-vault-copy.yml
```

5.4. RETRIEVING A SERVICE SECRET FOR AN IDM SERVICE USING ANSIBLE

Follow this procedure to use an Ansible playbook to retrieve a secret from a service vault on behalf of the service. In the example used in the procedure below, running the playbook retrieves a **PEM** file with the secret from an asymmetric vault named **secret_vault**, and stores it in the specified location on all the hosts listed in the Ansible inventory file as **ipaservers**.

The services authenticate to IdM using keytabs, and they authenticate to the vault using a private key. You can retrieve the file on behalf of the service from any IdM client on which **ansible-freeipa** is installed.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.
  - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the IdM administrator password.
- You have created an asymmetric vault to store the service secret.
- You have archived the secret in the vault .
- You have stored the private key used to retrieve the service vault secret in the location specified by the **private_key_file** variable on the Ansible controller.

Procedure

1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Optional: Create an inventory file if it does not exist, for example **inventory.file**:

```
$ touch inventory.file
```

3. Open your inventory file and define the following hosts:
   - Define your IdM server in the **[ipaserver]** section.
   - Define the hosts onto which you want to retrieve the secret in the **[webservers]** section. For example, to instruct Ansible to retrieve the secret to **webserver1.idm.example.com**, **webserver2.idm.example.com**, and **webserver3.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com

[webservers]
webserver1.idm.example.com
webserver2.idm.example.com
webserver3.idm.example.com
```

4. Make a copy of the **retrieve-data-asymmetric-vault.yml** Ansible playbook file. For example:

```
$ cp retrieve-data-asymmetric-vault.yml retrieve-data-asymmetric-vault-copy.yml
```

5. Open the **retrieve-data-asymmetric-vault-copy.yml** file for editing.
6. Modify the file by setting the following variables in the **ipavault** task section:
   - Set the **ipaadmin_password** variable to your IdM administrator password.
   - Set the **name** variable to the name of the vault, for example **secret_vault**.
   - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
   - Set the **private_key_file** variable to the location of the private key used to retrieve the service vault secret.
   - Set the **out** variable to the location on the IdM server where you want to retrieve the **private-key-to-an-externally-signed-certificate.pem** secret, for example the current working directory.
   - Set the **action** variable to **member**.
     This the modified Ansible playbook file for the current example:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
```

```
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Retrieve data from the service vault
    ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      vault_type: asymmetric
      private_key: "{{ lookup('file', 'service-private.pem') | b64encode }}"
      out: private-key-to-an-externally-signed-certificate.pem
      state: retrieved
```

7. Add a section to the playbook that retrieves the data file from the IdM server to the Ansible controller:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
  gather_facts: false
  tasks:
[...]
  - name: Retrieve data file
    fetch:
      src: private-key-to-an-externally-signed-certificate.pem
      dest: ./
      flat: true
      mode: 0600
```

8. Add a section to the playbook that transfers the retrieved **private-key-to-an-externally-signed-certificate.pem** file from the Ansible controller on to the webservers listed in the **webservers** section of the inventory file:

```
---
- name: Send data file to webservers
  become: no
  gather_facts: no
  hosts: webservers
  tasks:
  - name: Send data to webservers
    copy:
      src: private-key-to-an-externally-signed-certificate.pem
      dest: /etc/pki/tls/private/httpd.key
      mode: 0444
```

9. Save the file.

10. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file retrieve-data-asymmetric-vault-copy.yml
```

5.5. CHANGING AN IDM SERVICE VAULT SECRET WHEN COMPROMISED USING ANSIBLE

Follow this procedure to reuse an Ansible playbook to change the secret stored in a service vault when a service instance has been compromised. The scenario in the following example assumes that on **webserver3.idm.example.com**, the retrieved secret has been compromised, but not the key to the asymmetric vault storing the secret. In the example, the administrator reuses the Ansible playbooks used when storing a secret in an asymmetric vault and retrieving a secret from the asymmetric vault onto IdM hosts. At the start of the procedure, the IdM administrator stores a new **PEM** file with a new secret in the asymmetric vault, adapts the inventory file so as not to retrieve the new secret on to the compromised web server, **webserver3.idm.example.com**, and then re-runs the two procedures.

Prerequisites

- You have configured your Ansible control node to meet the following requirements:
  - You are using Ansible version 2.14 or later.
  - You have installed the **ansible-freeipa** package on the Ansible controller.
  - The example assumes that in the ***~/MyPlaybooks/*** directory, you have created an Ansible inventory file with the fully-qualified domain name (FQDN) of the IdM server.
  - The example assumes that the **secret.yml** Ansible vault stores your **ipaadmin_password**.
- The target node, that is the node on which the **ansible-freeipa** module is executed, is part of the IdM domain as an IdM client, server or replica.
- You know the **IdM administrator** password.
- You have created an asymmetric vault to store the service secret.
- You have generated a new **httpd** key for the web services running on IdM hosts to replace the compromised old key.
- The new **httpd** key is stored locally on the Ansible controller, for example in the **/usr/share/doc/ansible-freeipa/playbooks/vault/private-key-to-an-externally-signed-certificate.pem** file.

Procedure

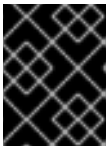
1. Navigate to the **/usr/share/doc/ansible-freeipa/playbooks/vault** directory:

```
$ cd /usr/share/doc/ansible-freeipa/playbooks/vault
```

2. Open your inventory file and make sure that the following hosts are defined correctly:
   - The IdM server in the **[ipaserver]** section.
   - The hosts onto which you want to retrieve the secret in the **[webservers]** section. For example, to instruct Ansible to retrieve the secret to **webserver1.idm.example.com** and **webserver2.idm.example.com**, enter:

```
[ipaserver]
server.idm.example.com
```

```
[webservers]
webserver1.idm.example.com
webserver2.idm.example.com
```



IMPORTANT

Make sure that the list does not contain the compromised webserver, in the current example **webserver3.idm.example.com**.

3. Open the **data-archive-in-asymmetric-vault-copy.yml** file for editing.
4. Modify the file by setting the following variables in the **ipavault** task section:
   - Set the **ipaadmin_password** variable to the IdM administrator password.
   - Set the **name** variable to the name of the vault, for example **secret_vault**.
   - Set the **service** variable to the service owner of the vault, for example **HTTP/webserver.idm.example.com**.
   - Set the **in** variable to **"{{ lookup('file', 'new-private-key-to-an-externally-signed-certificate.pem') | b64encode }}"**. This ensures that Ansible retrieves the file with the private key from the working directory on the Ansible controller rather than from the IdM server.
   - Set the **action** variable to **member**.
     This the modified Ansible playbook file for the current example:

```
---
- name: Tests
  hosts: ipaserver
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver.idm.example.com
      in: "{{ lookup('file', 'new-private-key-to-an-externally-signed-certificate.pem') | b64encode
}}"
      action: member
```

5. Save the file.
6. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file data-
archive-in-asymmetric-vault-copy.yml
```

7. Open the **retrieve-data-asymmetric-vault-copy.yml** file for editing.
8. Modify the file by setting the following variables in the **ipavault** task section:
   - Set the **ipaadmin_password** variable to your IdM administrator password.

- Set the **ipaadmin_password** variable to your IdM administrator password.
- Set the **name** variable to the name of the vault, for example **secret_vault**.
- Set the **service** variable to the service owner of the vault, for example **HTTP/webserver1.idm.example.com**.
- Set the **private_key_file** variable to the location of the private key used to retrieve the service vault secret.
- Set the **out** variable to the location on the IdM server where you want to retrieve the **new-private-key-to-an-externally-signed-certificate.pem** secret, for example the current working directory.
- Set the **action** variable to **member**.
  This the modified Ansible playbook file for the current example:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: no
  gather_facts: false

  vars_files:
  - /home/user_name/MyPlaybooks/secret.yml
  tasks:
  - name: Retrieve data from the service vault
    ipavault:
      ipaadmin_password: "{{ ipaadmin_password }}"
      name: secret_vault
      service: HTTP/webserver1.idm.example.com
      vault_type: asymmetric
      private_key: "{{ lookup('file', 'service-private.pem') | b64encode }}"
      out: new-private-key-to-an-externally-signed-certificate.pem
      state: retrieved
```

9. Add a section to the playbook that retrieves the data file from the IdM server to the Ansible controller:

```
---
- name: Retrieve data from vault
  hosts: ipaserver
  become: true
  gather_facts: false
  tasks:
[...]
  - name: Retrieve data file
    fetch:
      src: new-private-key-to-an-externally-signed-certificate.pem
      dest: ./
      flat: true
      mode: 0600
```

10. Add a section to the playbook that transfers the retrieved **new-private-key-to-an-externally-signed-certificate.pem** file from the Ansible controller on to the webservers listed in the **webservers** section of the inventory file:

```
---
- name: Send data file to webservers
  become: true
  gather_facts: no
  hosts: webservers
  tasks:
  - name: Send data to webservers
    copy:
      src: new-private-key-to-an-externally-signed-certificate.pem
      dest: /etc/pki/tls/private/httpd.key
      mode: 0444
```

11. Save the file.
12. Run the playbook:

```
$ ansible-playbook --vault-password-file=password_file -v -i inventory.file retrieve-data-asymmetric-vault-copy.yml
```

5.6. ADDITIONAL RESOURCES

- See the README-vault.md Markdown file in the **/usr/share/doc/ansible-freeipa/** directory.
- See the sample playbooks in the **/usr/share/doc/ansible-freeipa/playbooks/vault/** directory.