SSL connections for HIE implementation

Version 1.0

By ViCarePlus Team

Prepared by

ViSolve Inc.,
Contact: 408.666.4320
EMail: vicareplus_engg@visolve.com
www.vicareplus.com

21st January 2010

Table of contents

1. SSL Background

First some background on SSL/TLS and certificates.
A certificate has:
- An RSA public and private key
- A subject name
- Dates where the certificate is valid.
- An issuer (the subject name of the Certificate Authority)
- The issuer signature
- A serial number (used by issuer to identify certificates)

The main property of the certificate is the subject name.
For example, google's certificate subject is:
subject= /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
The most important part is the CN (common name). Browsers will usually check if the certificate CN matches the website you are visiting. For client certificates, the CN is usually a username (myuser) or an email addres (myuser@machine.com). A certificate without an issuer is called a certificate request. It is not really valid. A certificate must be signed by an issuer in order to be valid. A certificate can be signed by itself (the issuer and subject name are the same). The private key can be stored in a separate file or the same file as the other parts (public key, subject, certificate). In the examples below, OpenEMR will not have access to the CertificateAuthority private key; only the public parts will be in the file.

2. Creating the certificates

The ITI-19 Secure Connection requires 3 certificates:
- A Certificate Authority
- A SSL certificate for the HIE
- A SSL certificate for OpenEMR

The HIE will likely provide the Certificate Authority, but for testing purpose, we will create our own. We will use the openssl command line tool, which is located at

```
/usr/bin/openssl                (Unix/Linux)
C:/xampp/apache/bin/openssl.exe (Windows)
```

To view a list of openssl commands:

```
# openssl --help
```

To view options for a specific command:

```
# openssl <command> --help
# openssl req --help
# openssl x509 --help
```

Create the CA certificate and key. This certificate is "self-signed", in that the issuer of the certificate is itself.

```
# openssl req -new -x509 -nodes -subj "/C=US/CN=CertificateAuthority" -newkey rsa:1024 -keyout cakey.pem -out cacert.pem
```

Create the HIE certificate request and key.

```
# openssl req -new -nodes -subj "/C=US/CN=hie@machine.com" -newkey rsa:1024 -keyout hie.key -out hie.req
```

Sign the HIE certificate with the Certificate Authority.

```
# openssl x509 -days 365 -CA cacert.pem -CAkey cakey.pem -set_serial 10 -req -in hie.req -out hiecert.pem
# rm -f hie.req
```

Create the openemr certificate request and key

```
# openssl req -new -nodes -subj "/C=US/CN=openemr@localhost" -newkey rsa:1024 -keyout openemr.key -out openemr.req
```

Sign the OpenEMR certificate with the Certificate Authority

```
# openssl x509 -days 365 -CA cacert.pem -CAkey cakey.pem -set_serial 11 -req -in openemr.req -out openemrcert.pem
```

Combine the private key and certificate into a single file.
We must combine the files because the PHP functions require it.

```
# cat openemrcert.pem openemr.key > openemrcertkey.pem
# rm -f openemr.req
```

We now have the following files:

cacert.pem	- A self-signed CA certificate
cakey.pem	- The private key for cacert.pem
hiecert.pem	- An HIE server certificate signed by cacert.pem.
hie.key	- The private key for hie.pem
openemrcertkey.pem	- Has both the OpenEMR certificate and key

3. PHP code to create an SSL connection

For auditing, we have created the following function to create the SSL/TLS connection. This is located in file openemr/library/log.inc in our trunk:
https://openmedsoftware.org:10443/repos/OpenEMR/members/visolve/trunk

We can move/modify this function as needed to work with your HIE changes.

```
/* Create a TLS (SSLv3) connection to the given host/port.
* $localcert is the path to a PEM file with a client certificate and private key.
* $cafile is the path to the CA certificate file, for
*  authenticating the remote machine's certificate.
* If $cafile is "", the remote machine's certificate is not verified.
* If $localcert is "", we don't pass a client certificate in the connection.
*
* Return a stream resource that can be used with fwrite(), fread(), etc.
* Returns FALSE on error.
*/
function create_tls_conn($host, $port, $localcert, $cafile) {
  $sslopts = array();
  if ($cafile !== null && $cafile != "") {
      $sslopts['cafile'] = $cafile;
      $sslopts['verify_peer'] = TRUE;
      $sslopts['verify_depth'] = 10;
  }
  if ($localcert !== null && $localcert != "") {
      $sslopts['local_cert'] = $localcert;
  }
  $opts = array('tls' => $sslopts, 'ssl' => $sslopts);
  $ctx = stream_context_create($opts);
  $timeout = 60;
  $flags = STREAM_CLIENT_CONNECT;
  $olderr = error_reporting(0);
  $conn = stream_socket_client('tls://' . $host . ":" . $port, $errno, $errstr,
                              $timeout, $flags, $ctx);
  error_reporting($olderr);
  return $conn;
}
```

Sample usage

```
$conn = create_tls_conn("127.0.0.1", 6514, "openemrcertkey.pem", "cacert.pem");
if ($conn === FALSE) {
  echo("create_tls_conn failed\n");
}
else {
  fwrite($conn, "hello, this is a message\n");
  fclose($conn);
}
```

For reference, the PHP documentation on SSL functions is at
http://www.php.net/manual/en/transports.inet.php
http://www.php.net/manual/en/context.ssl.php
http://www.php.net/manual/en/function.stream-socket-client.php
http://www.php.net/manual/en/function.stream-context-create.php

4. Running an SSL/TLS server

You can run an SSL/TLS server using the openssl tool.
To see all the openssl server options, do:

```
# openssl s_server --help
```

We will use the following options:

```
# openssl s_server -tls1 -verify 10 -CAfile cacert.pem \
-key hie.key -cert hiecert.pem  -accept 6514
```

When a new client connection is accepted, openssl prints out the subject name and issuer name. It verifies the signature of the client against the cacert.pem certificate. If an error occurs, it prints "verify error". Also, the cipher/hash used in the connection is printed out.

For example:

```
# openssl s_server -tls1 -verify 10 -CAfile cacert.pem \
-key hie.key -cert hiecert.pem  -accept 6514
ACCEPT
depth=1 /C=US/CN=CertificateAuthority
verify return:1
depth=0 /C=US/CN=openemr@localhost
verify return:1
-----BEGIN CERTIFICATE-----
MIIByjCCATMCAQswDQYJKoZIhvcNAQEFBQAwLDELMAkGA1UEBhMCVVMxHTAbBgNV
BAMTFENlcnRpZmljYXRlQXV0aG9yaXR5MB4XDTEwMDExNDAwMTcxNFoXDTExMDEx
NDAwMTcxNFowLzELMAkGA1UEBhMCVVMxIDAeBgNVBAMUF3VzZXJAdmljYXJlLnZp
c29sdmUuY29tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDJqwdSw+MS/t95
juJ3N1ejUwzZB4F8uWdYjreO4KnkhszOr/f2KPifWpMNwudPeeCB0ulbJprF0ZQv
l8AtR4nGJ/mNfivC9LquznNTv+OgSjqCIQPyO6wJ55ABFv5+g2hnhg/lZGCRgDQz
WsvWuAuA0jLw3+HW4Xq6KUngo/3gdwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAJnR
DrqrO4qyJagMS7Igl1C37eT/awjEQVfjDm1E6FXKQcKSfdxkk6/KPEYkpY2dLSd8
3tMDo2tnQOcDLGXsrddlDTpnIJyc+LerCFBgHvbR+sNlvoRNp6RQqu4pU2iOUje0
9ezfpluTN+f0K/k86/bujSPARRFDq+Q5qdJXBBVY
-----END CERTIFICATE-----
subject=/C=US/CN=openemr@localhost
issuer=/C=US/CN=CertificateAuthority
CIPHER is DHE-RSA-AES256-SHA
```