Preparing Certificates for a PCE Deployment

Article Owner Russell Goodwin

Customer Viewable ✓

Customer URL https://my.illumio.com/apex/article?name=Preparing-Certificates-for-an-Illumio-MNC-Installation

Summary

Introduction

This article details the prerequisites and processes required to prepare your security certificates and to validate that they are ready prior to installing the Policy Compute Engine (PCE) software.

Overview

Illumio software deployments (single and multi-node clusters) use transport layer security (TLS) extensively to secure communication flows both between cluster members and between the PCE components and the Virtual Enforcement Nodes (VENs). As a result, the certificates used in the installation provide the public/private key pairs that provide this protection. This article guides you through the process of preparing and validating these components.

Creating the Certificates

Certificates are used for 3 major components in the PCE software installation that use TLS:

- Web Service – Used to secure access to the PCE web console, as well as that provided by the Illumio ASP REST API.
- Event Service – Provides continuous, secure connectivity from the PCE to the VENs under management and provides a notification capability so that VENs can be instructed to update policy on the host Workloads.
- Service Discovery – Used for cluster management between PCE node members in the cluster and allows real time alerting on service availability and status.

A common certificate can be used for all these functions, but it is important that all the right options are present in the certificate to allow for secure communication of the software.

Creating the Web Event Service and Certificates

Each customer deployment environment will likely have different processes for requesting and signing certificates. Sometimes, there is a self-help portal where keys and certificates can be generated. Other times, the key generation needs to be completed on the server and then a Certificate Signing Request (CSR) posted to the portal for signing. In some cases, another process may exist to send a CSR for signing. In this example, we will generate a key on the PCE node and then create a Signing Request to be sent in some form to the CA. All keys in this example will be RSA-based keys, using 2048 bit primes.

Note: *The keys and certificates do not actually need to be created on the target server since the keys are not tied to the host. You may create a key and certificate pair for one of the PCE nodes and then copy to the others.*

1. Create the Private Key using OpenSSL:

   ```
   [user@core0.pce certs]# openssl genrsa -out ./pce.test.com.key 2048 -nodes
   ```

   This will produce a file of the name `pce.test.com.key` in the local directory. The '`-nodes`' switch makes sure no password is applied. The key needs to be created without password protection. You can validate if a key is password protected as follows:

   ```
   [user@core0.pce certs]# openssl rsa -noout -text -in ./pce.test.com.key
   ```

   If you are prompted for a password, then the key is protected. This password will need to be removed, which can be achieved with the command below. If the command just provides the details to you on the modulus, primes and exponents, etc., then you are ready to move on to Step 2.

   ```
   [user@core0.pce certs]# openssl rsa -in ./pce.test.com.key -out ./pce.test.com.key
   ```

   This command takes the key as input, prompting for the password. It then writes the key out without any password protection.

2. Next, use the key to generate a Certificate Signing Request (CSR). The signed certificates require that some x509 Extended KeyUsage attributes are included. There are two options: you can create the CSR without these and request that the CA add these attributes; or you can embed them in the request, if the CA will accept them.

   a. Simple CSR without x509 attributes

   ```
   [user@core0.pce certs]# openssl req -key ./pce.test.com.key -new -sha256 -out ./pce.test.com.csr
   ```

   You will be prompted to complete information about the certificate request. The Common name should match your DNS name or DNS Alias for the PCE. The challenge password is optional and should be left blank.

   ```
   Country Name (2 letter code) [XX]:GB
   State or Province Name (full name) []:England
   Locality Name (eg, city) [Default City]:London
   Organization Name (eg, company) [Default Company Ltd]:test.com
   Organizational Unit Name (eg, section) []:lab
   Common Name (eg, your name or your server's hostname) []:pce.test.com
   Email Address []:user1@company.com

   Please enter the following 'extra' attributes
   to be sent with your certificate request
   A challenge password []:
   An optional company name []:
   [root@core0.pce.test.com certs]#
   ```

This CSR file should be either sent to the CA, or the contents may be cut and paste into a self-help portal created for requesting certificates. You need to specify when requesting the signing that:

- TLS Web Server Authentication Extended Key Usage is included
- TLS Web Client Authentication Extended Key Usage is included
- Optional : Subject Alternative Names (SAN) are included for the PCE cluster and Core node names

If you are cutting and pasting these into a portal, it is normal to submit these as base64 encoded. You can simply 'cat' the file to get this output, it will be a long string of characters with `-----BEGIN CERTIFICATE-----` and `----END CERTIFICATE----` bookends.

Note: *On a Windows CA, the 'domain controller' profile includes both the Server and Client key usage attribute so is suitable for signing the CSR.*

b. Imbedding the x509 Attributes

On the PCE cluster, create a small openssl.cnf configuration file based on the example contents.

```
[ req ]
prompt = no
distinguished_name = DN

[ DN ]
commonName = pce.test.com
countryName = GB
localityName = London
organizationName = lab
emailAddress = russell@illumio.com

[ extended ]
extendedKeyUsage = serverAuth, clientAuth
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = pce.test.com
DNS.2 = pce-core0.test.com
DNS.3 = pce-core1.test.com
```

The SAN is optional, but allows a browser to navigate directly to one Core node or the other without a certificate error. Depending on your network topology, this can be a useful test aid. Note, however, that many browsers will ignore the CN field if the SAN exists, so it is important to include the cluster name in the SAN also or attempts to reach the cluster will result in certificate validation errors.

Make the CSR and specify the config file and the extensions from the config to use.

```
[user@core0.pce certs]# openssl req -reqexts extended -config ./openssl.cnf -key ./pce.test.com.key -new -sha256 -out ./pce.test.com.csr
```

You can verify the request as follows.

```
[user@core0.pce certs]# openssl req -in ./pce.test.com.csr -noout -text

Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=pce.test.com, C=GB, L=London, O=lab/emailAddress=russell@illumio.com
        Subject Public Key Info:
//// content abbreviated ////
        Attributes:
        Requested Extensions:
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                DNS:pce.test.com, DNS:pce-core0.test.com, DNS:pce-core1.test.com
```

You can then send this request to your CA for signing. Note that your CA must be configured to accept these X509 Extended Key Usage attributes or they will be ignored. Contact your PKI administrator for more information on what is permitted.

3. Validating the Certificate.

You should receive a certificate file, which you will want in PEM format. If you look at the file, it should look something like this:

```
[user@core0.pce certs]# cat ./pce.test.com.crt
-----BEGIN CERTIFICATE-----
MIIF4zCCA8ugAwIBAgICEAAwDQYJKoZIhvcNAQELBQAwgYoxCzAJBgNVBAYTAkdC
MRAwDgYDVQQIDAdFbmdsYW5kMREwDwYDVQQKDAh0ZXN0LmNvbTEMMAoGA1UECwwD
bGFiMSQwIgYDVQQDDBtpbnRlcm1lZGlhdGUubGFiY2EudGVzdC5jb20xIjAgBgkq
hkiG9w0BCQEWE3J1c3NlbGxAaWxsdW1pby5jb20wHhcNMTUwOTIyMDcxNjQ4WhcN
MTcwOTIxMDcxNjQ4WjB7MQswCQYDVQQGEwJHQjEQMA4GA1UECAwHRW5nbGFuZDER
MA8GA1UECgwIdGVzdC5jb20xDDAKBgNVBAsMA2xhYjEVMBMGA1UEAwwMcGNlLnRl
c3QuY29tMSIwIAYJKoZIhvcNAQkBFhNydXNzZWxsQGlsbHVtaW8uY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA37yBXGWlgkVDm6PHLEt4OM94NTcz
BxDfDSURuvwZkb955ygcl9lPd66PoGSetETxMHW2mSp1UZ/+cUVOwUE0uhIrABk+
C2+gYzk3MYAlfZN/ziKIi08yABmizABjUSl7RiYacacm5iKe8y7xzA1tYjBMapVR
/eGC5mXuMRGlIrzCWGCu6vlVLwHKNZIw2KXxkT8cJhvyUeJh6QlztSKZKS3OetyC
K8op6fAK8lrz1IZcShYGlkj690fvN60DVUs6js2RmnZ7svjVZE+O4TpYfrEFBdPs
9dqAysmAXNXGOVAK5KccHTDYUhD6Ot0VY5oSQeJQ2Nf2x/yJVadzkZpdxQIDAQAB
o4IBXzCCAVswCQYDVR0TBAIwADARBglghkgBhvhCAQEEBAMCBkAwMwYJYIZIAYb4
QgENBCYWJE9wZW5TU0wgR2VuZXJhdGVkIFNlcnZlciBDZXJ0aWZpY2F0ZTAdBgNV
HQ4EFgQUPBLSjbXJAfsEBY3pJqrxZ68mB4cwgcEGA1UdIwSBuTCBtoAUxBvX29ll
lmXxJCfyYVGEY6WD4lqhgZmkgZYwgZMxCzAJBgNVBAYTAkdCMRAwDgYDVQQIDAdF
bmdsYW5kMQ8wDQYDVQQHDAZMb25kb24xETAPBgNVBAoMCHRlc3QuY29tMQwwCgYD
VQQLDANsYWIxHDAaBgNVBAMME3Jvb3QubGFiY2EudGVzdC5jb20xIjAgBgkqhkiG
9w0BCQEWE3J1c3NlbGxAaWxsdW1pby5jb22CAhAAMA4GA1UdDwEB/wQEAwIFoDAT
BgNVHSUEDDAKBggrBgEFBQcDATANBgkqhkiG9w0BAQsFAAOCAgEAWV6p7X+VCi1/
8CeDwCgtK+Vlq4suz2kOQDX0/EMc3tgXXIvCIFmGQHNkSP5XCGrw5F8IZWd9/fOM
+HI0kTBDhzQtovjxUPE6GoITM1Jrct5TGuvGlBzeGf/89x7IUrTX4r+8cIfeL9qz
Tp3QNuMSt5VVVTHSNmNh054RTj12tyn1Sbr54CApsPQgnIg7h2pErJDKCkqS7nPR
b2ix+PAWJ+O2bPt6wPltO32hSF128SXQ4a5O48FIxskSVyTqPxdTDLdE3kMLf+zE
/vfSfkPpK+H/qOL5un3Na025jwmPIGHZRj1d5r01I5wAyzYSL6hf3gMM3ywQsAJD
gnc0l0oHgy0txCqLXSPuWxU1OztywdpyLSh5PIZ5cOR/C6AEEVmgCN28Rx+8C74f
5i1hVOdYw4Nmq7IgMt51Z4XGsXzQyWGMwCza9xxOTlZHOgqPNMrUpT75U54jDfzl
irxyQl/4HLmhOtG/Qho7oRkBsoMmwWsiqCQN27O/UofYdNTZ8xqSOPt762LKGjPJ
aiIV2OzRb/MP+0hcMPnoECKgcFWz0ftYVNJYAd/JhPomrqJlqZ8yWb8bgGhGU10x
lGj4GHLNZEr63Bi94gExNvxQncBYzPjBwRrUKhgXxph5eHTh14tPj+YP5rkzsjJv
px57Lmrc7pDwG/8DsRkC8ejxhD72RoI=
-----END CERTIFICATE-----
```

If when entering this command the output looks like binary junk, you may have a DER encoded certificate. We can check this in the next step. You can validate the contents of the certificate with the following command:

```
[user@core0.pce certs]# openssl x509 -text -noout -in ./pce.test.com.crt
    Data:
        Version: 3 (0x2)
        Serial Number: 4096 (0x1000)
    Signature Algorithm: sha256WithRSAEncryption
Issuer: C=GB, ST=England, O=test.com, OU=lab,CN=intermediate.labca.test.com/emailAddress=russell@illumio.com

        Validity
```

```
            Not Before: Sep 22 07:16:48 2015 GMT
            Not After : Sep 21 07:16:48 2017 GMT
Subject: C=GB, ST=England, O=test.com, OU=lab, CN=pce.test.com/emailAddress=russell@illumio.com

//output suppressed

        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Server
            Netscape Comment:
                OpenSSL Generated Client and Server Certificate
            X509v3 Subject Key Identifier:
                A7:A0:09:D6:C8:1B:BC:DC:D6:9A:3C:56:97:0F:D8:C4:B1:A3:1F:3F
            X509v3 Authority Key Identifier:
                keyid:C4:1B:D7:DB:D9:65:96:65:F1:24:27:F2:61:51:84:63:A5:83:E2:5A
                DirName:/C=GB/ST=England/L=London/O=test.com/OU=lab/CN=root.labca.test.com/emailAddress=russell@illumio.com
                serial:10:00

            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Subject Alternative Name:
                DNS:pce.test.com, DNS:pce-core0.test.com, DNS:pce.test.com

//output suppressed

[user@core0.pce certs]#
```

In this output we see the issuer CN, which is the name of the certificate authority that signed the certificate. We also see that the subject CN should match our DNS hostname or Alias, as this will be seen by clients and VENs. We also see the next host up the signing chain under DirName. This may in many cases be the root CA. These are relevant as we will also need the certificates for all intermediates and the root CA so we can make a bundle file.

If, when you issue the X509 command you get something like the following, you may have a DER encoded certificate.

```
unable to load certificate
140190015219616:error:0906D06C:PEM routines:PEM_read_bio:no start
line:pem_lib.c:703:Expecting: TRUSTED CERTIFICATE
```

If this is the case, you check by telling OpenSSL this is a DER certificate and see if it can successfully read it. This can be done by adding '`-inform der`' to the OpenSSL command as follows:

```
[user@core0.pce certs]# openssl x509 -text -noout -in ./pce.test.com.crt -inform der
```

If this produces an error, check with your issuer to ensure the certificate is not damaged. If this provides the above output, you will want to convert this to PEM with the following command:

```
[user@core0.pce certs]# openssl x509 -in pce.test.com.der -inform der -out pce.test.com.crt
```

You can specify '`-outform pem`', but it isn't required as PEM is the default encoding format.

4. Check that the Private Key and the Certificate are related.

   This may seem obvious, but we regularly see issues where the Key and certificate do not match. This can be due to the way the certificate signing process has been followed, or if the key was accidently generated multiple times. As it is crucial that the two are a mathematical pair, we can enter the following command to validate this:

   ```
   [user@core0.pce certs]# openssl x509 -modulus -noout  -in ./pce.test.com.crt | openssl md5
   (stdin)= 013dcd11bfb6f2f3ff11406da383d03e

   [user@core0.pce certs]# openssl rsa -modulus -noout  -in ./pce.test.com.key | openssl md5
   (stdin)= 013dcd11bfb6f2f3ff11406da383d03e
   ```

   Here, we check the modulus on both the certificate and the key. These produce a large output, so we pipe this to MD5 to produce something more readable. Both these values should match. If they don't, then the key and certificate are not a pair.

Detail

5. Creating the Bundle File.

   The Web Service on the PCE leverages the nginx package to deliver the HTTPS interface. Nginx requires a bundle file, so this needs to be created. The bundle includes, in order, the host certificate, followed by all intermediates in order, followed by the root CA certificate.

   You will need the host certificate in PEM format, as seen at the start of section 3. You will also need the intermediate(s) and root CA certificates in the same format. To create the bundle, use the following command on a single line:

   ```
   [user@core0.pce certs]# cat ./pce.test.com.crt ./intermediate.labca.test.com.crt
   ./root.labca.test.com.crt > ./pce.test.com.bundle.crt
   ```

6. Checking that the CA and Intermediate certificates are trusted by the host.

   A number of PCE components require that the certificates are valid and are signed by a trusted authority. In many cases, the organization's Certificate Authorities would be pre-installed on their hosts. However, a number of cases exist where this isn't the case, especially in test environments.

   You can validate the certificate authorities are in the host store with the following command:

   ```
   [user@core0.pce certs]#
   pce.test.com.bundle.crt: OK
   ```

   If the output produces an error, then the intermediate and CA certificates may need to be installed into the server store. Contact your local Unix admin team for assistance, or you can use the following steps. This example is for RHEL and Centos6. Note that elevated privilege is needed to execute this step. In addition, there is a dependency on the '`ca-certificates`' package for this command to be run.

   ```
   [root@core0.pce certs]# cp intermediate.labca.test.com.crt /etc/pki/ca-trust/source/anchors/
   [root@core0.pce certs]# cp root.labca.test.com.crt /etc/pki/ca-trust/source/anchors/
   [root@core0.pce certs]# update-ca-trust enable
   [root@core0.pce certs]# update-ca-trust extract
   [root@core0.pce certs]# openssl verify ./intermediate.labca.test.com.crt
   ./intermediate.labca.test.com.crt: OK
   [root@core0.pce certs]# openssl verify ./pce.test.com.crt
   ./core0.pce.test.com.crt: OK
   ```

   On some distributions we have occasionally seen the following error. This can usually be dealt with using the '`force-enable`' option.

```
[root@core0.pce certs]# update-ca-trust enable

update-ca-trust: nss 32 bit is installed. You should install p11-kit-trust 32 bit.
update-ca-trust: aborting, because the nss / p11-kit setup is inconsistent.

[root@core0.pce certs]# update-ca-trust force-enable
[root@core0.pce certs]#
```

7. Placing the correct files in the correct locations.

   In a PCE 2x2 or 4x2 deployment, the Web Certificate bundle and the related private key will need to be placed on all PCE nodes. For example, in a typical 2x2 deployment, this would be on both core nodes and both data0 and data1 nodes.

Internal Only Comments

Attachment