Q2 2023 Tech Training - Automating Code Signing with Enterprise Pipelines

This guide is specifically written to be used within the Venafi AWS CloudFormation POC/Demo environment. your lab environment has been provisioned specifically for your use and is not shared with other people/organizations.

Your environment is here:

TPP instance: `tt23-<username>-tpp.se.venafi.com`
DevOps/CodeSigning Ubuntu instance: `tt23-<username>-devopslab.se.venafi.com`
Windows instance: `tt23-<username>-vsatworker.se.venafi.com`

For signing on the DevOps/CodeSigning Ubuntu and the VSAT Windows worker instance we will use the following account:

User: `venafilab`
Password: `Ven@fiPassw0rd`

This guide walks through automating code signing as part of several Jenkins pipelines using a CodeSign Protect Certificate environment. We'll walk through specific use cases around signing Jar files, a Windows executable, and finally sign a container image using the Sigstore cosign tool.

This guide assumes you have a working knowledge of Docker and Jenkins.

Jenkins is already installed and configured on the DevOps/CodeSigning Ubuntu instance with the same credentials as above.

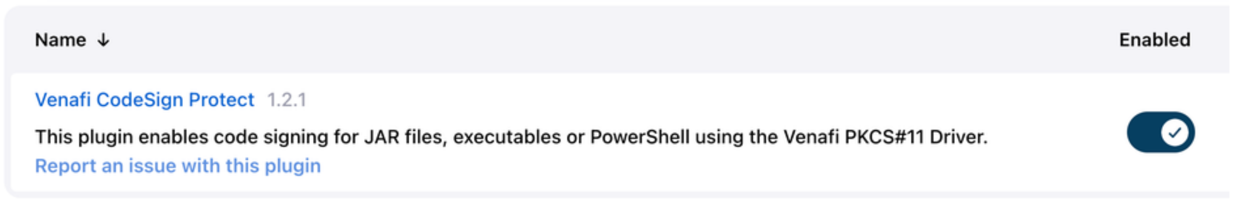
Please use the following credentials for the TPP platform:

User: `administrator`
Password: `Ven@fiPassw0rd`

Instructions

Jenkins Configuration

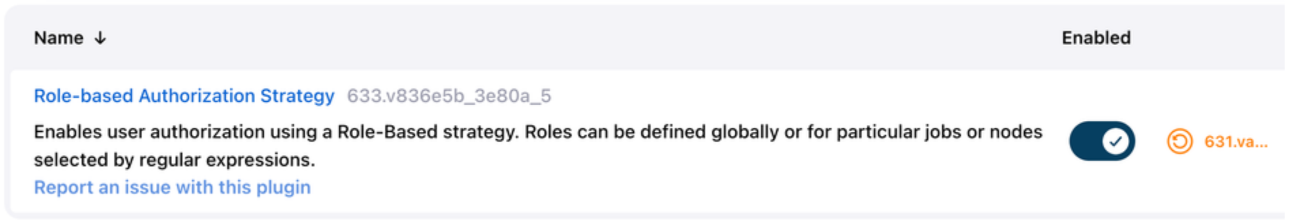
1. Launch a browser tab and connect to `https://tt23-<username>-devopslab.se.venafi.com:8002`. Ignore any certificate errors/warnings since this is a demo environment.
2. Install the Venafi CodeSign Protect plugin via `Dashboard` → `Manage Jenkins` -> `Plugin Manager` -> `Available Plugins` → `Venafi CodeSign Protect`. Click on `Download now and install after restart`. You may also have to check the `Restart Jenkins when installation is complete and no jobs are running` option.



Name ↓	Enabled
Venafi CodeSign Protect 1.2.1 This plugin enables code signing for JAR files, executables or PowerShell using the Venafi PKCS#11 Driver. Report an issue with this plugin	✓

3. Configure `Venafi CodeSign Protect` plugin via `Manage Jenkins` → `Configure System`
   a. Go to `Venafi Code Signing` section, then click on `Add` button under Venafi TPPs:
      i. Enter TPP Name → `tpp`
      ii. Enter Authorization URL → `https://tt23-<username>-tpp.se.venafi.com/vedauth`
      iii. Enter HSM URL → `https://tt23-<username>-tpp.se.venafi.com/vedhsm`
      iv. Click on the `Save` button at the bottom.
4. Install the `Role Strategy` plugin via `Manage Jenkins` → `Plugin Manager` → `Available Plugins` → `Role-based Authorization Strategy`. Click on `Download now and install after restart` . You may also have to check the `Restart Jenkins when`

`installation is complete and no jobs are running` option.


Name ↓ | Enabled

Role-based Authorization Strategy 633.v836e5b_3e80a_5

Enables user authorization using a Role-Based strategy. Roles can be defined globally or for particular jobs or nodes selected by regular expressions.

Report an issue with this plugin

631.va...

5. Create Folder via `Dashboard` → `New Item` → `Folder`
   a. Provide a name for the folder → `App Team 1`
   b. Optionally enter `Display Name` such as `App Team 1` and then Save
6. Create a domain via The `App Team 1` folder
   a. Click on Credentials
   b. Click on Stores scoped to `App Team 1`
   c. Click on `Add Domain` and enter a sample domain such as `venafilab.com`, then Save
7. Create a Username/Password credential to represent the user account that will be signing
   a. Click on the domain that you just created
   b. Click on `+ Add Credentials`
   c. Make sure Kind is `Username and password`
   d. Username = `sample-cs-user`
   e. Password = `NewPassw0rd!`

   *Note the **ID**, as we will need this when creating the groovy pipeline*

Install Jenkins Agent on Ubuntu:

Let's first start by setting up the Jenkins agent in your environment. This will be used to sign a sample Jar file.

1. Browse to `https://tt23-<username>-devopslab.se.venafi.com:8002/` and authenticate using the credential described above
2. Go To `Dashboard` -> `Manage Jenkins` -> `Manage Nodes and Clouds` and select `New Node`
3. Give it a name: `linux_node` and set it as a Permanent Agent
4. Set `Remote root directory` to `/home/venafilab` Set `Labels` to `linux_node` and Save
5. On the `Nodes` page select the agent you just created
6. SSH into the DevOps/Ubuntu instance using the credential provided above. Follow the steps outlined in the `Run from agent command line`, with some slight modifications such as adding the `-noCertificateCheck` argument to the `java` command line. Here is an example:

```
curl -sO -k https://tt23-<username>-devopslab.se.venafi.com:8002/jnlpJars/agent.jar
java -jar agent.jar -noCertificateCheck -jnlpUrl https://tt23-<username>-devopslab.se.venafi.com:8002/manage/comp
```

✓ May 19, 2023 6:19:05 AM hudson.remoting.jnlp.Main$CuiListener status

INFO: Remote identity confirmed: 74:5f:ad:f1:08:ff:4f:4e:f9:9e:b2:77:a6:99:71:09

May 19, 2023 6:19:05 AM hudson.remoting.jnlp.Main$CuiListener status

INFO: Connected

Install Jenkins Agent on Windows:

Since we will be signing a sample executable we'll need to have a Windows environment for the Jenkins pipeline to execute the `signtool` command:

1. Via the Jenkins Web Admin console Go To `Dashboard` -> `Manage Jenkins` -> `Nodes` and select `New Node`
2. Give it a name: `windows_node` and set it as a Permanent Agent
3. Set `Remote root directory` to `c:\temp` and Set `Labels` to `windows_node` and Save
4. On the `Nodes` page select the agent you just created
5. RDP into the VSAT Windows worker instance (`tt23-<username>-vsatworker.se.venafi.com`) using the credential provided above. Follow the steps outlined in the `Run from agent command line`, with some slight modifications such as adding the `-noCertificateCheck` argument to the `java` command line. Here is an example:

```
curl -sO -k https://tt23-<username>-devopslab.se.venafi.com:8002/jnlpJars/agent.jar
java -jar agent.jar -noCertificateCheck -jnlpUrl https://<customer>-devopslab.se.venafi.com:8002/manage/computer/
```

✓ May 19, 2023 6:19:05 AM hudson.remoting.jnlp.Main$CuiListener status

INFO: Remote identity confirmed: 74:5f:ad:f1:08:ff:4f:4e:f9:9e:b2:77:a6:99:71:09

May 19, 2023 6:19:05 AM hudson.remoting.jnlp.Main$CuiListener status

INFO: Connected

Install Venafi CodeSign Protect Clients on Jenkins Agent Systems:

Now install the Venafi CodeSign Protect client for Ubuntu using the following commands. Make sure to replace the <customer> with your assigned POC `customer` name.

```
curl -o "venafi-codesigningclients-23.1.0-linux-x86_64.deb" https://tt23-<username>-tpp.se.venafi.com/csc/clients
sudo dpkg -i "venafi-codesigningclients-23.1.0-linux-x86_64.deb"
```

Install the Venafi CodeSign Protect client for Windows if it is not already available on the Windows node:
Make sure to run with elevated privileges (e.g. `run as administrator`)

```
curl -o "VenafiCodeSigningClients-23.1.0-x64.msi" ^https://tt23-<username>-tpp.se.venafi.com/csc/clients/venafi-c
start /wait msiexec /qn /i "VenafiCodeSigningClients-23.1.0-x64.msi"
```

Configure Jenkins Pipeline for Windows Signing:

Now that we have the Venafi CodeSign Protect client installed as well as the Jenkins agent up and running, let's configure a Jenkins pipeline to leverage signtool to sign a sample executable.

1. Within the `Folder` you created earlier, Select `+ New Item`
2. Provide the `item name` such as `signtool-test` and select `Pipeline`
3. Enter the pipeline script as follows:

```
node {
    agent 'windows_node'
    venafiCodeSignWithSignTool tppName: 'tpp',
    fileOrGlob: 'c:\\temp\\sample.exe',
    subjectName: 'Sample Code Signers Are Us, LLC',
    credential: [credentialsId: '<credential_id_goes_here>'],
    timestampingServers: [[address: 'http://timestamp.digicert.com']]
}
```

4. Run the job by clicking on `Build Now`
5. Assuming a successful run your executable should now be signed. You can view the output of the run by clicking on the job # in the `Build History`.

Configure Jenkins Pipeline for Jar Signing

Now that we have the Venafi CodeSign Protect client installed as well as the Jenkins agent up and running, let's configure a Jenkins pipeline to leverage jarsigner to sign a sample jar file.

1. Within the `Folder` you created earlier, Select `+ New Item`
2. Provide the `item name` such as `jarsigner-test` and select `Pipeline`
3. Enter the pipeline script as follows:

```
pipeline {
    agent { label 'windows_node' }
    stages {
        stage("Venafi") {
            steps {
                venafiCodeSignWithJarSigner tppName: 'tpp',
                file: 'c:\\temp\\test.jar',
                certLabel: 'Sample-Development-Environment',
                credential: [credentialsId: '<credential_id_goes_here>'],
                timestampingServers: [[address: 'http://timestamp.digicert.com']]
            }

        }
    }
}
```

4. Run the job by clicking on `Build Now`
5. Assuming a successful run your jar file should now be signed. You can view the output of the run by clicking on the job # in the `Build History`.

Configure Jenkins Pipeline for Signing a Container Image

In this part of the lab we will now seup a local stateless Docker-based registry so that we can store as well as distribute signed and unsigned container images that we've obtained from Docker hub. From there we'll run the cosign tool to sign and publish the signed container image tag.

1. Within the `Folder` you created earlier, Select the `+ New Item`
2. Provide the `item name` such as `cosign-test` and select `Pipeline`
3. Enter the pipeline script as follows (make sure to modify URLs specific to your environment):

```
pipeline {
    agent { label 'linux_node' }
    environment {
        CSP_HSM_URL = 'https://tt23-<username>-tpp.se.venafi.com/vedhsm'
        CSP_AUTH_URL = 'https://tt23-<username>-tpp.se.venafi.com/vedauth'
    }
    stages {
        stage("Venafi") {
            steps {
                withCredentials([usernamePassword(credentialsId: '<credential_id_goes_here>',
                                                  usernameVariable: "USERNAME", passwordVariable: "PASSWORD")
                    sh '/opt/venafi/codesign/bin/pkcs11config getgrant --force --authurl=$CSP_AUTH_URL --hsmu
                    sh 'docker run -d -p 5000:5000 --name registry registry:2'
                    sh 'docker pull alpine:latest'
                    sh 'docker image tag alpine:latest localhost:5000/alpine:signed'
                    sh 'docker image push localhost:5000/alpine:signed'
                    sh 'cosign sign --tlog-upload=false --key "pkcs11:slot-id=0;object=Sample-Development-Env
                    sh 'docker container rm registry --force'
                }
            }

        }
    }
}
```

4. Run the job by clicking on `Build Now`
5. Assuming a successful run your container image should be signed. You can view the output of the run by clicking on the job # in the `Build History`