Google Cloud

Edge Security

Transport Security Considerations

About TLS/SSL

- Secure Socket Layer is the predecessor to Transport Layer Security
- TLS is the standard security technology for establishing an encrypted link between a web server and a web client, such as a browser or an app.
- An encrypted link ensures that all data passing between the server and the client remains private.
- To use TLS, a client makes a secure request to the server by using the encrypted HTTPS protocol, instead of the unencrypted HTTP protocol.

Edge support - TLS/SSL

- Edge supports one-way TLS and two-way TLS in both public cloud and private cloud deployments
- One-way TLS enables the TLS client to verify the identity of the TLS server.
  - For example, an app running on an Android phone (client) can verify the identity of Edge APIs (server).
- Edge also supports a stronger form of authentication using two-way, or client, TLS.
  - You typically implement two-way TLS to enhance security end-to-end and protect your data from client attacks such as client spoofing or man-in the middle attacks.
  - In two-way TLS, the client verifies the identity of the server followed by the server verifying the identity of the client.

What information is encrypted via TLS?

- Connection is made across network using IP address
    - IP address can generally be used to get a domain name
- Once connection is established, all data is encrypted
    - Including URL, headers, query parameters, verb and payload
- Snoopers know the destination server and how much data is sent and nothing more

Can passwords be sent as query parameters?

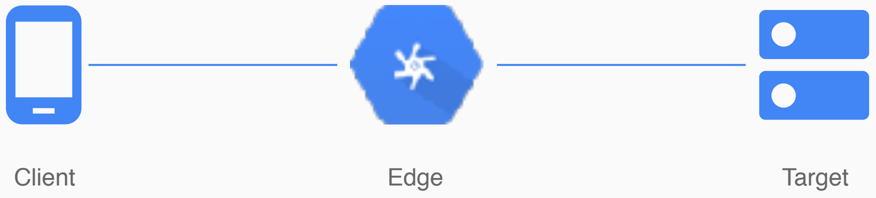
```
GET https://example.com/v1/api/login?user=bob&pw=opensesame
```

- Since TLS provides communications security, can passwords be sent as query parameters?
    - DON'T!
- The issue is data at rest – not data in motion
    - URLs are often logged in clear text server logs, including query parameters
    - GET requests are bookmarkable and visible in browser history
- URLs are generally not treated in a secure fashion, so don't use them to send sensitive data
    - Payload or headers are better

Transport layer level security

Transport layer level security (1 and 2-way TLS)

- Client → Edge
- Edge → Target



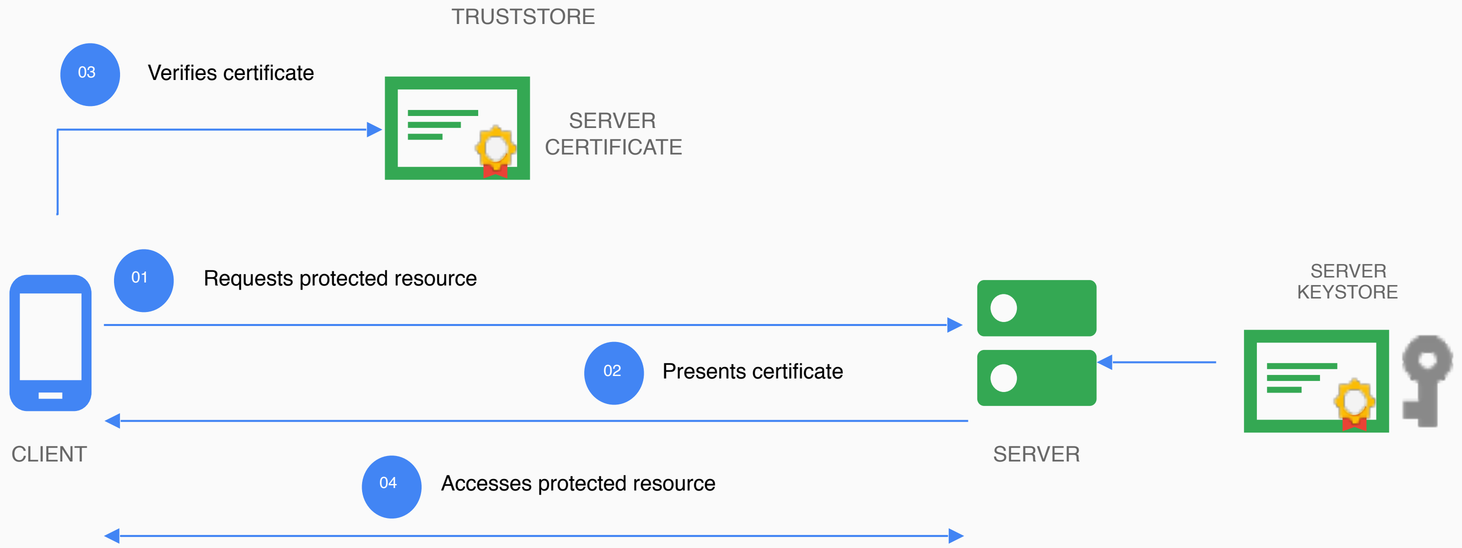
One way vs Two way TLS

- One-way TLS (server validation)
  - Server presents certificate, client does not
  - Client optionally validates the server certificate
  - Server must validate client via other means (HTTP message traffic)
  - Basic Auth, OAuth, etc.
  - This is standard web https
- Two-way TLS (mutual authentication)
  - Both client and server present a certificate

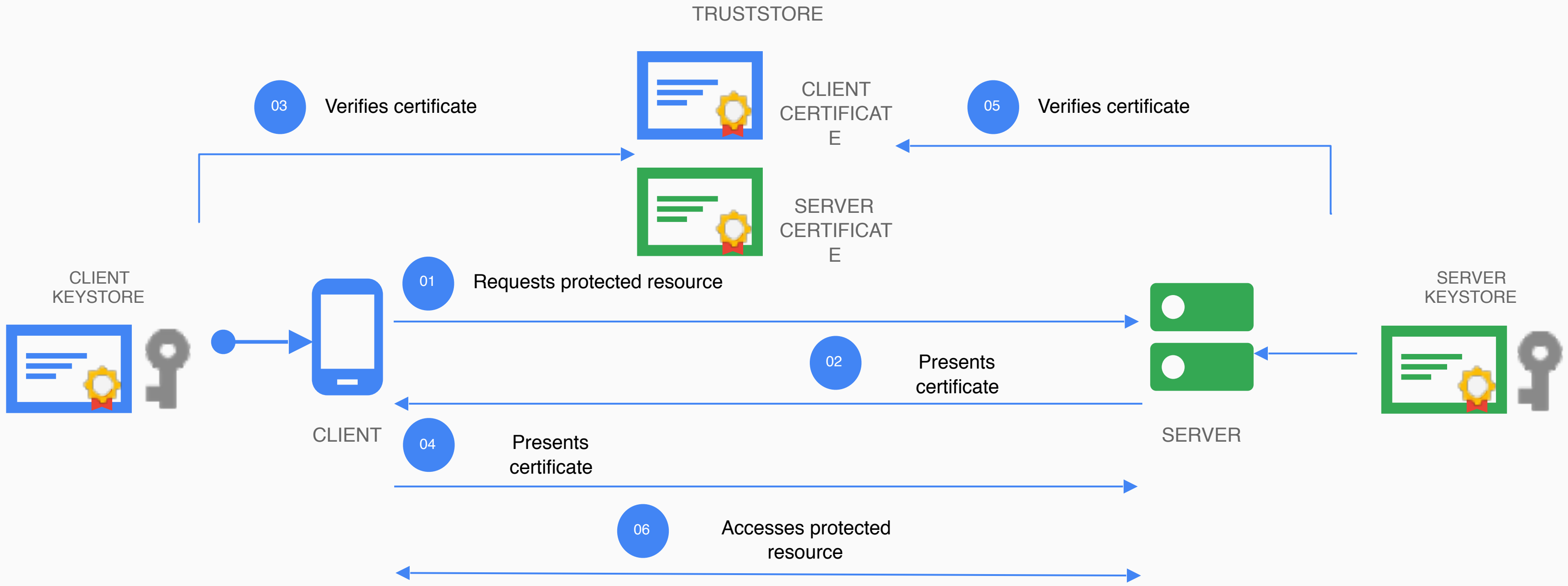
Edge keystores and truststores

- Keystore
    - Used to store certificate(s) to be presented to remote server during SSL communication
    - Also stores private key used to encrypt TLS traffic to the remote server
- Truststore
    - Used to store certificates to compare with remote certificates received during TLS communications
    - Communication may be set to only allow communication with trusted servers
- Keystores and truststores are used for both client communication (via virtual hosts) and target communication (via target endpoints and target servers)

Transport Layer Security (1-way TLS)



Transport Layer Security (2-way TLS)



Configuring 2-way TLS from client to Edge (Public Cloud)

01

Server keystore

Create a keystore and upload certificate and private key of the server

02

Truststore

If the client uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain of the client certificate.

03

Virtualhost

Create a support ticket so the virtualhost is created with the suitable configuration.

```
<VirtualHost name="myTLSVHost">
  <HostAliases>
    <HostAlias>apiTLS.myCompany.com</HostAlias>
  </HostAliases>
  <Port>443</Port>
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>false</
ClientAuthEnabled>
    <KeyStore>myTestKeystore</KeyStore>
    <KeyAlias>myKeyAlias</KeyAlias>
  </SSLInfo>
</VirtualHost>
```

HTTP persistent connections

- HTTP 1.0 connections are not persistent
    - Use Connection: Keep-Alive header
- HTTP 1.1 connections are persistent by default
    - Client or server can send Connection: close header to tear down a connection
- Connection establishment and teardown are relatively expensive
- For TLS, we want to use a persistent connection if more traffic is likely to come from the client
    - For Edge to backend, we almost always want a persistent connection
- Traffic from all clients generally flow to the same few targets, so connection is likely to be reused quickly

Virtual host SSL configuration

- Used to differentiate incoming traffic
- Configured on Edge
- Only for client requests, not target communication
- For public cloud, virtual hosts can only be configured by Edge Support

```
GET https://api.enterprise.apigee.com/v1/o/{org}/e/{env}/virtualhosts/secure

 {
   "hostAliases" : [ "myorg-prod.apigee.net" ],
   "interfaces" : [],
   "name" : "secure",
   "port" : "443",
   "sSLInfo" : {
     "ciphers" : [],
     "clientAuthEnabled" : true,
     "enabled" : true,
     "ignoreValidationErrors" : false,

     "keyAlias" : "myKey",
     "keyStore" : "myKeystore",

     "protocols" : [],
     "trustStore" : "myTruststore"
   }
 }
```

Securing calls to the backend

- Generally the backend is locked down to only allow calls from Edge
  - Don't want apps to be able to call directly to backend
- Options for securing the communication to the backend
  - Credentials
  - OAuth (adds significant complexity to backend calls)
  - Two-way TLS
  - IP Whitelisting (can be spoofed)

Securing backend communication with 2-Way TLS

- Obtain/generate client certificate for Edge
- Create and populate a keystore on Edge containing Edge's cert and private key
- Create and populate a truststore on Edge containing trusted certs
- Configure the TargetEndpoint or TargetServer

Configuring 2-way TLS from Edge to target

01

Client keystore

Create a keystore and upload certificate and private key (These are typically supplied by the target system) using management API.

02

Truststore

If the backend server uses a self-signed certificate, or a certificate that is not signed by a trusted CA, create a truststore on Edge that contains the CA chain that you received from the backend server using management API.

03

Target server

Create the target server in the Edge UI with the suitable configuration using management API.

```
<TargetServer name="target1">
  <SSLInfo>
    <Enabled>true</Enabled>
    <ClientAuthEnabled>true</ClientAuthEnabled>
    <KeyAlias>myKeystore</KeyAlias>
    <KeyStore>myKey</KeyStore>
    <TrustStore>myTrustStore</TrustStore>
  </SSLInfo>
</TargetServer>
```

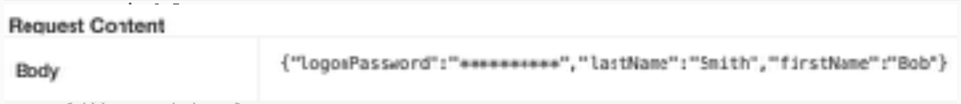
Data masking

- Edge's trace tool allows developers to capture runtime traffic
- Some of the data exposed by the trace tool may be sensitive information, such as passwords, credit card numbers, or personal health information
- To filter this data out of the captured trace information, Edge provides data masking
- Data masking can block values in XML payloads, JSON payloads, and variables
- Data masking configurations can be set
  - globally for an organization
    - `POST /v1/o/{org}/maskconfigs`
  - or on specific apis

Using mask configurations

- XML payloads: Using XPath, you identify XML elements to be filtered from request or response message payloads.
- JSON payloads: Using JSONPath, you identify JSON properties to be filtered from request or response message payloads.
- Flow variables: You can specify a list of variables that should be masked in debug output. When you specify the request.content, response.content, or message.content flow variables, the request/response body is also masked.

```
<MaskDataConfiguration name="default">
  <XPathsRequest>
           <XPathRequest>/apigee:Greeting/apigee:User</
 XPathRequest>
  </XPathsRequest>
  <XPathsResponse>
    <XPathResponse>/apigee:Greeting/apigee:User</
 XPathResponse>
  </XPathsResponse>
  <JSONPathsRequest>
    <JSONPathRequest>$.store.book[*].author</
 JSONPathRequest>
  </JSONPathsRequest>
  <JSONPathsResponse>
<JSONPathResponse>$.store.book[*].author</
 JSONPathResponse>
  </JSONPathsResponse>
  <XPathsFault>
           <XPathFault>/apigee:Greeting/apigee:User</
 XPathFault>
  </XPathsFault>
  <JSONPathsFault>
           <JSONPathFault>$.store.book[*].author</
 JSONPathFault>
  </JSONPathsFault>
```



Request Content

Body	{"logonPassword":"**********","lastName":"Smith","firstName":"Bob"}

```
  </Variables>
 </MaskDataConfiguration>
```

Network level security using Access Control policy

IP Whitelisting / Blacklisting using AccessControl policy

```
<AccessControl name="ACL">
  <IPRules noRuleMatchAction="ALLOW">
    <MatchRule action="DENY">
      <SourceAddress mask="32">10.10.10.10</
SourceAddress>
    </MatchRule>
  </IPRules>
</AccessControl>
```

Denies all client request from 10.10.10.10 and allows others. More info and config documented here

THANK YOU