6) Do the steps in Section 3 and 4.

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

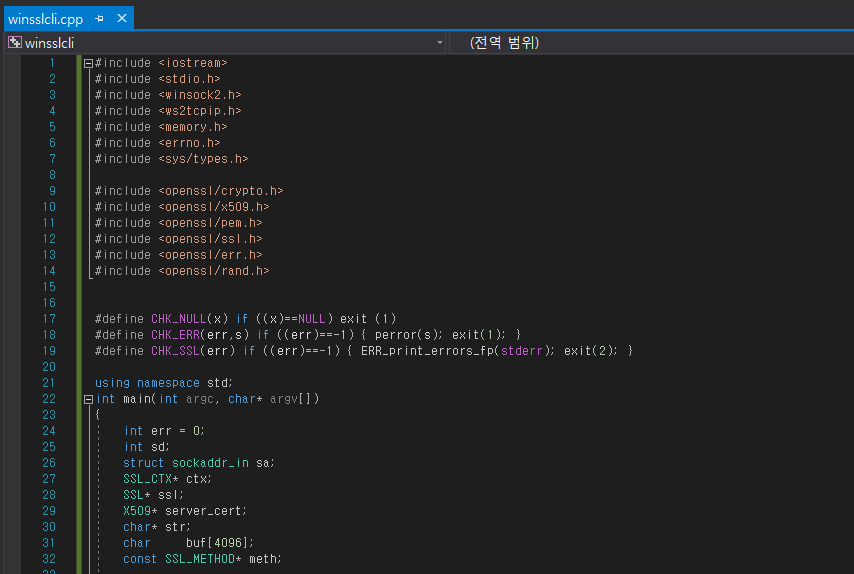
project>properties>vc++ directories>library directory

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

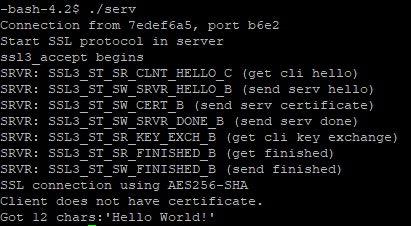
ws2\_32.lib, libsslMT.lib, Crypt32.lib, libcryptoMT.lib

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

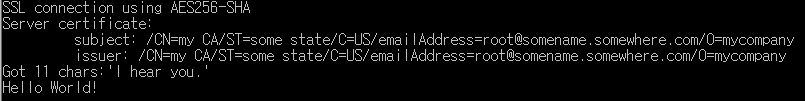
client in window



serv (openssl-1.0.1f/demos/ssl)



cli

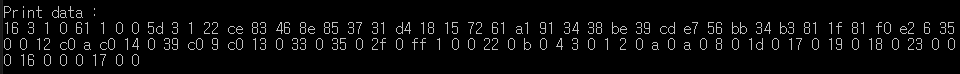


visual studio에 winsslcli를 만들고 서버랑 연결해준 결과, 서버와 클라이언트가 정상적으로 연결돼서 ‘Hello World!’와 ‘I hear you’를 주고받은걸 확인할 수 있었다.

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)

 (내 비쥬얼 스튜디오에서의 스니퍼 위치)

client hello



record type : 16 (Handshake)

version : 3.1 (2bytes)

length of the protocol message : 0x61 (2 bytes)

message type : 1 (Client hello)

message length : 0x5d (이 이후부터 다 message) (3bytes)

version : 3.1

time : 0x22ce8346

random number : 8e853731d418157261a1913438be39cde756bb34b3811f81f0e20635 (28bytes)

session ID length : 0 (1byte) (만약 0이 아니라면 이 길이만큼 session ID들이 있음)

cipher suite length : 0x12 (2bytes)

cipher suit : c00ac0140039c009c01300330035002f00ff (12bytes)

(TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA

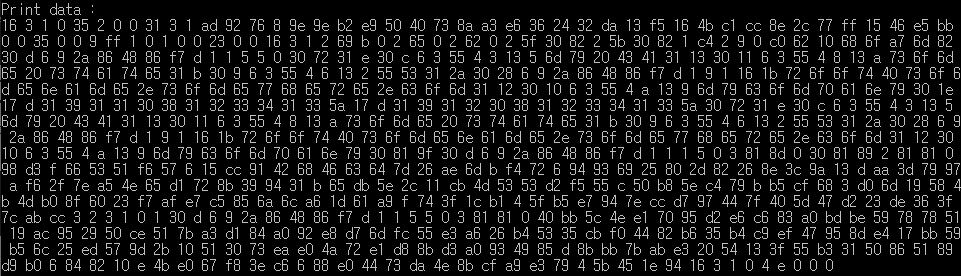
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_EMPTY\_RENEGOTIATION\_INFO\_SCSV)

compression method length : 0x01 (1byte) = one compression algorithm

compression method : 0x00

server hello



record type : 16 (Handshake)

version : 3.1

length of the protocol message : 0x35 (2bytes)

message type : 2 (Server hello) (1byte)

message length : 0x31 (3bytes) (00 00 31)

version : 3.1

time : 0xad927608

random number : 9e9eb2e95040738aa3e6362432da13f5164bc1cc8e2c77ff1546e5bb (28bytes)

session ID length : 0 (1 byte)

selected cipher suite : 0x00, 0x35 (TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA)

selected compression method : 0x00 (여기선 길이 없이 바로 selected compression method 나옴)

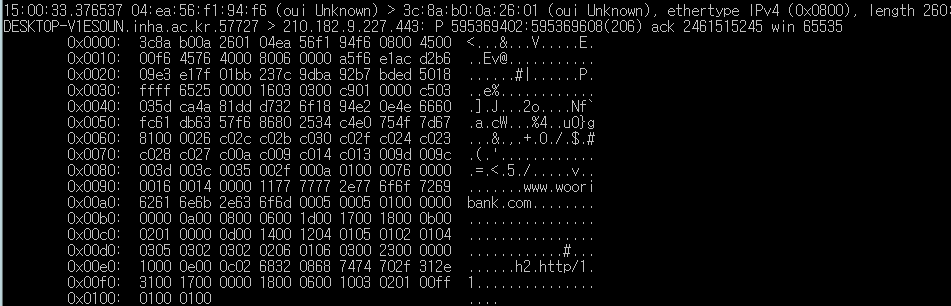
7) Make the sniffer sniff port 443 which is the default ssl port. Use your web browser to connect to www.wooribank.com and let the sniffer analyze the packets. Analyze only client hello and server hello. (read: <http://max.euston.net/d/tip_ssldump.html>)

cmd



(먼저 저 사이트부터 열고 이거 실행)

client hello





record type : 16 (Handshake)

version : 3.3

length of the protocol message : 0xc9

message type : 1 (Client hello)

message length : 0xc5

version : 3.3

time : 0x5dca4a81

random number : ddd7326f1894e20e4e6660fc61db6357f686802534c4e0754f7d6781

session ID length : 0

cipher suite length : 0x26

cipher suit : c02cc02bc030c02fc024c023c028c027c00ac009c014c013009d009c003d003c0035002f 000a

(TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256

TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384

TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256

TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384

TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256

TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA

TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384

TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256

TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256

TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256

TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA

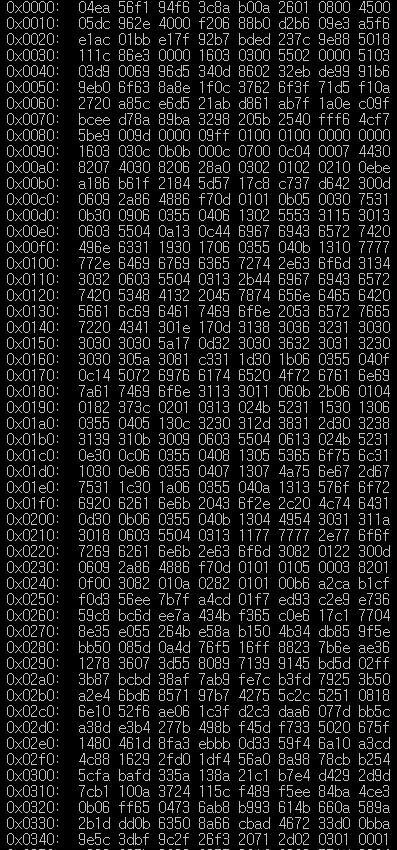
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA

TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA)

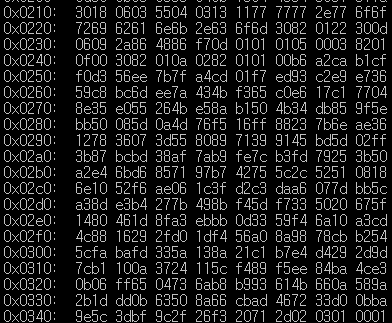
compression method length : 0x01

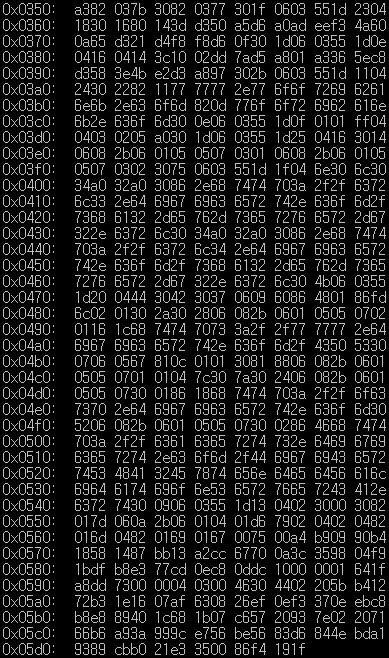
compression method : 0x00

server hello









record type : 16 (Handshake)

version : 3.3

length of the protocol message : 0x55

message type : 2 (Server hello)

message length : 0x51

version : 3.3

time : 0xd9006996

random number : d5340d860232ebde9991b69eb06f638a8e1f0c37626f3f71d6f10a27

session ID length : 0x20

selected cipher suite : 0x00, 0x9d (TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384)

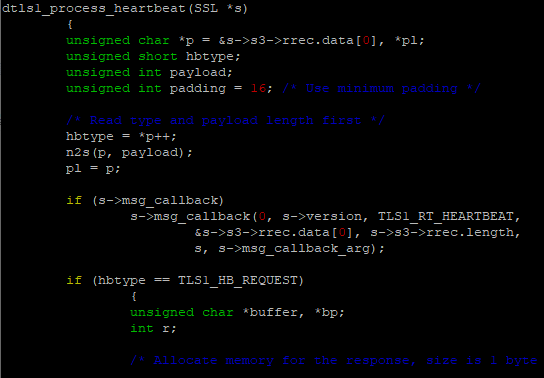
selected compression method : 0x00

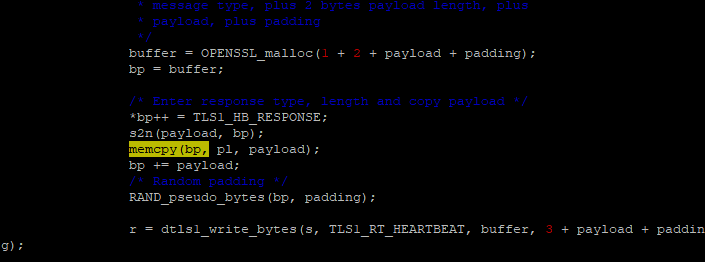
10) Read articles about“heartbleed” attack in the Internet and summarize. Find actual attack incidents also.

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

뉴스에서는 Andrei Tyurin이라는 러시아 사람이 미국의 은행, 경제 회사 등등을 해킹해서 많은 중요한 정보들을 훔쳤고, 이것이 가능하게 된 데에는 heartbleeding을 포함한 많은 테크닉적 트릭들이 관여돼있다는 내용을 다루고 있다. Heartbleeding에 대한 실제 사례로는 해커들이 heartbleed 버그를 이용해서 4.5백만명의 환자의 기록들을 훔쳤다는 사례가 있다.

openssl-1.0.1f/ssl/t1\_lib.c





Heartbleeding은 process\_heartbeat라는 함수에서 일어나고, memcpy(bp, p1, payload); 라는 단 한줄 때문에 발생하는 버그이다. Heartbeat라는 것은 유저가 아무것도 안하고 있을 때에도 클라이언트와 서버가 서로 연결돼있다는걸 확인하기 위한 일종의 소통 방식이다. 여기에서 컴퓨터는 다른 컴퓨터로부터 heartbeat request의 크기인 payload를 함께 받게 되고, 이 payload는 p1의 크기와 같아야 bp에 정확히 복사되지만, payload와 p1의 크기가 같은지 비교할 수 있는 방법이 없기 때문에 만약 payload의 크기를 p1의 크기보다 크게 줘버린다면, heartbeat request를 받은 컴퓨터는 평소와 같이 payload의 크기만큼 메모리 버퍼를 만들고 그만큼을 다 채우지 않으면 안되기 때문에 그정도의 공간에 실제로 받은 p1과 컴퓨터 안에 있던 정보들을 추가적으로 함께 보내주게 되기 때문에 attacker들은 그 추가 정보들을 얻을 수 있게 된다.