Network Programming Project 4 - SOCKS 4

NP TA

Deadline: Sunday, 2021/12/26 23:55

1 Introduction

In this project, you are going to implement the **SOCKS 4/4A protocol** in the application layer of the OSI model.
SOCKS is similar to a proxy (i.e., intermediary-program) that acts as both server and client for the purpose of making requests on behalf of other clients. Because the SOCKS protocol is independent of application protocols, it can be used for many different services: telnet, ftp, WWW, etc.
There are two types of the SOCKS operations, namely CONNECT and BIND. You have to implement both of them in this project. We will use **Boost.Asio** library to accomplish this project.

2 SOCKS 4 Implementation

After the SOCKS server starts listening, if a SOCKS client connects, use **fork()** to tackle with it.
Each child process will do:

1. Receive **SOCKS4_REQUEST** from the SOCKS client
2. Get the destination IP and port from SOCKS4_REQUEST
3. Check the firewall (socks.conf), and send **SOCKS4_REPLY** to the SOCKS client if rejected
4. Check CD value and choose one of the operations
   (a) CONNECT (CD=1)
      i. Connect to the destination
      ii. Send **SOCKS4_REPLY** to the SOCKS client
      iii. Start relaying traffic on both directions
   (b) BIND (CD=2)
      i. Bind and listen a port
      ii. Send **SOCKS4_REPLY** to SOCKS client to tell which port to connect
      iii. (SOCKS client tells destination to connect to SOCKS server)
      iv. Accept connection from destination and send another **SOCKS4_REPLY** to SOCKS client
      v. Start relaying traffic on both directions

If the SOCKS server decides to reject a request from a SOCKS client, the connection will be closed immediately.

```
SOCKS4_REQUEST packet (VN=4, CD=1 or 2)
Type 1: CONNECT
      +----+----+----+----+----+----+----+----+----+....+----+
      | VN | CD | DSTPORT |       DSTIP       |    USERID    |NULL|
      +----+----+----+----+----+----+----+----+----+....+----+
bytes:   1    1      2              4              variable     1

Type 2: BIND (SOCKS 4A)
      +----+----+----+----+----+----+----+----+----+....+----+----+----+....+----+
      | VN | CD | DSTPORT |   DSTIP(0.0.0.x)  |    USERID    |NULL| DOMAIN NAME  |NULL|
      +----+----+----+----+----+----+----+----+----+....+----+----+----+....+----+
bytes:   1    1      2              4              variable     1      variable     1

e.g.
DSTIP=140.113.1.2
DSPPORT=1234 (hint: 1234 = 4*256 + 210 = DSTPORT[0]*256 + DSTPORT[1])
USERID=MOZ
      +----+----+----+----+----+----+----+----+----+----+----+----+
      | 4  | 1  | 4  |210 |140 |113 | 1  | 2  | M  | O  | Z  |    |
      +----+----+----+----+----+----+----+----+----+----+----+----+
bytes:   1 |  1 |    2    |         4         |    variable  |  1


SOCKS4_REPLY packet (VN=0, CD=90(accepted) or 91(rejected or failed))
      +----+----+----+----+----+----+----+----+
      | VN | CD | DSTPORT |       DSTIP       |
      +----+----+----+----+----+----+----+----+
bytes:   1    1      2              4
```

Please refer to these webpages for more detailed SOCKS 4 specification.

- SOCKS 4
- SOCKS 4A

3 Requirements

- Part I: SOCKS 4 Server Connect Operation
  - Turn on and set your SOCKS server, then
    - Be able to connect any webpages on Google Search.
    - Your SOCKS server need to show messages below:
      - Do **NOT** print other contents (e.g., debug messages) in the SOCKS server

```
<S_IP>: source ip
<S_PORT>: source port
<D_IP>: destination ip
<D_PORT>: destination port
<Command>: CONNECT or BIND
<Reply>: Accept or Reject
```

- Part II: SOCKS 4 Server Bind Operation
  - Connect to FTP server with SOCKS
    - Set config with your SOCKS server
    - Connection type is **FTP** (cannot be SFTP)
    - Data connection mode is **Active Mode (PORT)**
      - You can use FlashFXP as the FTP client
  - Download files larger than 1GB completely. E.g., Ubuntu 20.04 ISO image (download link)
    - The SOCKS server's output message should show that BIND operation is used
- Part III: CGI Proxy
  - Modify console.cgi in Project 3 to implement SOCKS client mode
    - Accept SocksIP and SocksPort parameters in QUERY_STRING as **sh** and **sp**, respectively
    - Use SocksIP and SocksPort to connect to your SOCKS server (by CONNECT operation)
    - Rename console.cgi into hw4.cgi in Makefile
  - Use the HTTP server hosted on the NP Server
    - Place files in ~/public_html
    - Visit `http://[NP_server_host]/~[your_user_name]/[your_cgi_name].cgi`
  - Clear browser's proxy setting
    Open your http server, connect to panel_socks.cgi
    Key in IP, port, filename, SocksIP, SocksPort
    Connect to 5 ras/rwg servers through SOCKS server and check the output Test Case (same as Project 3, no hidden test case) t1.txt-t5.txt
- Firewall
  - You only need to implement a simple firewall. List permitted **destination** IPs into socks.conf (deny all traffic by default)

    ```
    e.g.,
    permit c 140.113.*.*          # permit NYCU IP for Connect operation
    permit b *.*.*.*              # permit all IP for Bind operation
    ```

  - Be able to change firewall rules **without** restarting the SOCKS server.
- Specification
  - Port number of SOCKS server is specified by the first argument: ./socks_server [port]
  - The rules in socks.conf default to

    ```
    permit c *.*.*.*
    permit b *.*.*.*
    ```

  - You can only use **C/C++** to implement this project. Except for **Boost**, other third-party libraries are NOT allowed.
  - Every function that touches network operations (e.g., DNS query, connect, accept, send, receive) **MUST** be implemented using the library Boost.Asio.
  - Both synchronous and asynchronous functions can be used. Notice that some situations only work with non-blocking operations. Be thoughtful when using the synchronous ones.

4 About Submission

1. E3
   (a) Create a directory named your student ID, put your files in the **same directory layer**.
   (b) You must provide a **Makefile**, which compiles your source code into two **executables** named **hw4.cgi**(modified from Project 3) and **socks_server**. The executables should be under the same directory as the source codes. We will use these executables for demo.
   (c) Upload **only** your code and Makefile. (e.g., **console.cpp**, **socks_server.cpp**...)
   Do not upload anything else (e.g., **http_server.cpp**, **panel_socks.cgi**...)
   (d) **zip** the directory and upload the .zip file to the E3 platform
   Attention!! we only accept .zip format

```
e.g.,
Create a directory 310551000, the directory structure may be:
310551000
    |-- Makefile
    |-- console.cpp
    |-- socks_server.cpp
    |...
zip the folder 310551000 into 310551000.zip, and upload 310551000.zip onto E3
```

2. Bitbucket:
   (a) Create a **private** repository: ${your_student_ID}_np_project4 inside the **nycu_np_2021** team, under **np_project4**.
   Set the ownership to **nycu_np_2021**

```
e.g., 310551000_np_project4
```

   (b) You need to commit on Bitbucket for **at least 5 times**.
   (c) You can push anything you need onto bitbucket as long as the size of the file is reasonable.

5 Notes

1. **We take plagiarism seriously.** You will get zero points on this project for plagiarism.
2. You will lose points for violating any of the rules mentioned in this spec.
3. Any abuse of NP server will be recorded.