Securing non-interactive NPA's in JBoss EAP 6.4

Contents

Introduction

JBoss password vault

Configuration of JBoss EAP and associated applications requires potentially sensitive information, such as usernames and passwords. Instead of storing this sensitive information as plain text in configuration files, use the Password Vault feature to mask this information and store it in an encrypted keystore.

Instead of storing the password as plain text in configuration files, you can use the Password Vault feature to mask the password information and store it in an encrypted keystore. Once the password is stored, you can include references in Management CLI commands or your own applications. The Password Vault uses the Java Keystore as its storage mechanism. Password Vault consists of two parts: storage and key storage. Java Keystore is used to store the key, which is used to encrypt or decrypt sensitive strings in Vault storage.

JBoss documentation:
https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Security_Guide/chap-Secure_Passwords_and_Other_Sensitiv

JBoss provides a tool for creating and maintaining the secret vault. The tool is shipped together with the JBoss application server and can be found in the JBoss bin directory: <JBoss home>/bin/vault.sh

It is not secure! But why it is still recommended?

We should clearly understand, that the eventual level of security provided by JBoss vault is absolutely the same as in case of plain text passwords. First we will consider why (because it may be not obvious). Then we will see why in spite of this there are some benefits.

- What is the purpose of the JBoss vault?
  - The purpose is to prevent easy revealing of passwords used in JBoss configuration files. Actually not only passwords, but any sensitive data
- Why is it not secure?
  - JBoss needs to be able to restore passwords without user input. If JBoss asked user for password each time it starts, it would be secure, but not maintainable. It would be impossible to guarantie high availability of servers
  - It means JBoss should be able to restore passwords based only on resources available on server, i.e. vault and key store
  - If somebody obtains access to these files on server, he can easily restore encrypted passwords, even without knowing any secret information
  - Don't believe? See decryptor in the attachment. Start the application and provide the data that you used in the <vault> section of JBoss config
- But what's the reason to use it then?
  - Without vault
    - Configuration of your application changes time to time. Sometimes you wish to restore the state that was "10 changes before" the current. That's why you wish to keep track of all changes. Where do you keep all previous versions? In an archive with a secret password? Or in some secret repository? Not bad
    - Then what you do if developers request your configs to analyze some problem? Do you thoroughly check the configs and remove all passwords before sending files to devs? Have you replaced all passwords? In all files you send? Ahh, OK, you use scripts. But are you sure you have adjusted the scripts after one more datasource was added last month? OK, you've done it
    - Are you sure it was on this environment, for this staging server? Not different config structure for another branch or stage?
    - Are you sure each time? Of course this can work. But what is the price? How much headake do you have each time?
  - With vault
    - All passwords in your configs are encrypted
    - You replace the original passwords in configs only once and dont't care any more
    - You can share the configs with devs
    - You can keep all versions of configs in a normal non-secret repository (svn, git). Many people can access them (your devs, ops, admins, customer, your repo hoster like github or bitbucket) without compromising security
    - You "disclose" your configs AS IS. No need to remove passwords. No need to warry each time if you have have overlooked any sensitive info
    - You can automate backing up of all configs without compromising security
    - The only thing you should care about is the key store. You should keep it apart from your configs. But it is much easier. It is just a single file
  - Idea
    - Basically we split the secret. One part is stored in config file. It is public. Another part is the key store. Keep it secretly. Of course, everyone who has access to the key store on your server can restore all the encrypted *** secrets. But you avoid the headake with maintenance of your configs

Creating the secret vault

Create a java keystore

The base of the secret vault is a standard java kestore. So first we have to create a keystore that we will use later to initialize the secret vault.

Warning
You must generate the keystore using the keytool utility from the same vendor as the JDK you use to run JBoss as Java keystore implementations might differ.

The secret vault implementation in JBoss EAP 6.4 supports just the jceks key type and the 128 bit key length. If you use a different type or longer key, the secret vault initialization will fail. Lets create the new keystore inside JBoss home directory. The alias can be any string but you have to provide it for the secret vault management.

```
/usr/lib/jvm/jdk1.8.0_45/bin/keytool -genseckey -alias vault -storetype jceks -keyalg AES -keysize 128 -storepass 123456 -keypass 123456 -val
```

Once the keystore is created we can list the content of it with the keytool --list command. It is important to provide the type here, otherwise the keytool will be unable to read the keystore file.

```
$ /usr/lib/jvm/jdk1.8.0_45/bin/keytool --list -storetype jceks -keystore /home/adam/runable/jboss/EAP-6.4.0_second/vault/vault.keystore
Enter keystore password:
Keystore type: JCEKS
Keystore provider: SunJCE
Your keystore contains 1 entry
vault, Oct 27, 2016, SecretKeyEntry,
As we can see, the keystore contains only one entry with the alias we provided when we created the store.
```

Initialize the secret vault

We have to cd the jboss bin directory and run the vault.sh tool. This tool has an interactive command line and can be run non-interactively. If we start it without any parameter it will start in interactive mode, which is required for initializing the secret vault.

After starting the vault.sh tool, you have to press 0, for starting the console. Then you have to provide the following information:

- Keystore URL: provide the full path to the previously created keystore
- Keystore password: the master password of the keytore that we provided previously in the --storepass parameter.
- Enter 8 character salt: Here you have to come up with an 8 character long random data that is used as an additional input to the one-way function that "hashes" the passwords and the sensitive date. It can be anything you want to. You have to remember it, it will be required for maintaining the secret vault, and it has to be provided in the domain.xml file.
- Enter iteration count as a number: That controls how many times the hash function should be applied. You can chose anything you have to remember it, it will be required for maintaining the secret vault, and it has to be provided in the domain.xml file.
- Enter Keystore Alias: It is the alias of the keytore that we created previously. It has to be the same string.

```
$ cd /home/adam/runable/jboss/EAP-6.4.0_second/bin
$ ./vault.sh
=========================================================================
  JBoss Vault
  JBOSS_HOME: /home/adam/runable/jboss/EAP-6.4.0_second
  JAVA: /usr/lib/jvm/jdk1.8.0_45//bin/java
=========================================================================
**********************************
****  JBoss Vault  ***************
**********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:/home/adam/runable/jboss/EAP-6.4.0_second/vault/
Enter Keystore URL:/home/adam/runable/jboss/EAP-6.4.0_second/vault/vault.keystore
Enter Keystore password: 123456
Enter Keystore password again: 123456
Values match
Enter 8 character salt:12345678
Enter iteration count as a number (Eg: 44):44
Enter Keystore Alias:vault
Initializing Vault
Oct 27, 2016 3:21:11 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in configuration file:
********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="/home/adam/runable/jboss/EAP-6.4.0_second/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-11/Uou6siko"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="12345678"/>
  <vault-option name="ITERATION_COUNT" value="44"/>
  <vault-option name="ENC_FILE_DIR" value="/home/adam/runable/jboss/EAP-6.4.0_second/vault/"/>
</vault><management> ...
********************************************
Vault is initialized and ready for use
Handshake with Vault complete
```

At the and, the console will print the required configuration section that we have to insert in the domain.xml.

Note
The keystore password is masked, but the password can be easily retrieved if we learn the method how it was created.

Open the domain.xml and insert the <vault>...</vault> section after the system properties but before the management section. If there are no system properties just insert it between the extension and the management section.

```
<server xmlns="urn:jboss:domain:1.7">
    <extensions>
        .....
    </extensions>
    <system-properties>
       ....
    </system-properties>
    <vault>
          ....
    </vault>
    <management>
          .....
```

Restart jboss. Now the secret vault is ready to use.

Insert password/sensitive string into the vault

We will use the non interactive mode of the vault.sh tool to insert sensitive data into the secret vault. The following properties has to be provided:

- keystore: The full path of the java keytore that we created before
- keystore-password: the passowerd of the java keystore that we created before
- alias: The name of the java keystore. In our case: vault
- vault-block: We can group our passowrds/sensitive data in so called "vault-blocks". We can come up with any name e.g. databasePasswords, tibcoRelated etc and place our passwords always in the proper group (vault-block). When we provied the referer key in the configuration file the name of the vault-block has to be provieded where the encripted passoword of the key is.
- attribute: This is the name of the key that will replace the sensitive information in the configuration files.
- sec-attr: The plain text format of the sensitive data/password that we would like to hide.
- enc-dir: The same directory must be given as in the previous secton when we created the secret vault.
- iteration: The iteration number must be the same as the iteration number we for creating the secret vault.
- salt: also the same salt has to be used as the salt we used for creating the vault

In the following example we will insert "password12345" sensitive string into the secret vault. The name of the key is going to be "tibco_password". That is the key that will replace the original password in the config file.

```
$ ./vault.sh --keystore /home/adam/runable/jboss/EAP-6.4.0_second/vault/vault.keystore --keystore-password 123456 --alias vault --vault-block
=====================================https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Security_G
  JBoss Vault
  JBOSS_HOME: /home/adam/runable/jboss/EAP-6.4.0_second
  JAVA: /usr/lib/jvm/jdk1.8.0_45//bin/java
=========================================================================
Oct 28, 2016 1:30:30 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Secured attribute value has been stored in vault.
Please make note of the following:
********************************************
Vault Block:myprojectVaultBlock
Attribute Name:tibco_password
Configuration should be done as follows:
VAULT::myprojectVaultBlock::tibco_password::1
********************************************
```

The replacement string will be printed out on the screen: **VAULT::myprojectVaultBlock::tibco_password::1**

It contains three sections separated with "::". It starts with the VAULT keyword to tell to JBoss that this this key has to be resolved from the secret vault. The second part is the name of the vault-block, and the last part is the key itself.

Replacing sensitive strings

Replace sensitive strings in the jboss configuration files

Almost every subsystem supports the usage of secret vault. You just have to replace the original string with the new replacement string provided by the vault.sh tool surrounded by ${...}. According to the example above the configuration section of the tibco external-context would be the following:

```
        <subsystem xmlns="urn:jboss:domain:naming:1.4">
            <bindings>
                <external-context name="java:global/remoteTibcoContext/" module="org.jboss.genericjms.provider" class="javax.naming.InitialCo
                    <environment>
                        .....
                        <property name="java.naming.security.credentials" value="${VAULT::myprojectVaultBlock::tibco_password::1}"/>
                    </environment>
                </external-context>
            </bindings>
            <remote-naming/>
        </subsystem>
```

Resolving passwords in java code

In the java code the secret vault can be accesses with the help of the org.jboss.security.vault.SecurityVaultUtil class. It can be found in the jboss-security-spi.jar. We can get it with the following Maven dependency:

```
    <dependency>
            <groupId>org.picketbox</groupId>
            <artifactId>jboss-security-spi</artifactId>
            <version>4.9.7.Final</version>
        </dependency>
```

In the java code the secret data can be resolved in the following way:

```
import org.jboss.security.vault.SecurityVaultUtil;
....
String pwd = "";

         try {
                pwd = SecurityVaultUtil.getValueAsString("VAULT::myprojectVaultBlock::tibco_password::1");
            } catch (SecurityVaultException e) {

            }
```