gRPC Network Operations Interface

The Google Remote Procedure Call (gRPC) Network Operations Interface (gNOI) is a suite of microservices, each corresponding to a set of operations. This module describes the supported gNOI services.

Information About the gRPC Network Operations Interface

gNOI Protocol

gNOI defines a set of gRPC-based microservices for executing operational commands on network devices. The gNMI service defines operations for configuration management, operational state retrieval, and bulk data collection through streaming telemetry. gNOI only allows the adoption of services that a device supports. gNOI supports the OS installation service.

gNOI can be used with or without user authentication. User authentication is disabled by default. Use the **gnxi secure-password-auth** command to enable user authentication. For information about enabling user authentication through the OpenConfig model, see https://github.com/YangModels/yang/blob/master/vendor/cisco/xe/1751/openconfig-system-management.yang.

The gNOI protocol supports the following operations:

- Certificate Management
- Bootstrapping

Certificate Management Service

The Certificate Management Service primarily exports two main RPCs, Install and Rotate, that are used for the installation of new certificates, and the rotation of existing certificates on a device, respectively.

The following RPCs are supported by the Certificate Management Service:

- Install: Installs a certificate. All certificates are uniquely identified by a certificate ID. The certificate ID is a string.

- Rotate: Rotates an existing certificate.
- RevokeCertificates: Revokes one or more certificates.
- GetCertificates: Queries all certificates.
- CanGenerateCSR: Queries whether the device can generate a Certificate Signing Request (CSR).

Trustpoints and certificates created through the RPCs mentioned above persist across switchovers and device reboots.

The following is a sample Certificate Management Service definition:

```
service CertificateManagement {
 rpc Install(stream InstallCertificateRequest)
  returns (stream InstallCertificateResponse);

 rpc Rotate(stream RotateCertificateRequest)
  returns (stream RotateCertificateResponse);

 rpc RevokeCertificates(RevokeCertificateRequest)
  returns (RevokeCertificateResponse);

 rpc GetCertificates(GetCertificateRequest)
  returns (GetCertificateResponse);

 rpc CanGenerateCSR(CanGenerateCSRRequest)
  returns (CanGenerateCSRResponse);
}
```

Install RPC

The Install RPC adds a new certificate to a device by creating a new CSR request. The new certificate is associated with a new certificate ID on the device. If the device has a pre-existing certificate with the given certificate ID, the operation fails.

The Install RPC is a bidirectional streaming RPC. It has an input (InstallCertificateRequest) and an output (IntsallCertificateResponse) both of which are streaming. If the stream is broken, or any steps in the process fail, the device rolls back the changes.

The following is an example of the Install RPC definition and messages:

```
rpc Install(stream InstallCertificateRequest)
returns (stream InstallCertificateResponse);

// Request messages to install new certificates on the target.
message InstallCertificateRequest {
  // Request Messages.
  oneof install_request {
    GenerateCSRRequest generate_csr = 1;
    LoadCertificateRequest load_certificate = 2;
  }
}
// Request to generate the CSR.
message GenerateCSRRequest {
  // Parameters for creating a CSR.
  CSRParams csr_params = 1;
  // The certificate id with which this CSR will be associated. The target
  // configuration should bind an entity which wants to use a certificate to
  // the certificate_id it should use.
  string certificate_id = 2;
}
// Parameters to be used when generating a Certificate Signing Request.
message CSRParams {
  // The type of certificate which will be associated for this CSR.
  CertificateType type = 1;

  // Minimum size of the key to be used by the target when generating a
  // public/private key pair.
  uint32 min_key_size = 2;

  // If provided, the target must use the provided key type. If the target
  // cannot use the algorithm specified in the key_type, it should cancel the
  // stream with an Unimplemented error.
  KeyType key_type = 3;

  // --- common set of parameters applicable for any type of certificate --- //
  string common_name = 4;           // e.g "device.corp.google.com"
  string country = 5;               // e.g "US"
  string state = 6;                 // e.g "CA"
  string city = 7;                  // e.g "Mountain View"
  string organization = 8;          // e.g "Google"
  string organizational_unit = 9;   // e.g "Security"
  string ip_address = 10;
  string email_id = 11;
}
// A certificate.
message Certificate {
  // Type of certificate.
  CertificateType type = 1;

  // Actual certificate.
  // The exact encoding depends upon the type of certificate.
  // for X509, this should be a PEM encoded Certificate.
  bytes certificate = 2;
}

message LoadCertificateRequest {
  // The certificate to be Loaded on the target.
  Certificate certificate = 1;

  // The key pair to be used with the certificate. This is provided in the event
  // that the target cannot generate a CSR (and the corresponding public/private
  // keys).
  KeyPair key_pair = 2;

  // Certificate Id of the above certificate. This is to be provided only when
  // there is an externally generated key pair.
  string certificate_id = 3;

  // Optional pool of CA certificates to be used for authenticating the client.
  repeated Certificate ca_certificate = 4;
}

// A message representing a pair of public/private keys.
message KeyPair {
  bytes private_key = 1;
  bytes public_key = 2;
}

// Response Messages from the target for the InstallCertificateRequest.
message InstallCertificateResponse {
  // Response messages.
  oneof install_response {
    GenerateCSRResponse generated_csr = 1;
    LoadCertificateResponse load_certificate = 2;
  }
}

// GenerateCSRResponse contains the CSR associated with the Certificate ID
// supplied in the GenerateCSRRequest. When a Certificate is subsequently
// installed on the target in the same streaming RPC session, it must be
// associated to that Certificate ID.
//
// An Unimplemented error will be returned if the target cannot generate a CSR
// as per the request. In this case, the caller must generate its own key pair.
message GenerateCSRResponse {
  CSR csr = 1;
}

// A Certificate Signing Request.
message CSR {
  // Type of certificate.
  CertificateType type = 1;

  // Bytes representing the CSR.
  // The exact encoding depends upon the type of certificate requested.
  // for X509: This should be the PEM encoded CSR.
  bytes csr = 2;
}
```

After the target device is up and gNOI is in default state, the controller (a third-party implementation) uses the Install RPC to install a certificate that is signed by a Certificate Authority (CA). The certificate is uniquely identified by a certificate ID. This ID is used as the trustpoint name in the Public Key Infrastructure (PKI) configuration. The installation will fail, if you try to install a certificate that has an existing certificate ID.

The following section describes how a CSR is generated by a device:

1. The device generates a self-signed certificate through the Install RPC. The controller does not require a copy of this certificate because in encrypted mode (or gNMI default state) the controller does not validate the certificate presented by the target device. This is the default state.
2. The controller requests the device to generate a CSR, sends the CSR to the CA, and gets the signed certificate back from the CA.
3. The signed certificate is installed into the device along with the CA certificates used to sign the certificate. The CA certificate is present in the *ca_certificates* bundle, and is required by the PKI to install the device certificate.
4. The gNMI or the gNOI service restarts using the newly installed certificate that is now in the provisioned state.

Rotate RPC

The Rotate RPC renews an existing certificate; a certificate that is already installed. If a certificate is not already installed, the Rotate RPC fails. A certificate that is not in use can be rotated, but the client cannot test it.

The following is a sample Rotate RPC definition:

```
rpc Rotate(stream RotateCertificateRequest)
returns (stream RotateCertificateResponse);

// Request messages to rotate existing certificates on the target.
```

```
message RotateCertificateRequest {
  // Request Messages.
  oneof rotate_request {
    GenerateCSRRequest generate_csr = 1;
    LoadCertificateRequest load_certificate = 2;
    FinalizeRequest finalize_rotation = 3;
  }
}

// A Finalize message is sent to the target to confirm the Rotation of
// the certificate and that the certificate should not be rolled back when
// the RPC concludes. The certificate must be rolled back if the target returns
// an error after receiving a Finalize message.
message FinalizeRequest {
}

message RotateCertificateResponse {
  // Response messages.
  oneof rotate_response {
    GenerateCSRResponse generated_csr = 1;
    LoadCertificateResponse load_certificate = 2;
  }
}
```

The Rotate RPC differs from the Install RPC in the following ways:

- PKI has to save or cache the old certificate and the CA certificate when installing a new certificate (for the purpose of rollback).
- The controller creates a new connection to test whether the renewed certificate works, and in case of success, finalizes the certificate rotation.

Revoke RPC

This RPC is used to revoke one or more certificates, each uniquely identified by a certificate ID. Revocation of a certificate results in the corresponding trustpoint to be removed from the Cisco IOS XE configuration. If the corresponding trustpoints are currently in use, or if the trustpoints do not exist, revocation of the certificates may fail.

A RevokeCertificate RPC may have certificates revoked successfully or unsuccessfully. On the target device, revocation is a simple delete operation; the actual revocation with the CA is done by the client. If the client revokes a certificate that is in use, new connections fail, but the existing connections are unaffected.

The following is a sample RevokeCertificate RPC:

```
// An RPC to revoke specific certificates.
// If a certificate is not present on the target, the request should silently
// succeed. Revoking a certificate should render the existing certificate
// unusable by any endpoints.
rpc RevokeCertificates(RevokeCertificatesRequest)
returns (RevokeCertificatesResponse);

message RevokeCertificatesRequest {
  // Certificates to revoke.
  repeated string certificate_id = 1;
}

message RevokeCertificatesResponse {
  // List of certificates successfully revoked.
  repeated string revoked_certificate_id = 1;

  // List of errors why certain certificates could not be revoked.
  repeated CertificateRevocationError certificate_revocation_error = 2;
}

// An error message indicating why a certificate id could not be revoked.
message CertificateRevocationError {
  string certificate_id = 1;
  string error_message = 2;
}
```

GetCertificate RPC

This RPC queries all certificate IDs.

The response to the query contains the following information:

- Certificate information for all the certificates that are identified by a certificate ID.
- The list of endpoints, for example, tunnels, daemons, and so on, that use this certificate.

Note Endpoints are not supported.

Note Responses do not contain the *ca_certificate* bundle.

The following is a sample GetCertificate RPC:

```
// An RPC to get the certificates on the target.
rpc GetCertificates(GetCertificatesRequest) returns (GetCertificatesResponse);

// The request to query all the certificates on the target.
message GetCertificatesRequest {
}

// Response from the target about the certificates that exist on the target what
// what is using them.
message GetCertificatesResponse {
  repeated CertificateInfo certificate_info = 1;
}

message CertificateInfo {
  string certificate_id = 1;
  Certificate certificate = 2;

  // List of endpoints using this certificate.
  repeated Endpoint endpoints = 3;

  // System modification time when the certificate was installed/rotated in
  // nanoseconds since epoch.
  int64 modification_time = 4;
}
```

```
// An endpoint represents an entity on the target which can use a certificate.
message Endpoint {
  // Type of endpoint that can use a cert. This list is to be extended based on
  // conversation with vendors.
  enum Type {
    EP_UNSPECIFIED = 0;
    EP_IPSEC_TUNNEL = 1;
    EP_DAEMON = 2;
  }
  Type type = 1;

  // Human readable identifier for an endpoint.
  string endpoint = 2;
}
```

CanGenerateCSR RPC

This RPC queries whether a device can generate a CSR for a specific key type, certificate type, and key size. The supported key type is Rivest, Shamir, and Adelman (RSA), and the supported certificate type is X.509.

When this RPC request is made for installing a completely new certificate as part of the Install RPC, the device must ensure that the certificate ID is new and no entities on the device are bound to this certificate ID. If any existing certificate matches the certificate ID, this request fails.

When this RPC request is made for rotating an existing certificate as part of the Rotate RPC, the device must ensure that the certificate ID is already available. If certificate rotation proceeds to load the certificate, it must associate the new certificate with the previously created certificate ID.

The following is a sample CanGenerateCSR RPC:

```
// An RPC to ask a target if it can generate a Certificate.
rpc CanGenerateCSR(CanGenerateCSRRequest) returns (CanGenerateCSRResponse);

// A request to ask the target if it can generate key pairs.
message CanGenerateCSRRequest {
  KeyType key_type = 1;
  CertificateType certificate_type = 2;
  uint32 key_size = 3;
}

// Algorithm to be used for generation the key pair.
enum KeyType {
  // 1 - 500, for known types.
  // 501 and onwards for private use.
  KT_UNKNOWN = 0;
  KT_RSA = 1;
}

// Types of certificates.
enum CertificateType {
  // 1 - 500 for public use.
  // 501 onwards for private use.
  CT_UNKNOWN = 0;
  CT_X509 = 1;
}

// Response from the target about whether it can generate a CSR with the given
// parameters.
message CanGenerateCSRResponse {
  bool can_generate = 4;
}
```

Mutual Authentication

Mutual authentication is a two-way authentication; two parties authenticate each other at the same time. To enable mutual-authentication, use the **gnmi-yang secure-peer-verify-trustpoint** command. If this command is not enabled, the authentication service validates the gNMI client against all the existing trustpoints and the contents of the trustpool.

Rotation of the CA certificates for mutual authentication requires the client to present a new bundle to the target device, and the old bundle to be removed. However, the CA certificates reside in a trustpool, and cannot be selectively deleted from the trustpool.

Bootstrapping with Certificate Service

After installing gNOI certificates, bootstrapping is used to configure or operate a target device. When a target device does not have any pre-existing certificates, bootstrapping allows the installing of certificates by using the gNOI Certificate Management Service. After the certificate installation, the device is capable of establishing secure gNOI or gNMI connections. This process assumes a pre-existing secure environment.

To enable gNMI bootstrapping, use the **gnxi secure-init** command.

Note The gNOI Certificate Management Service must be installed before bootstapping.

The gNOI Certificate Management Service has two states. These states are supported by both the gNOI service and the gNMI service.

- Default/Encrypted: gNOI and gNMI on the device use a self-signed (default) certificate that the client does not verify; the certificate does not require authentication. In this state, only the gNOI certificate service is enabled on the target device.
- Provisioned: gNOI and gNMI on the device use an installed certificate that is verified by the client, and the client presents its certificate, which the device verifies against its certificate store. The device verifies the client certificate only if mutual authentication is enabled.

Additional References for the gRPC Network Operations Interface

Related Documents

Related Topic	Document Title
DevNet	https://developer.cisco.com/site/ios-xe/
gNOI	https://github.com/openconfig/gnoi

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/support

Feature Information for the gRPC Network Operations Interface

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information for the gRPC Network Operations Interface

Feature Name	Release	Feature Information
gNOI Certificate Management	Cisco IOS XE Amsterdam 17.3.1	The gNOI Certificate Management Service provides RPCs to install, rotate, get certificate, revoke certificate, and generate certificate signing request. In Cisco IOS XE Amsterdam 17.3.1, this feature was implemented on the following platforms: • Cisco Catalyst 9200 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 Series Switches • Cisco Catalyst 9600 Series Switches

Feature Name	Release	Feature Information
gNOI Bootstrapping with Certificate Service	Cisco IOS XE Amsterdam 17.3.1	After installing gNOI certificates, bootstrapping is used to configure or operate a target device. gNMI bootstrapping is enabled by using the **gnxi-secure-init** command and disabled by using the **secure-allow-self-signed-trustpoint** command. In Cisco IOS XE Amsterdam 17.3.1, this feature was implemented on the following platforms: • Cisco Catalyst 9200 Series Switches • Cisco Catalyst 9300 Series Switches • Cisco Catalyst 9400 Series Switches • Cisco Catalyst 9500 Series Switches • Cisco Catalyst 9600 Series Switches