COMP 4337/9337 - Securing Wireless Networks Implementation Assignment (V1.0)

April 28, 2016

Change Log a. Version 1.0 released on April 29th, 2016

Due Date: 30th May 2016, 11:59 pm
Total Marks: 20 (towards 45 marks for lab/project component in the course outline)

1 Learning Objectives:

On completing this assignment you will gain sufficient expertise in the following skills:

- Understanding and developing security protocols.
- Reading and understanding standard documents for security such as RFCs.

2 General Instructions

This document contains the project specifications for COMP 4337/9337 - Securing Wireless Networks. Please note the followings:

- *Equal marking:* It is assumed that all group members contribute equally to a submitted work. Therefore, the project marks will be applied to all group members equally.
- *Grade distributions:* This project contributes to a maximum of 20 marks towards your final grade. Your marks are based on the quality

of your submitted work. We expect that you work in a group of two. If you are not able to find a partner, please contact LIC as soon as possible (after you have tried via openlearning). By default we will assume that you have retained your research project group unless notified otherwise by 5 pm, 4th May. Also, please note that each group can only submit one project.

- **Alternate Project:** *Groups can also suggest an alternate project. They must send a brief description (i.e., what will be built, why is this worthwhile and how it relates to this subject, and what will be demonstrated) to the LIC via email latest by 3 May, 2016. LIC will approve this or seek further clarification during the lecture on May 4, 2016. The project must be of similar or higher complexity in building a secure system over wired/wireless network. You can use publicly available programming modules (but identify them in your report and demo). Project should not duplicate work done in another course e.g. 3441*

- *Programming Language Support:* We recommend that you use Python 2.7 for this project. You may choose to work in other programming languages as libraries may be availabe publicly. However, please note that there will be no support from our side if your choose another programming languages.

- *Consultation and queries support:* Please discuss general questions via openlearning. We will keep an eye. If you have specific questions, please contact Dr Hailun Tan (thailun@cse.unsw.edu.au) as he will be in-charge of this assignment. He will also advertise a regular consultation slot.

3 Project Scope

We will implement a simplified version of SSL/TLS (miniSSL) and a simplified application-layer protocol (miniGet) in this project. The goal is to get a better understanding of the SSL/TLS protocols.

miniSSL and miniGET

An miniSSL is a barebone version of Secure Socket Layer (SSL). An miniSSL includes a simplified SSL handshake, which leads for user authentication.

A session key need to be distributed securely for data encryption and integrity check (by HMAC) in miniGet later. You are expected to use RSA and implement the client authentication (advanced requirement: mutual authentication).

An miniGET is simply a `GET` operation sent from client to server after the miniSSL hand-shake. The server sends the requested file, encrypted and HMACed with the session key. The session key for data encryption and integrity check was established during the miniSSL handshake. We always use the off-the-shelf cryptographic algorithms, AES 128-bit encryption and SHA1-based HMAC, for data encryption and integrity check. Once the file is delivered, both client and server would terminate the session without any further notifications.

Protocol flow

The simplified protocol you are going to implement is shown as follows. C denotes as client, S denotes as server. We have an miniSSL-CA.

We start with the miniSSL handshake. Note that we group the different message types of SSL/TLS in new, custom ones. In the following, the comma operator indicates the border between message fields, and the `|` operator indicates concatenation of two (bit) strings.

1. C chooses a nonce n_c of length 28 bytes. It chooses the cipher suite to be 128-bit AES encryption and SHA1 as the HMAC function. It is denoted as a string `AES-HMAC`. The type of the first message is `ClientInit`. C sends this to S:

   $C \rightarrow S$: `ClientInit`, $\texttt{n}_\texttt{c}$, `AES-HMAC`

2. Upon receiving this message, S chooses a random nonce n_s of length 28 bytes. It acknowledges the client's cipher choice and also sends a certificate. It can add an optional request for the client to authenticate with a certificate by sending the `CertRequest` string:
   $S \rightarrow C$: `ServerInit`, $\texttt{n}_\texttt{s}$, `AES-HMAC`, $\texttt{Cert}_\texttt{s}$ [, `CertReq`]

3. C verifies that the unexpired server certificate was issued by the miniSSL-CA (checking the signature, with the certificates in its root store) and whether the common name matches with the expected one. It extracts S's public key. It generates a pre-master secret p as a random value with 46 bytes in length. From this, it derives two session keys k_1 and k_2 as k_1=`HMAC-SHA1`$(p, n_c|n_s|\texttt{enc})$ and k_2=`HMAC-SHA1`$(p, n_c|n_s|\texttt{mac})$, with

`enc,mac` being the binary strings `00000000` and `11111111`. Finally, it computes an HMAC over all messages up to this point in the following way:
m_c=`HMAC-SHA1`(k_2,`ClientInit`—$\texttt{n}_\texttt{c}$—`AES-HMAC`—`ServerInit`—$\texttt{n}_\texttt{s}$—$\texttt{Cert}_\texttt{s}$ [|`CertReq`])

C sends the following to the server (note the optional client certificate). E means encryption with the respective public key:
$C \rightarrow S$: `ClientKex`, $E_S(p), m_c[, \texttt{Cert}_\texttt{C}, \texttt{Sig}_\texttt{C}(\texttt{n}_\texttt{s}|\texttt{E}_\texttt{S}(\texttt{p}))]$

$Sig_C(n_s)$ is C's signature on the server's nonce.

4. The server, upon receiving this message, also verifies that $Cert_C$ was issued by miniSSL-CA (see above) and that the certificate is not expired. There is no need to check the Common Name. It computes k_1, k_2 and verifies that m_c has the correct value. It computes an HMAC over all messages up to this point in the following way:
m_s=`HMAC-SHA1`(k_2,`ClientKex`—$E_S(p)|m_c|[, \texttt{Cert}_\texttt{C}]$).

   It sends this to C:
   $S \rightarrow C : m_s$

5. C verifies m_s. The handshake is complete. C will now initiate `miniGET`. This protocol uses k_1 for encryption and k_2 for HMACs.

Compulsory Requirements

The following requirements are compulsory for your program.

- Your implementation must allow the protocol to run between client and server on different hosts, with different IP addresses. Therefore, TCP/IP sockets should be specified readily in your program.
- The miniSSL handshake should be implemented with the establishments of two session keys, one for AES data encryption, the other for HMAC integrity check.
- Your program should be able to accommodate multiple sessions, i.e. it must be possible to have two clients communicating with one server at the same time.
- Client and server must record the state of the current handshake, i.e., session management. All cryptographic information in a given protocol (nonces, keys, etc.) must be stored internally! There is an immediate

effect that client and server cannot be confused when a third party starts sending them well-formed messages - they should be able to reject these as the state information is incorrect.

- Your client must verify that the server certificate has been issued from the minissl CA, carries the expected Common Name (it is in the certificate), and is not expired. No other checks are necessary. It must use the public key found in the server certificate.

- Your server must support two modes: Simple and ClientAuth. In Simple, the client does not need to authenticate to the server. In ClientAuth, the server verifies that the client presents a valid certificate from the minissl CA and this certificate is not expired.

- After the handshake, the client must download the text file `payload.txt` from the server. Generate this file yourself. This messages must be encrypted with the session key, with the respective HMAC. In ClientAuth mode, the server could only process the received message if the client has presented the correct certificate. You have liberty of the design and implementations on your own session termination signals in miniGet . We suggest you implement a simple GET and have server and client(s) simply terminate the connection when the file is successfully received.

- By checking the SHA-1 checksum, you must prove that both client and server have identical copies of `payload.txt`.

- If either the server or client presents an invalid certificate, the other side must terminate the connection (i.e., close the socket). There are rogue certificates in the attached project package. Use them to verify that you do not accidentally accept the invalid certificates.

- Your program must handle all the exceptions well without crashing if the packet is misconfigured. In addition, client and server must terminate the session successfully if it happened.

Program Formats

The client should be started as follows:

```
./client.py dst_ip dst_port clientcert clientprivkey
```

The server must be started like this:

`./server.py listen_port servercert serverprivkey {SimpleAuth, ClientAuth} payload.txt`

i.e. server and client read in their certificates and private keys from the provided file.

Design Liberties Except the above compulsory requirements, you have liberties of implementing these functions in various ways. Refer to RFC4346 for potential options. In particular, you are free to choose:

- Protocol field format and field delimiters
- Appropriate encodings for message types and fixed strings, if needed
- The way you do RSA encryption (e.g., padding). Find a way to deal with the problem of plaintexts that are too long.
- You may choose a higher key length, but then you need to choose the string indicating that.
- Blocking or non-blocking sockets
- Details of the file transfer
- We suggest you use ports > 1024 to avoid the administrative rights

Evaluation factors

The following factors will be evaluated for this project.

- Correctness of programs.
- Completion of correct handshake and correct file transmissions.
- The completion of the compulsory requirements .
- Well-commented codes.

References

Sockets Here are some readings on how to use sockets in Python: `http://docs.python.org/2/howto/sockets.html`.

Libraries Under Python, we found that the use of three libraries yields best results (this tells you a lot about crypto for Python):

- pycrypto for simple RSA, AES and HMAC operations, `https://www.dlitz.net/software/pycrypto/api/current/`
- m2crypto for direct verification of a certificate, `http://www.heikkitoivonen.net/m2crypto/api/`
- pyopenssl for reading X.509 fields, `http://packages.python.org/pyOpenSSL/`

Under a Debian or Ubuntu, you can install them on by doing a `sudo apt-get install python-crypto python-m2crypto python-openssl`.

As having to use three crypto libs is awkward, we have coded up some help for you in `keyutils.py`. You'll mostly only need PyCrypto.

Submisson Instructions (TBA)

The code is to be submitted by 30^{th} of May, 2016, 11:59 PM. Submission instruction will be advised a week before the deadline. You will need to demonstrate your implementation during the lab session on 1^{st} of June, 2016. There will be a link to submit your report and codes.

Demonstration Schedule (TBA)

Late Submission Penalty:

Late penalty will be applied as follows:

- 1 day after deadline: 10% reduction
- 2 days after deadline: 20% reduction
- 3 or more days late: NOT accepted

NOTE: The above penalty is applied to your final total. For example, if you submit your assignment 1 day late and your score on the assignment is 20, then your final mark will be 20 2 (10

Plagiarism:

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO and reported to school plagiarism register. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over.