CS 447 : Networks and Data Communications

Programming Assignment #03

Total Points: 150

Assigned Date : Tuesday, November 13, 2018
Due Date : Tuesday, December 04, 2018 @ 12:29:59 p.m. **(firm deadline)**

Overview

For your third (and final) project, you will improve your streaming application by adding basic user authentication and secure communication capabilities to it. Per RTSP RFC#7829 Sec.19.1, RTSP and HTTP share common authentication schemes. For this assignment, you will follow the "Basic" HTTP authentication (RFC #7617). TCP secure communication is achieved through TLS (RFC #8446), while UDP secure communication is achieved through DTLS (RFC #6347).

Back Story

Haddock is receiving bizzare data from his deep sea probe. *"Billions of bilious blue blistering barnacles. What on ten thousand thundering typhoons is going on here?"* Haddock cried out furiously. *"I bet that evil Rastapopoulos is behind all this"*. Sure enough, Calculus notices that their vanilla RTSP communication is not secure and Rastapopoulos is executing *man-in-the-middle* attacks on it. Moreover, he also discovers that any unauthenticated user can login to the probe and establish their own streams without Haddock's approval. To improve the security, Calculus plans to provide authentication capabilities and secure the TCP control connection using TLS. In addition, Calculus plans to implement server-side logs for incident management and postmortem diagnosis.

Technical Requirements

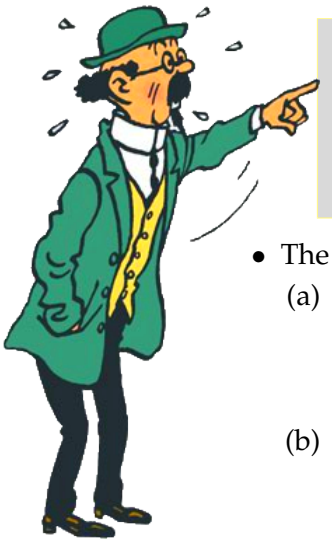
All technical requirements from PR02 are still applicable to PR03. Additional technical requirements that you must meet are as follows:

1. **TLS programming:** Establish a secure channel between your `controller` ↔ `server` using **TLS**. To keep the project within the 2 week time spec, you are not required to implement DTLS. See **Logistics** section below for additional information.
2. **User Authentication:** Before initiating data streaming, authenticate the user to the probe by implementing `WWW-Authenticate` and `Authorization` headers (see RFC #7617 sec. 2).
    - You are only required to implement `Basic` authentication. You may implement `Digest` authentication for extra credit with proper documentation if you desire (but not required).

- The `server` responds to the initial `SETUP` command with a `401 Unauthorized` message. For this project, use **`CS447F18`** as the "`realm`".

```
S → C: RTSP/2.0 401 Unauthorized
       CSeq:  302
       Date:  Tue, 23 Oct 2018 12:29:29 -0500
       WWW-Authenticate:  Basic realm="CS447F18"
```

- The `controller` responds to this 401 with a username and a password separated by a ":" encoded in Base64. Here's an example for username=`haddock`, password=`pirates!`.



```
C → S: SETUP rtsp://<server-hostname>/ RTSP/2.0
       CSeq:  303
       Transport:  UDP;unicast;dest_addr":4588"
       Sensor:  *
       Authorization:  Basic aGFkZG9jazpwaXJhdGVzIQ==
```

- The server behavior upon receiving the `Authorization` header is two-fold.
  (a) For first time users, the `server` will extract the password from the `Authorization` header, salt it by prefixing `CS447` to it, encodes it in base64, and store the encoded password along with the corresponding username in a hidden password file named "`.user_pass`". Hence, first time users are immediately successfully authenticated.
  (b) For registered users, the `server` will extract the received password, salt it with prefix `CS447`, encode it in base64 and check against the stored password for the user in `.user_pass`. Authentication is successful if the two values match.
- A successful authentication is marked with a `200 OK` reply. At this point, the protocol proceeds as defined in PR02.
- A failed authentication attempt is marked with a `403 Forbidden` reply. This response is similar to the 401 response above, thus allowing the user to re-attempt `one more time` (pay attention to `CSeq` numbers).
- Upon two successive failed authentication attempts, the `server` will forcefully terminate the connection with a simple `410 Gone` response. The client will have to re-initiate a new connection at this point.

```
S → C: RTSP/2.0 410 Gone
```

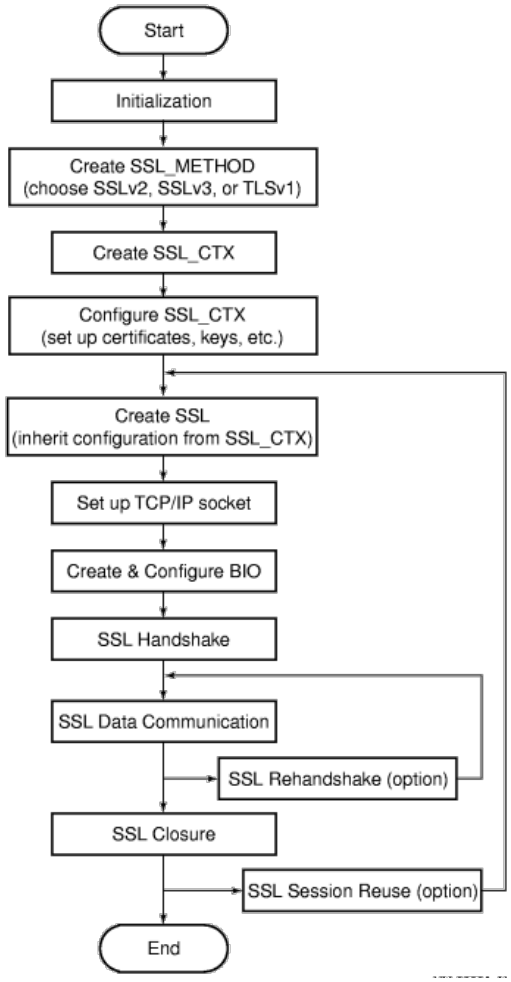
3. **Server Incident Management:** The server keeps an active log file (named `.server_log`) of the connection activities. Every time the server sends or responds to a `control message`, a log entry is added to this log file. More specifically, each log entry (single line) has the following format:

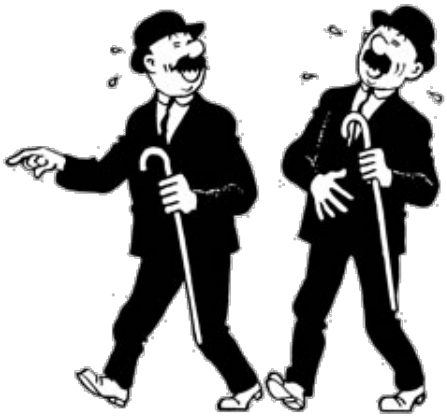
```
timestamp from-ip to-ip protocol-command message-code brief-description
```

4. **Multi-threading:** Support more than one concurrent independent streaming clients.

Logistics

- All applicable PR02 logistics requirements applies to PR03. Additional requirements are listed herewith.
- You must provide proof of secure communication. To do this, run `Wireshark` with and without TLS separately, and provide appropriately annotated screenshots in your report. Failure to provide sufficient proof of secure communication will be considered an indication of NOT meeting project requirements, and is subject to severe penalties.
- The general workflow of TLS integrated socket communication (formerly known as SSL) is as follows[‡]. At minimum, implement TLS 1.2 (or better).





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 the opportunity to improve your skills-set. Here are few resources that I believe will help you.

[‡]Figure from the HP SSL Programming Tutorial.

1. **HP SSL Programming Tutorial:**
   http://h41379.www4.hpe.com/doc/83final/ba554_90007/ch04s03.html
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. **OWASP – Using the Java Secure Socket Extensions:**
   https://www.owasp.org/index.php/Using_the_Java_Secure_Socket_Extensions
5. http://simplestcodings.blogspot.com/2010/08/secure-server-client-using-openssl-in-c.html
6. http://stilius.net/java/java_ssl.php
7. 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 purposes. 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.

Instructions

- This is an individual assignment. **Do your own work**.
- **Start early!! Take backups of your code often!!**. Use of a version control software is highly recommended!
- Make sure to test your program properly before your final submission. It is **highly** recommended to test build and run your submission on the home server, **home.cs.siue.edu**.
- You may use any programming language of your choice out of `C`, `C++`, `Java`, `Python` or `Go` as these are all available on the home server. However, you **must** make sure that your code compiles and runs on a typical Linux machine. Absolutely **DO NOT** include executables with your submissions.
- A **Makefile** is mandatory. Whether or not your program needs to be compiled, have it echo instructions to run the program.
- Follow a good coding standard. Use the Google C++ coding standard found here http://goo.gl/1rC1o, if you don't already follow one.
- The report part of your solution must be produced using a word processor. LaTeX is highly recommended but not a requirement.
- Your final report should be in **PDF** format. No exceptions.
- Any figures, graphs, plots, etc., should also be produced using appropriate computer applications. If using LaTeX, the pgfplots package is very useful for making all sorts of graphs.
- You are strongly encouraged to answer your own design and/or implementation questions based on the the RFCs, rather than defaulting to the instructor immediately. Make sure to properly document any such choices in your report and be as much descriptive as possible with proper sectional citation(s) from the relevant RFCs (*that way I will know why you did what you did*).
- The due date of this assignment is **Tuesday, December 04, 2018 @ 12:29:59 p.m.** A dropbox will

be opened for submission on Moodle. PR03 has a **firm deadline** and there will be no late penalty period.

Deliverables

A complete solution comprises of:

- A short report of the design and implementation of your system. The report should be **PDF** format. At minimum, your report should include the following sections:
  - Introduction: Your objective and what you hope to gain from assignment.
  - Overall design, specific design choices, and reply codes used.
  - The output of a sample run (including screenshots where applicable). See Logistics requirement above.
  - Summary and Issues encountered. What you were able to achieve from your own objectives (from the introduction) as well as project specifications. Make sure to explicitly list functionality you failed to implement (or buggy).
- A compressed tarball that contains:
  - a directory containing (only) your source code. **Do not** include executables, folders created by your programs, or any other files not specifically listed here as required.
  - A short `readme` file with compilation instructions.
  - A `makefile` to automate compilation. A (except python).

To create a compressed tarball of the directory source, use the following command:
`tar -zcvf siue-id-pr3.tar.gz source/`. e.g. `tar -zcvf tgamage-pr3.tar.gz PR03/`

Caution: File formatting standards (`PDF`, `readme`, `make`, and `tar.gz`) as well file naming requirements listed under technical requirements above 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 each others 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 including online sources. The exercise is meant for you to learn network programming, not to test your googling abilities. Issues related to academic integrity and plagiarism have **ZERO** tolerance.

Extra Credit

- Implement DTLS and secure your `data` connection [**20 points**]
- Support for `Digest` authentication with proof [**5 points**]
- Use a Secure Hash Function instead of Base64 when storing passwords in `.user_pass` file [**10 points**]
- Submissions that meet (or exceed) at least 80% of project objectives are eligible for up to 15% extra credit for `ncurses` text-based UI integration `https://www.gnu.org/software/ncurses/`.

Note: If you take any of the extra credit options, you must explicitly list that in your documentation including your report and readme.

Useful Resources

- Datagram Transport Layer Security Version 1.2 RFC #6347 https://tools.ietf.org/html/rfc6347
- The Transport Layer Security (TLS) Protocol Version 1.3 RFC #8446 https://tools.ietf.org/html/rfc8446
- The 'Basic' HTTP Authentication Scheme RFC #7617 https://tools.ietf.org/html/rfc7617
- Linux Man pages – found in all linux distributions
- Beej's Guide to Network Programming – A pretty thorough free online tutorial on basic network programming http://beej.us/guide/bgnet/output/print/bgnet_USLetter.pdf
- Simple Mail Transfer Protocol RFC #2821 https://tools.ietf.org/html/rfc2821
- Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content RFC #7231 https://tools.ietf.org/html/rfc7231
- Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing RFC #7230 https://tools.ietf.org/html/rfc7230
- SMTP Service Extension for Authentication RFC #4954 https://tools.ietf.org/html/rfc4954
- Base64 encoding/decoding:
  - C/C++
    - glib – https://developer.gnome.org/glib/stable/glib-Base64-Encoding.html
    - openssl http://fm4dd.com/openssl/manual-crypto/BIO_f_base64.htm
    - GNU coreutils http://www.gnu.org/software/coreutils/coreutils.html
  - Java 8 – https://docs.oracle.com/javase/8/docs/api/java/util/Base64.html
  - Python 2 – https://docs.python.org/2/library/base64.html
  - Python 3 – https://docs.python.org/3.5/library/base64.html
  - Additional Resource – http://rosettacode.org/wiki/Base64_encode_data#C

  If you use any other resources, make sure to cite those in your report. 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.