EVE Connectivity Through HTTPS Proxy Server

Motivation

EVE Nodes are managed from EVC(Edge Virtualization Controller). EVE uses HTTPS APIs to talk to the EVC for fetching the configuration and exporting operational information. These APIs are HTTP URI endpoints, and the connection is secured through a TLS session. To establish trust either way, mutual TLS authentication scheme is used, where both EVE(the client) and EVC(the server) need to prove their identity. For this EVE and EVC exchange their X.509 certificates.

However this mechanism does not work if the TLS session is intercepted by a network proxy, e.g. a HTTPS proxy doing lawful inspection of the payload exchanged between the client and server or a TLS terminating HTTPS load-balancer. This is because the proxy server may terminate TLS connection from EVE, and sometimes break that into 2 separate TLS sessions, one facing the server and another one facing the client. The one facing the client will act as server, and one facing the server will act as client. So this poses the following issues:

1. From EVC perspective, identifying an EVE instance with the incoming X.509 client certificate will not work, since the client certificate may not be from the actual client anymore (instead it is from the proxy server). Or in case of load-balancing proxies, EVC might not have access to any client certificate.
2. From EVE perspective, using existing root CA to verify EVC identity will not work, as the incoming X.509 certificate from EVC may not be from the actual EVC server anymore (instead it is from the proxy server)
3. Since proxy is terminating TLS and re-opening TLS on the other segment, there is a possibility of payload getting modified in the middle. Without any mechanisms to detect this, this can compromise integrity of EVE-EVC communication channel
4. Since proxy can inspect the TLS payload in clear text, sensitive parts of payloads(like access credentials, keys etc) should be protected from getting exposed in transit.

Overview

Therefore, to enable EVE connectivity in such environments where proxy is used, following mechanism is used.

1. There will be new version of EVC APIs introduced. The current version of APIs is v1, so the new version of APIs will be called v2 APIs (more on this below)
2. Unlike v1, v2 APIs will not use client certificates to identify the source of the connection.
3. Concept of "Payload Envelopes" is introduced in v2 to protect integrity of the payload in transit, by signing the payload. (more on this below)
4. Object level security in transit is employed by encrypting sensitive parts of the payload end-to-end between EVE and EVC (more on this below)

Payload Envelopes

To protect the payload from getting modified in transit, the payload needs to be signed End-to-End between EVE and EVC. EVE or EVC, when they receive payload from the other, will hash the payload and compute signature of the digest, and compare it with the signature that accompanied the payload. We call this additional data that goes with the payload, and carries the signature of the payload, as the "Envelope". If the signature in the envelope is different from the signature computed from the payload, then one can detect that the payload has been modified during the transit. Therefore "Envelopes" help detect that payload has been modified during the transit. This mechanism is as strong as the signing method used, so the following method is used to compute the signature of the payload:

1. EVE uses its device private key to sign the payload. The device certificate carries the public key. So EVC can validate the signature using device certificate
2. EVC uses private key of its choice, makes the corresponding certificate available for download by anyone in the Internet. EVC signs using its private key, and EVE uses the published certificate to validate the signature
3. The certificates themselves are exchanged using secure transactions. (please see "Provisioning" section below)
4. As long as these EVE and EVC private keys are protected at their endpoints, signatures can't be forged in transit

Provisioning of X.509 Certificates for Envelope Verification

To use payload envelopes, both EVE and EVC need to know the certificates of the other party, before they can start sending payloads with envelopes carrying signatures of the payload. This is how they are provisioned:

1. EVE, during self-register, sends its device certificate as payload. This payload is then signed by the onboarding key. Since EVC generates onboarding certificate and key, EVC can verify the signature and make sure that the device certificate was indeed sent by EVE. One can also provision the device certificate through offline registration, by adding the device certificate of EVE to EVC's list of device certificates, if one has administrative access to EVC.

2. EVC, makes its signing certificates publicly downloadable at a well-known HTTP API. EVE has access to the root CA that signs other certificates created by EVC. Therefore, by downloading the certificates from the well-known API, and verifying the certificate chain all the way to the root CA that it has, EVE can validate that the signing certificate was indeed from the EVC.

With these steps, EVE and EVC are ready to sign and also validate the signature of the other party.

EVE-EVC communication with V2 APIs

EVE does the following whenever it does a POST to EVC's v2 API:

1. EVE prepares its POST payload as it would for a v1 API. Then it calculates digest and signature of the payload, and adds an envelope around it that has:
    a. The actual payload
    b. SHA of the payload, signed by the device certificate
    c. An Identifier of the certificate that corresponds to the private key used to sign

2. EVE sends this envelope as the payload to the v2 version of the API

EVE does the following whenever it receives a POST response to a v2 API:

1. EVE treats the response payload as envelope structure, and retrieves the certificate identifier
2. EVE checks if it has the certificate associated with the identifier in the envelope. If yes, proceeds with 4 below.
3. If certificate is not available, EVE downloads the certificates again from EVC portal
4. EVE computes digest from the payload, and verifies the signature using the certificate
5. If verification is successful, further processing of payload happens as it would in the v1 API

EVC does the following whenever it receives a POST request for a v2 API:

1. EVC treats the POST payload as envelope structure, and receives the certificate identifier
2. By this time EVC MUST have the certificate, as this certificate is the device certificate used to register the device
3. EVC computes digest from the payload, and verifies the signature using the device certificate
4. If verification is successful, further processing of the payload happens as it would in the v1 API

EVC does the following whenever it prepares response to a POST of a v2 API:

1. EVC prepares the response as it would for a v1 API. Then it calculates the digest and signature of the payload and adds similar envelope discussed above
2. EVC sends the envelope as the response to the POST

Transitioning to v2 APIs

The eventual goal is to EOL the v1 APIs, and support only v2 APIs in EVC. But today we might already have EVE instances talking v1 APIs to a given EVC. Following is the plan to transition EVE and EVC to v2

Let the current EVC software version be 3.0 and EVE version be 3.0, and there is no support for v2 API in either of EVC and EVE versions. Also, assume that some of the EVE instances are running slightly older versions, say 2.0.

Now, following is the transitioning order to move to v2

1. EVC implements support for v2, and supports both v1 and v2 APIs in version 3.1.
2. EVC is upgraded to 3.1. At this point, both v1 and v2 APIs are supported by EVC, so existing EVE instances with 3.0 or below will continue to work with v1.
3. EVE software adds support to switch to v2 APIs, with default choice as v1, in version 3.1.
4. Some of the EVE instances are upgraded to 3.1, and configured to use v2 API. They will work too, since EVC supports v1 and v2 at this point. Any EVE-EVC integration issues can be debugged and fixed at this point.
5. All the devices are upgraded to 3.1, and switched to v2 API
6. EVC stops supporting v1 from 3.2
7. EVC is upgraded to 3.2. This should not be a problem since all the devices are moved to v2 at this point
8. EVE stops supporting v1, and makes v2 as the only supported API version, in 3.2
9. EVE instances are upgraded to 3.2.
10. After this point, any EVE instance trying to go back to a version 3.1 or below(which use v1 API), will see failure, and come back to the current version. This is because the older version will try to talk v1 to EVC, which is discontinued, and hence EVE will timeout, declare the old image toxic and fallback to current version.

Certificate Rotation

If EVC decides to change the signing keys it is using, it can do the following steps:

1. Create new private, public key
2. Generate the new X.509 certificate signed by the intermediate CAs rooted to the root CA shared with the EVE.
3. Make the new X.509 certificate available in the API to fetch EVC certificates
4. Update the certificate identifier in the payload envelopes

Then,

1. EVE does a lookup for the certificate identifier in the envelope

2. The lookup fails, and that triggers EVE to fetch EVC certificates again
3. With this fetch, EVE now has the latest EVC certificate to verify the signature

Object Level Security

One of the issues discussed above was to secure the sensitive information in the payload from getting exposed in transit, due to termination of TLS at proxy server. To address this, sensitive parts of the configuration will be encrypted using a shared symmetric key. The details on how this shared symmetric key is arrived at, and how the encryption and decryption happen, can be found at Secure Storage of Sensitive Information on EVE

Normative Reference of v2 APIs

This is a normative reference to the various EVC APIs, their v1/v2 differences, the payload fields, components of envelope etc.

Order	API	Version	Operation	TLS Client Cert	Args	Response	Request Enveloped?	Response Enveloped?	Context of invocation
1	edgeDevice/**register**	v1	POST	onboarding certificate	serial, soft serial, device cert	standard HTTP response code	No	No	initial onboarding
1	edgeDevice/**register**	v2	POST	None	onboarding key, serial, soft serial device cert	standard HTTP response code	Yes, signed by onboarding certificate	No	initial onboarding
2	edgeDevice/**Certs** ***(new)***	v1/v2	GET	None. Preference is to use HTTP if proxy supports.	None	Controller Certs, and HTTP response code	No	No	at every boot, if device does not have any controller cert
3	edgeDevice/**config**	v1	POST	device certificate	device certificate	Device Configuration	No	No	at boot time. When we act on PCR values, PCR Quote will have to be sent along with the config request
3	edgeDevice/**config**	v2	POST	None	device certificate	Device Configuration	Yes, signed by device certificate, and contains **device certificate**	Yes, signed by Controller Cert shared in Order 2	at boot time. When we act on PCR values, PCR Quote will have to be sent along with the config request
4	edgeDevice/id/<uuid>/**attest** *Sub-type* ***ATTEST_REQ_CERT***	v2	POST	None	Additional Certs created by Device	standard HTTP response code	Yes, signed by device certificate. Envelope has **device UUID**	No	at boot time
5	edgeDevice/id/<uuid>/**attest** *Sub-type* ***ATTEST_REQ_NONCE***	v2	POST	None	None	Nonce	Yes, signed by device certificate. Envelope has **device UUID**	Yes, signed by Controller Cert shared in Order 2	Precedes PCR Quote POST
6	edgeDevice/id/<uuid>/**attest** *Sub-type* ***ATTEST_REQ_QUOTE***	v2	POST	None	Nonce, PCR Quote. Quote is signed with restricting signing key from device	AttestResult(pass, fail) along with standard HTTP response code	Yes, signed by device certificate. Envelope has **device UUID**	Yes, signed by Controller Cert shared in Order 2	Precedes config request
7	edgeDevice/**config**	v1	POST	device certificate	device UUID, hash of current config	Device Configuration, sensitive data encrypted with shared symmetric key	No	No	When we act on PCR values, PCR Quote will have to be sent along with the config request
7	edgeDevice/**config**	v2	POST	device certificate	device UUID, hash of current config	Device Configuration, sensitive data encrypted with shared symmetric key	Yes, signed by device certificate. Envelope has **device UUID**	Yes, signed by Controller Cert shared in Order 2	When we act on PCR values, PCR Quote will have to be sent along with the config request