CS 447 : Networks and Data Communications

Programming Assignment #03

Total Points: 150

Assigned Date : Wednesday, April 11, 2018
Due Date : Wednesday, April 25, 2018 @ 11:59:59 a.m. (Strong Deadline)

Overview

For your third (and final) project, you will implement a secure client/server application using TLS (Transport Layer Security). **Note:** This final assignment has a **firm deadline** and does not include the typical 48-hour late penalty period. Given the strong deadline, we'll extend PR01 to create a secure network application. Here's the back story.

Back Story

Haddock was quite happy with the geometric calculator until Thomson and Thompson showed up one day with some devastating news. *"You know Haddock, we have strong suspicions that Rastapopulus has been sniffing all your calculations using* `Wireshark`*, right Thompson?"*, said Thomson. *"Precisely Thomson. In fact, insecure communication means not only eavesdropping but also possible changes to plaintext exchanges during transit too."* replied Thompson.
Haddock ran down the hallway and stormed into Calculus's office. *"Blue blistering barnacles! Calculus"*, screamed Haddock. *"Thompsons are telling me that Rastapopulus is changing my calculations using some sort of a 'Shark'??"*. Calculus calmed Haddock and explained to him that he plans to use Transport Layer Security (TLS) in the next iteration of the calculator to make it even more secure.

Technical Requirements

In order to keep this assignment within the time spec of the project and to accommodate the fact that TLS requires a reliable Trasport Layer protocol, the technical requirements of this assignment are simplified in the following manner.

1. You are only required to develop a secure `client` $\leftrightarrow$ `server` application that supports TCP. You can ignore the UDP interaction. All other PR01 technical requirements applies to PR03, including support for multiple concurrent clients.
2. Before initiating any calculator related interaction, your `client` and the `server` must establish a secure channel using TLS. See Logistics below for more information.
3. After establishing a secure channel, your `client` is now required to login to the server using a password for authentication purposes.

- Client authentication is initiated through a special **`AUTH`** command. This new command effectively eliminates/replaces the **`HELO`** command used earlier.
- The **`AUTH`** command must be the first command in any new `client` $\leftrightarrow$ `server` interaction.
- Following the **`AUTH`** command, the `server` replies back with code **`334 dXNlcm5hbWU6`**[†] prompting the user to enter his/her username. (e.g. `username@447.edu`).
- Once the user sends his username, the `server` then responds back with code **`334 cGFzc3dvcmQ6`**[‡] prompting the user to enter his/her password.
- A successful (or failed) authentication is marked by the appropriate reply code. Read Section 6 of RFC #4954 for status codes and adapt appropriate reply codes from SMTP specifications.

4. To keep the project within the time spec, the following simplified password strategy is proposed.
   (a) The first time the user responds to **`334 dXNlcm5hbWU6`**, the `server` replies back with a 5-digit randomly generated password over reply code **`330`** (instead of code **`334 cGFzc3dvcmQ6`**). The `server` adds 447 to this number, encodes it in base64, and stores the encoded password along with the corresponding username in a hidden master password file named "**`.user_pass`**".
   (b) Upon receiving the 330 code and the temporary password, the `client` immediately terminates the current connection, waits for 5 seconds, and re-initialize a fresh connection.
   (c) On all successive connections, the `client` responds to `server's` request for password, i.e. **`334 cGFzc3dvcmQ6`**, by typing in the previously received password.
   (d) The `server` adds 447 to the received password, encodes it in base64, and checks the encoded value against the stored value in the password file. If these two values match, the authentication phase is successful.
   (e) Your `client` program will take the plaintext `username` and `password` entered by the user and encode it in base64 before sending. In otherwords, the `server` receives base64 encoded `username` and `password`.
5. Your solution must use a `TLSv1` (or newer depending on your language support) negotiation. Refrain from using any of the (old) SSL negotiations.
6. Fix all outstanding issues from your PR01.

Logistics

- All applicable PR01 Logistics requirements applies to PR03. You may ignore any logistical requirements related to UDP.
- Run `Wireshark` on a secure calculator interaction and on an unsecure calculator interaction. Include your observations with appropriately annotated screenshots on your project report. Are you convinced your new program is secure? Comment on your observations.
- The general workflow of SSL integrated socket communication is shown in Figure 1 below.

The nature of this assignment will force you to do significant amount of unsupervised learning – online research, forum scans, trial-and-error, and troubleshooting. Most likely, you will also discover (on your own) that there are more than one library package implementation of `openssl` standard, especially for C/C++, which might further complicate your task. So, don't get frustrated but instead use this as a very valuable learning opportunity. Here are some resources that I believe will help you.

1. **HP SSL Programming Tutorial:**
   http://h41379.www4.hpe.com/doc/83final/ba554_90007/ch04s03.html

[†] `'username' in base64`
[‡] `'password' in base64`

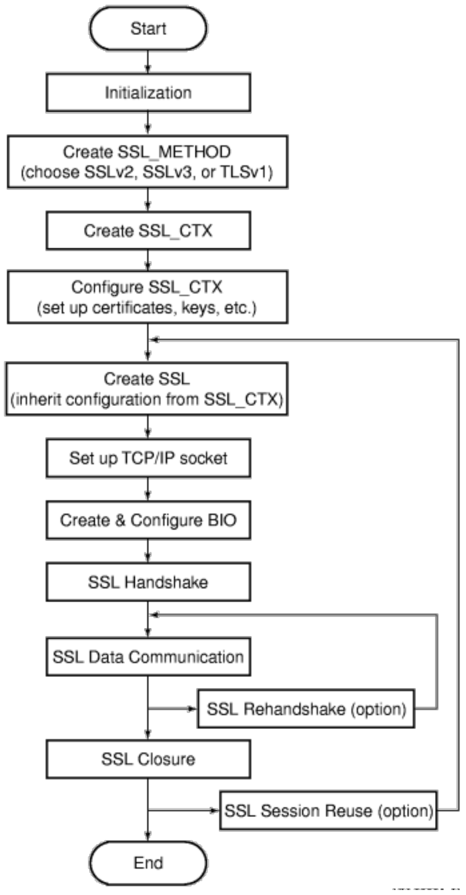
Start
Initialization
Create SSL_METHOD (choose SSLv2, SSLv3, or TLSv1)
Create SSL_CTX
Configure SSL_CTX (set up certificates, keys, etc.)
Create SSL (inherit configuration from SSL_CTX)
Set up TCP/IP socket
Create & Configure BIO
SSL Handshake
SSL Data Communication
SSL Rehandshake (option)
SSL Closure
SSL Session Reuse (option)
End

Figure 1: A Typical SSL Communication Workflow from the HP SSL Programming Tutorial.

2. **Fedora Security Team – Defensive Coding**
   https://docs.fedoraproject.org/en-US/Fedora_Security_Team/1/html/Defensive_Coding/index.html
3. **OpenSSL Wiki – Simple TLS Server:**
   https://wiki.openssl.org/index.php/Simple_TLS_Server
4. **OpenSSL Cookbook:**
   https://www.feistyduck.com/library/openssl-cookbook/online/
5. **JSSE Reference Guide**
   https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html
6. **OWASP – Using the Java Secure Socket Extensions:**
   https://www.owasp.org/index.php/Using_the_Java_Secure_Socket_Extensions
7. http://simplestcodings.blogspot.com/2010/08/secure-server-client-using-openssl-in-c.html
8. http://stilius.net/java/java_ssl.php
9. **pyOpenSSL Documentation**
   https://media.readthedocs.org/pdf/pyopenssl/latest/pyopenssl.pdf

Almost all of these helpful resources also include sample code that might aide you in your overall design and development. Using online resources **does not** mean you are allowed to copy and use someone else's

code for your purpose. Such incidents, if detected, will be treated as academic dishonesty, so please do not copy-paste some else's code in your solution. Instead, try to grasp the core idea behind their solution and weave it in to your own work.
If you find a resource that you think will help your classmates, please share it with me so that I can disperse it among you colleagues.
Extra Credit (20 points): Use a secure hash function instead of base64 encoding when storing salted passwords in the hidden password file.

Instructions

- **Start early!!**
- **Take backups of your code often!!**.
- Follow a good coding standard. Use a Google style guide matching for your favorite programming language found here `https://google.github.io/styleguide/`, if you don't already follow one.
- The due date of this assignment is **Wednesday, April 25, 2018 @ 11:59:59 a.m.** A dropbox will be opened for submission through Moodle. This is a **firm deadline** with no late submission period.

Deliverables

A complete solution comprises of:

- A short report (max 5 pages) in **PDF** format of the design and implementation of your system. Your report should include the following subsections:
  - Introduction
  - Design choices and protocol/reply codes used.
  - The output of a sample run (including screenshots where applicable).
  - Summary and Issues encountered (if applicable).

  Note: Your report will carry a significantly more weight in compared to the other two projects as it will be a reflection of your unsupervised learning efforts. Please make sure to document your efforts and to develop a good project report.
- A short **`readme`** file with compilation instructions. A **`makefile`** is mandatory if your solution involves running multiple compilation instructions.
- A compressed tarball of the directory containing your source code. Since you will be essentially improving your PR01 code base, submit appropriate **`patch files`** instead of full source code. Use `diff` to generate patch files.
- **Do not** include executables, folders created by your programs, or your test emails in this tarball. To create a compressed tarball of the directory source, use the following command: `tar -zcvf siue-id-pr2.tar.gz source/`. e.g. `tar -zcvf tgamage-pr2.tar.gz PR03/`.

Caution: File formatting standards (`PDF`, `readme`, `make`, and `tar.gz`) as well as some of the logistics requirements are set forth to streamline the grading process. Submissions that take a unnecessarily long time grade due to not following the standards listed in this document will be subject to penalties.

Collaborating on ideas or answering questions is always encouraged. Most times, I find that you learn a lot from your peers. However, do not share/copy/duplicate code from others. If you use code found online, remember to cite their source in your report. Issues related to academic integrity and plagiarism have **ZERO** tolerance.