Lecture 10 Programming with openssl

(TLS for tcp, DTLS for udp. DTLS is for Internet of Things and used with CoAP)

1. SSL(Secure Socket Layer)

A protocol for secure connection over internet. Uses a public-key algorithm to generate a session key, which is a symmetric key.

(version hisotry: SSL 1.0, SSL 2.0, SSL 3.0, TLS 1.0(SSL 3.1), TLS 1.1(SSL 3.2), TLS 1.2(SSL 3.3))

1) client->server: client hello

protocol version list (SSLv2, SSLv3, TLSv1, ...)

session ID

cipher suite list

key exchange algorithm list (RSA, Diffie-Hellman, ...)

symmetric key algorithm list (TripleDES, DES, ...)

message digest algorithm list (SHA-1, MD5, ...)

compression algorithm list (PKZip, gzip, ...)

random number (196, 201, 083)

2) server -> client: server hello

protocol version (TLSv1, SSLv3, ...)

session ID

chosen cipher suite

key exchange algorithm (RSA)

symmetric key algorithm (DES)

message digest algorithm (SHA-1)

compression algorithm (PKZip)

random number (823, 495, 127)

3) server -> client: server certificate

4) server -> client : hello done

5) client -> server: pre-master secret (48 bytes)

encRSA((client\_version, random[46]), e\_S)

// e\_S is the public key of the server

Client and Server create 6 symmetric keys based on this pre-master secret. 3 of them will be used for client->server communication; 3 others for server->client communication.

6) server <-> client: finished

encDES(finished\_message, session\_key)

\* OpenSSL toolkit provides:

libssl.a :implementation of SSLv2, SSLv3, TLSv1 to support ssl client/server.

API to use openssl (SSL\_connect, SSL\_accept, SSL\_read, SSL\_write, ..)

libcrypto.a: encryption library and x509 stuff. needed by ssl/tls.

for ciphers, it has

libdes (DES), RC4, RC2, Blowfish, IDEA

for digests, it has

MD5, MK2, SHA, SHA-1, MDC2

for public key, it has

RSA, DSA, Diffie-Hellman

for x509, it has

x509 encoding/decoding routines

openssl: a command line tool

2. Making an ssl client/server in the lab server.

1) Copy openssl-1.0.1f.tar.gz into your directory, and un-tar it to create openssl-1.0.1f directory.

$ pwd

/home/sec/12345

$ cp ../../openssl-1.0.1f.tar.gz .

$ tar xvf openssl-1.0.1f.tar.gz

2) config

$ mkdir openssl

$ cd openssl-1.0.1f

$ ./config no-dso --prefix=/home/sec/12345/openssl --openssldir=/home/sec/12345/openssl

3) compile openssl-1.0.1f library

$ make

will compile openssl library code. libssl.a, libcrypto.a will be created in the top directory (/home/sec/12345/openssl-1.0.1f).

4) Install

4.1) Remove "install\_docs" from Makefile (to reduce compile time) as below.

$ vi Makefile

and change

install: all install\_docs install\_sw

to

install: all install\_sw

4.2) Install

$ make install

will install openssl in /home/sec/12345/openssl.

Now you should have all necessary files in /home/sec/openssl directory.

all files will be installed in / home/sec/12345/openssl

lib will have libssl.a, libcrypto.a

include/openssl will have header files

bin will have openssl binary files

certs will have certificates

private will have private keys

5) Compile application: Compile cli.cpp and serv.cpp in demos directory.

5.1) Go to demos/ssl directory.

$ cd openssl-1.0.1f/demos/ssl

5.2) Generate server rsa key

openssl genrsa –out servkey.pem 1024

5.3) Make a config file servconf.txt and generate server certificate:

vi servconf.txt

........type the contents (refer to myconf.txt in lect8) ...........

openssl req –config servconf.txt –new –x509 –key servkey.pem –out servcert.pem

5.4) Change followings.

serv.cpp:

- change the port number

- change the file name for the certificate and key file

- change the return data type of main() to "int"

- change "size\_t client\_len" to "socklen\_t client\_len"

cli.cpp:

- change the server port number and IP address

- change the return data type of main() to "int"

- include <unistd.h>

- Change ssl version to TLSv1: use "TLSv1\_client\_method()" instead of "SSLv2\_client\_method()" in cli.cpp.

5.5) Compile the client and server

g++ -L/home/sec/12345/openssl/lib -I/home/sec/12345/openssl/include

-fpermissive -o serv serv.cpp -lssl -lcrypto -ldl

g++ -L/home/sec/12345/openssl/lib -I/home/sec/12345/openssl/include

-fpermissive -o cli cli.cpp -lssl -lcrypto -ldl

-L is option for additional library directory and -I for additional header file directory.

-lssl means "link with libssl.a" (for static library) or "link with libssl.so" (for dynamically linked library, or shared object) and -lcrypto "link with libcrypto.a" or "link with libcrypto.so".

5.6). Run the server first and run the client to see if they can talk.

3. Making an ssl client in windows (For MacOS, get openssl-1.0.1f.tar.gz from the lab server via sftp and install following the steps in Section 2)

1) Go to http://www.npcglib.org/~stathis/blog/precompiled-openssl/

Download precompiled openssl libray and header files(one matching your vs version: e.g., openssl-1.1.0f-vs2015.7z[md5]) and uncompress. (You may need to download 7-zip from internet to uncompress.)

2) Make an empty visual studio project and create a c++ source file with the code in lect10-winsslcli.docx.

3) Set lib, include path of this project to the lib, include directory (32 bit version) of the uncompressed openssl.

project>properties>vc++ directories>include directory

project>properties>vc++ directories>library directory

4)Add following 4 libraries in linker>input>additional dependencies:

ws2\_32.lib, libsslMT.lib, Crypt32.lib, libcryptoMT.lib

5)Run the ssl server in the lab machine, run windows ssl client, and see if they can talk.

4. Run the sniffer. Run the server in 165.246.38.151 and run the windows ssl client. Analyze the packets between the client and the server. The format of the TLS packet is in the next section. (Analyze only client hello and server hello)

5. General format of TLS record (refer to " http://en.wikipedia.org/wiki/Secure\_Sockets\_Layer" for detail)

byte 0: content type (RT: Record Type)

0x14 : ChangeCipherSpec

0x15 : Alert

0x16 : Handshake

0x17 : Application Data

0x18 : Heartbeat

byte 1, 2 : Version (byte 1:Major, byte 2: Minor)

SSLv3: 3, 0

TLS 1.0 : 3, 1

TLS 1.1 : 3, 2

TLS 1.2 : 3, 3

byte 3, 4 : Length (byte 3: bits 15..8, byte 4: bits 7..0) -- the length of the protocol message

byte 5..(m-1) : protocol message

byte m..(p-1) : MAC(Message Authentication Code) -- optional

byte p..(q-1) : Padding(block ciphers only)

6. Handshake protocol

byte 0: 0x16

byte 1, 2: Version

byte 3, 4: Length of the protocol message

byte 5 : MT(Message Type)

0 : HelloRequest

1 : ClientHello

2 : ServerHello

11: Certificate

12 : ServerKeyExchange

13: CertificateRequest

14: ServerHelloDone

15: Certificate Verify

16: ClientKeyExchange

20: Finished

byte 6, 7, 8: Handshake message data length(byte 6: bits 23..16, byte 7:bits 15..8, byte 8:bits7..0)

byte 9..(n-1): Handshake message data

byte n..(n+3): message type, handshake message data length (repeat)

byte (n+4).. : handshake message data

...........

7. ClinetHello handshake message data

struct {

ProtocolVersion client\_version; // 2 bytes

Random random; // 4 byte for time, 28 bytes for random number

SessionID session\_id; // 1 byte for length (0 for clientHello). x bytes for id

CipherSuite cipher\_suites<2..2^16-1>; //2 bytes for length. x ciphersuits

CompressionMethod compression\_methods<1..2^8-1>; // 1 byte for length. x method

} ClientHello;

enum { null(0), (255) } CompressionMethod; // CompressionMethod.null is 0 and

// it consumes 1-byte memory. not C/C++.

unit8 CipherSuite[2]; // cryptographic suite selector

byte 1, 2, 3: reserved

opaque SessionID<0..32>;

struct{

uint32 gmt\_unix\_time;

opaque random\_bytes[28]; // 28 bytes

} Random;

struct{

uint8 major, minor;

} ProtocolVersion;

CipherSuite: refer Section A.5 of "http://www.ietf.org/rfc/rfc2246.txt" to see the list of ciphersuites. or http://www.thesprawl.org/research/tls-and-ssl-cipher-suites/

0x00, 0x09: SSL\_RSA\_WITH\_DES\_CBC\_SHA

0x00, 0x0A : SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

0x00, 0x07 : SSL\_RSA\_WITH\_IDEA\_CBC\_SHA

0x00, 0x015 : SSL\_DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

...................

8. ServerHello message data

struct {

ProtocolVersion client\_version; // 2 bytes

Random random; // 4 byte for time, 28 bytes for random number

SessionID session\_id; // 1 byte for length (0 for clientHello). x bytes for id

CipherSuite cipher\_suites; //2 bytes for selected ciphersuit

CompressionMethod compression\_methods; // 1 byte for method

} ServerHello;

9. Alert protocol

Message type: close\_notify, unexpected\_message, ...

10. HeartBeat protocol

Message type HB request, HB response

11. Homework

1) Do the steps in Section 2.

2) Modify cli.cpp such that it displays "Start SSL protocol in client" before it calls SSL\_connect(ssl). Also modify serv.cpp such that it displays "Start SSL protocol in server" before it calls SSL\_acept(ssl). Recompile cli, serv, and rerun them to see the effect.

3) cli.cpp calls SSL\_connect() which in turn calls ssl3\_connect() (defined in openssl-1.0.1f/ssl/s3\_clnt.c). Add

printf("ssl3\_connect begins\n");

in the beginning of ssl3\_connect(). Go to the SSL top directory (openssl-1.0.1f) and recompile ssl library with "make". Re-install ssl library with "make install". Now go to demos/ssl and recompile cli.cpp and serv.cpp and rerun them to see if the client prints "ssl3\_connect begins".

4) serv.cpp calls SSL\_accept() which in turn calls ssl3\_accept() (defined in openssl-1.0.1f/ssl/s3\_srvr.c). Add

printf("ssl3\_accept begins\n");

in the beginning of ssl3\_accept(). Recompile and re-install ssl library. Recompile cli.cpp and serv.cpp and see if the server displays the above message.

5) Modify ssl3\_connect(), ssl3\_accept() such that they print some message at each ssl protocol stage. Recompile ssl libraries, cli, serv, and rerun. Match the state changes in the client and the server with the state changes explained in Section 1.

6) Do the steps in Section 3 and 4.

7) Make the sniffer sniff port 443 which is the default ssl port. Use your web browser to connect to www.wooribank.com and let the sniffer analyze the packets. Analyze only client hello and server hello. (read: http://max.euston.net/d/tip\_ssldump.html)

8) Write a client and server that exchange a DES key through an SSL channel and communicate securely with this DES key.

9) Modify openssl such that the server always asks for a client certificate.

10) Read articles about“heartbleed” attack in the Internet and summarize. Find actual attack incidents also.

https://www.bbc.com/news/technology-45472766

11) openssl-1.0.1f has Heartbleed bug. Modify your ssl client in windows such that it can exploit this bug and obtain the server private key.

12) Modify ssl library such that it can prevent the Heartbleed bug. Do not copy the official fix published in the internet.