Implementing Mutual SSL Authentication

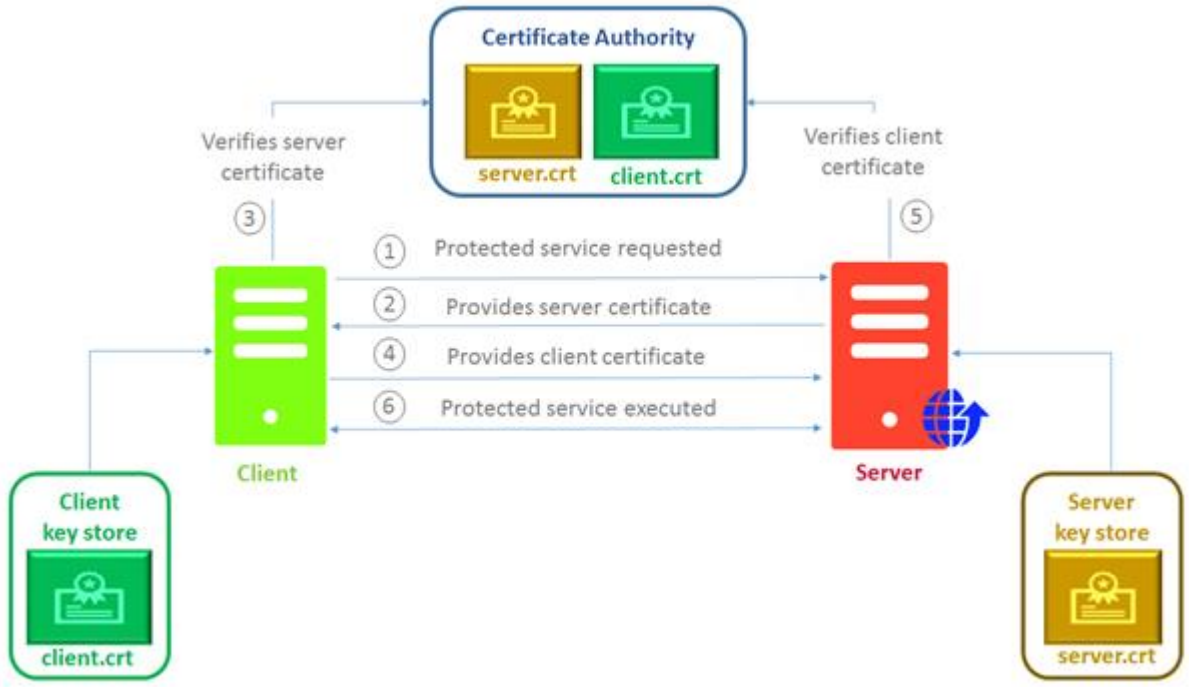
Below listed content lists about what is mutual SSL authentication and how to implement it using a sample NodeJS application.

About Mutual Authentication

HTTPS servers do a basic Transport Layer Security (TLS) handshake and accept any client connection. But what the server can do is challenge the client to provide a certificate during the TLS handshake. It will force the client to present a valid certificate before the service of the server could be accessed. Basic SSL authentication is a one way SSL authentication of server providing the client its own identity by sending the server certificate. On the other hand, mutual SSL authentication is the concept of two parties authenticating each other at the same time. So the client has to provide its own identity to the server and in the same manner server must provide its own identity to the client

How it works

In basic SSL authentication the certificate presented by the server is used by the client to verify it against its trusted certificate authorities. So once the client verifies that the CA is authentic it verifies the server certificate. Specific to mutual SSL authentication using digital certificates, both server and client verify each other. This way the encrypted channel is established between server and client.



Both server and client need to get their certificates issued by a trusted certificate authority. Below listed steps, provides an encrypted channel and process to authenticate, using digital certificates.

- A protected service present on a server is requested by client
- Server provides its certificate (i.e. signed by a certificate authority) to the client
- The server's certificate is verified by the client

- Client then sends its certificate (i.e. signed by a certificate authority)to the server
- Verification of client certificate is done by server

Once the verification is successful, client is able to execute the protected service

Implementation Details

Below listed are the steps of creating certificates, build an HTTPS server and client to use them. The client server code is implemented using NodeJS.

1. Installation of “openSSL”

I used the below link to download and install “openSSL” on my windows machine.

http://gnuwin32.sourceforge.net/packages/openssl.htm

2. One has to generate a signing key specific to the certificate authority. This will be used to sign other certificates. This is for performing self-signing for testing purpose. One has to specific the name of the output file and validity period. Here it is mentioned as 365 days.

Execute this command

```
opensslreq -new -x509 -days 365 -keyout ca-key.pem -out ca-
crt.pem
```

Output

It generates a 2048 bit RSAprivate key. The key is generated on the provided file name. While generating the key below listed questions are asked. Here I have listed the answers applied by me specific to this example.

```
Generating a 2048 bit RSA private key
.+++
.....................+++
writing new private key to 'ca-key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:IN
State or Province Name (full name) [Some-State]:Delhi
Locality Name (eg, city) []:New Delhi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ABC
Organizational Unit Name (eg, section) []:Admin
Common Name (e.g. server FQDN or YOUR name) []:abc.com
Email Address []:admin@abc.com
```

3. Generate Signed Server Certificate

One has to generate the private key of the server. After that using the private key, the certificate signing request has to be generated. Lastly using CA’s private key the certificate has to be signed and generated

Execute this command to generate a private key for the server

```
opensslgenrsa -out server-key.pem 4096
```

Execute this command to generate a signing request

```
opensslreq -new -sha256 -key server-key.pem -out server-csr.pem
```

Output

While generating the certificate signing request below listed questions are asked. I have listed the answers which I have applied specific to this sample.

```
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: IN
State or Province Name (full name) [Some-State]: Delhi
Locality Name (eg, city) []: New Delhi
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
TEST-INFO-CORP
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:
testmutualauthapi.com
Email Address []: admin@test-info-corp.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: mypassword
An optional company name []:
```

Note: *In the above mentioned form the common name is applied as "testmutualauthapi.com". As am using a local windows machine so to use this common name I have modified the windows hosts file with below listed entry.*
127.0.0.1 testmutualauthapi.com
If you don't want to do it then just apply localhost in common name field while filling the form.

Execute this command to perform the signing

```
openssl x509 -req -days 365 -in server-csr.pem -CA ca-crt.pem -
CAkey ca-key.pem -CAcreateserial -out server-crt.pem
```

Output

The result below lists that the signing of the certificate performed by the CA. While it asks the pass phrase which is required to apply the private key of the CA. Below it is listing the information which is embedded in the certificate

```
Signature ok
subject=/C=IN/ST=Delhi/L=New Delhi/O=TEST-INFO-
CORP/OU=IT/CN=testmutualauthapi.com/emailAddress=admin@test-
info-corp.com
Getting CA Private Key
Enter pass phrase for ca-key.pem
```

4. Generate signed client specific certificates

One has to generate the private key of the client. After that using the private key, the certificate signing request has to be generated. Lastly using CA's private key, the certificate has to be signed and generated

- Generating certificate for first client

Execute this command to generate a private key for the client

```
opensslgenrsa -out clientA-key.pem 4096
```

Execute this command to generate a signing request

```
openssl req -new -sha256 -key clientA-key.pem -out clientA-
csr.pem
```

Output

While generating certificate signing requestfor “clientA” below listed questions are asked. I have listed the answers which I have applied specific to this sample for ca-key.pem.

```
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: IN
State or Province Name (full name) [Some-State]: WB
Locality Name (eg, city) []: Kolkata
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
TEST-INFO-CLIENTA
Organizational Unit Name (eg, section) []: IT
Common Name (e.g. server FQDN or YOUR name) []: clienta.com
Email Address []: admin@clienta.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: mypassword
An optional company name []:
```

Execute this command to perform the signing

```
openssl x509 -req -days 365 -in clientA-csr.pem -CA ca-crt.pem -
CAkey ca-key.pem -CAcreateserial -out clientA-crt.pem
```

Output

```
Signature ok
subject=/C=IN/ST=WB/L=Kolkata/O=TEST-INFO-
CLIENTA/OU=IT/CN=clienta.com/emailAddress=admin@clienta.com
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

· Generating for second client

Execute this command to generate a private key for the second client

```
opensslgenrsa -out clientB-key.pem 4096
```

Execute this command to generate a signing request

```
openssl req -new -sha256 -key clientB-key.pem -out clientB-csr.pem
```

Output

While generating the certificate signing requestfor “clientB” below listed questions are asked. I have listed the answers which I have applied specific to this sample.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]: IN
State or Province Name (full name) [Some-State]: Punjab
Locality Name (eg, city) []: Chandigarh
Organization Name (eg, company) [Internet Widgits Pty Ltd]: TEST-INFO-CLIENTB
Organizational Unit Name (eg, section) []: IT
Common Name (e.g. server FQDN or YOUR name) []: clientb.com
Email Address []: admin@clientb.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: mypassword
An optional company name []:
```

Execute this command to perform the signing

```
openssl x509 -req -days 365 -in clientB-csr.pem -CA ca-crt.pem -
CAkey ca-key.pem -CAcreateserial -out clientB-crt.pem
```

Output

The result below lists that the signing of the certificate performed by the CA. While it asks the pass phrase which is required to apply the private key of the CA. Below it is listing the information which is embedded in the certificate.

```
Signature ok
subject=/C=IN/ST=Punjab/L=Chandigarh/O=TEST-INFO-
CLIENTB/OU=IT/CN=clientb.com/emailAddress=admin@clientb.com
Getting CA Private Key
Enter pass phrase for ca-key.pem:
```

5. Server side code to setup https using NodeJS application

Below code shows how to create a HTTPS server. The server side code expects to first provide server key file, server certificate and the certificate of the certificate authority. It also specifies that for requesting a server resource client has to send the client certificate as well. The option “rejectUnanuthoried” specifies whether to reject the request if certificate is not provided by the client. When client invokes the service the checkAuth method is invoked where the client certificate is verified based on the common name.

```
varclientCertificateAuth = require('client-certificate-auth');
var https = require('https');
var opts = {
// Specify the key file for the server
key: fs.readFileSync('cert/server-key.pem'),
// Specify the certificate file
cert: fs.readFileSync('cert/test/server-crt.pem'),
// Specify the Certificate Authority certificate
ca: fs.readFileSync('cert/ca-crt.pem'),
// Requesting the client to provide a certificate, to
authenticate the user.
requestCert: true,
```

```
// As specified as "true", so no unauthenticated traffic
// will make it to the specified route specified
rejectUnauthorized: true
};
varcheckAuth = function(cert) {
// Here one can apply any business logic based on the
certificate details received
// In this code access is only allowed to Client A.
console.log('Client Certificate: ',cert);
console.log('Client Certificate Common Name: '+cert.subject.CN);
return cert.subject.CN === 'clienta.com';
};
var app = express();
app.use(clientCertificateAuth(checkAuth));
// Create this rest service on the server to provide the welcome
message based on the provided name
vartrouter = require('userController.js');
router.get("/greetUser/:userName", trouter.greetUser);
// Service defined in controller.js
module.exports = {
greetUser :greetUser
};
// Method to greet a user
function greetUser(request, response) {
var result={};
varuserName = request.params.userName;
result.message="Hey "+ userName +"! Howz it going...";
response.status(200).send(result);
return result;
}
```

6. Client side code to invoke the NodeJS server application service

Below listed is the NodeJS client side code to connect to the server and request the service at server end. The client side code expects to first provide client key file, client certificate and the certificate of the certificate authority. Other than that it specifies the service which is to be invoked. Below code lists the invocation of the “greetUser” service by multiple clients.

```
var options1 = {
host: 'testmutualauthapi.com',
port: 3005,
```

```
path: '/greetUser/ClientA',
method: 'GET',
key: fs.readFileSync("cert/clientA-key.pem"),
cert: fs.readFileSync("cert/clientA-crt.pem"),
ca: fs.readFileSync("cert/ca-crt.pem")
};
var options2 = {
host: 'testmutualauthapi.com',
port: 3005,
path: '/greetUser/ClientB',
method: 'GET',
key: fs.readFileSync("cert/clientB-key.pem"),
cert: fs.readFileSync("cert/clientB-crt.pem"),
ca: fs.readFileSync("cert/ca-crt.pem")
};
var req1 = https.request(options1, function(res) {
console.log("Client A statusCode: ", res.statusCode);
console.log("Client A headers: ", res.headers);
console.log("Server Host Name: "+
res.connection.getPeerCertificate().subject.CN);
res.on('data', function(d) {
process.stdout.write(d);
});
});
req1.end();
req1.on('error', function(e) {
console.error(e);
});
var req2 = https.request(options2, function(res) {
console.log("Client B statusCode: ", res.statusCode);
console.log("Client B headers: ", res.headers);
// Using this code at client end one can fetch the server
certificate and get all the values provided in it.
// Here we are printing the common name
console.log("Server Host Name: "+
res.connection.getPeerCertificate().subject.CN);
res.on('data', function(d) {
process.stdout.write(d);
});
});
req2.end();
req2.on('error', function(e) {
console.error(e);
});
```

7. Client code execution results

Below listed is the output of the execution of the client code. Client A's invocation shows that it is getting the required message back from the server. But client B's invocation shows that the request is rejected by the server. On checking the server side code you can find that it is done based on the common name of the provided client certificate.

```
Client A statusCode:  200
Client A headers:  { 'x-powered-by': 'Express',
'access-control-allow-origin': '*',
'content-type': 'application/json; charset=utf-8',
'content-length': '43',
etag: 'W/"2b-gWIqTUhkIj3YobcoObubzZO6Cb4"',
'set-cookie': [
'connect.sid=s%3AiSHMlwO3gTID42s16J06Bi3GMNXD0tWi.oRnY956YFxOh9N
9nPBxYpyWWVf59PqvgAfsd1zVh5aE; Path=/; HttpOnly' ],
date: 'Wed, 11 Oct 2017 05:55:43 GMT',
connection: 'close' }
Server Host Name: testmutualauthapi.com
{"message":"HeyClientA! Howz it going..."}
Client B statusCode:  500
Client B headers:  { 'x-powered-by': 'Express',
'access-control-allow-origin': '*',
'content-type': 'application/json; charset=utf-8',
'content-length': '26',
etag: 'W/"1a-pljHtlo127JYJR4E/RYOPb6ucbw"',
date: 'Wed, 11 Oct 2017 05:56:24 GMT',
connection: 'close' }
Server Host Name: testmutualauthapi.com
{"message":"Unauthorized"}
```

8. Server side logs

Below listed are few of the details of client certificate details listed at server end while parsing the request. It lists both the details of the client and the certificate authority who issued the certificate

```
Client A Certificate
{ C: 'IN',
ST: 'WB',
L: 'Kolkata',
O: 'TEST-INFO-CLIENTA',
```

```
OU: 'IT',
CN: 'clienta.com',
emailAddress: 'admin@clienta.com' },
issuer:
{ C: 'IN',
ST: 'Delhi',
L: 'New Delhi',
O: 'ABC',
OU: 'Admin',
CN: 'abc.com',
emailAddress: 'admin@abc.com' },
valid_from: 'Oct 11 05:32:20 2017 GMT',
valid_to: 'Oct 11 05:32:20 2018 GMT',
....
}
Client B Certificate
{ subject:
{ C: 'IN',
ST: 'Punjab',
L: 'Chandigarh',
O: 'TEST-INFO-CLIENTB',
OU: 'IT',
CN: 'clientb.com',
emailAddress: 'admin@clientb.com' },
issuer:
{ C: 'IN',
ST: 'Delhi',
L: 'New Delhi',
O: 'ABC',
OU: 'Admin',
CN: 'abc.com',
emailAddress: 'admin@abc.com' },
valid_from: 'Oct 11 05:37:34 2017 GMT',
valid_to: 'Oct 11 05:37:34 2018 GMT',
....
}
```