TÉCNICO LISBOA

Redes de Computadores

1st Semester 2020/2021
LEIC Alameda

Programming using the *Sockets* interface

"RC Two-Factor Authentication"

1. Introduction

The goal of this project is to develop a prototype of a file server whose access is protected by a two-factor authentication (2FA) mechanism, complementing the usage of the traditional password with a code sent to a personal device.

The development of the project requires implementing: (i) an *Authentication Server* (*AS*); (ii) a *File Server* (*FS*); (iii) a *Personal Device Application* (*PD*); and (iv) a *User Application* (*User*). The *AS* and *FS* servers, the *PD* and the various *User* application instances are intended to operate on different machines connected to the Internet.
Both the *AS* and *FS* will be running on machines with known IP addresses and ports.
In the following the 'user' (person) is referred to using boldface: **user**. The **user** has simultaneously access to two applications: the *PD* and the *User*.

The **user**, after completing the two-factor authentication procedure, can ask to list, upload, retrieve or delete files to/from the *FS*, as well as to remove all its information. For any single transaction with the *FS*, the **user** must first request a new transaction ID (*TID*) to the *AS*.

The operation is as follows.
Any **user** must first register with the *AS* using the *PD*, by means of a **user** identity (*UID*) and a password (*pass*). The *UID* is a 5 digit number and the *pass* contains 8 alphanumeric (only letters and digits) characters.
The first registration of a **user** with the *AS*, defines the pair *UID*/*pass* at the *AS* database. The registration is approved by the *AS*, here simply by checking an existing list of students' numbers. Subsequent registrations of the same **user** must comply with the predefined *UID*/*pass*. At each registration, the *PD* also sends its IP and a suitable UDP port for the *AS* to be able to contact back the *PD*. The **user** may later remove its registration from the *AS* database.

The possible operations (*Fop*) for the **user** are: requesting a list (L) of all its files stored at the *FS* or to upload(U), retrieve (R) or delete (D) any single file from the *FS*. The **user** may also request to remove (X) its information from the *AS* database, and at the same time to delete all its files from the *FS*.
To obtain the *TID* needed for any single operation, the **user** must have previously registered with the *AS* by using the *PD* application.
For any operation to be possible, the **user** must first login with the *AS*, by using the *User* application, and sending its *UID*/*pass*.
If the login is successful, the *User* requests a transaction ID (*TID*) – a 4-digit number – from the *AS*, sending the **user**'s *UID* and the type of operation to perform (*Fop*). If the

Fop is R, U, or D, the *User* additionally sends to the *AS*, as an extra security measure, the name of the file (*Fname*). The *AS* stores this information in its database and sends to the **user**'s *PD* a validation code (*VC*) – a 4-digit number, as well as information about the type of request made (*Fop*), and the *Fname* if the *Fop* is R, U, or D.
Then the **user** reads this *VC* number at the *PD* and inserts the *VC* at the *User* application to be sent to the *AS* again. The *AS* checks the *VC*, generates the required transaction ID (*TID*), stores this information for posterior validation of *FS* operations, and replies to the *User* with the *TID*. The **user** may be informed of the *TID*, but he will not use it directly at the keyboard. Only the *User* application will use the *TID* at the protocol level. Having received a *TID* the *User* application can complete the desired operation with the *FS*, always identifying the messages with the *UID* and *TID*.
The *FS* validates an operation request with the *AS* by sending it the *UID* and *TID*. As a reply the *AS* sends a message to the *FS* with the *UID* + *TID* + *Fop*, as well as the *Fname*. If, during the validation, the *TID* sent by the *FS* to the *AS* corresponds to a remove operation (*Fop* = X), the *AS* removes the *UID/pass* corresponding to this **user**, and confirms that the *FS* can remove all the files and directories associated with this **user**.
For a list operation, the *FS* will return the list of files already uploaded by the **user** (from this or other instances of the *User* application). In reply to a retrieve operation the *FS* will send the requested file, if possible. The upload operation consists in transmitting the selected file to the *FS*, which confirms the success (or not) of the upload. Each **user** can have a maximum of 15 files stored in the *FS* server. For the delete operation the *FS* will remove the identified file from the server. The remove operation is used to instruct the *FS* to remove all files previously uploaded by this **user**, as well as the corresponding directories, and during validation the *FS* informs the AS to remove this **user**'s account.

The *PD* application performs the following operations:
1 – Register as the personal device of the **user** with ID *UID*, providing a password *pass*.
2 – Receive 4-digit verification codes (*VC*) from the *AS*, which the *User* application instances of this **user** can copy, to be used as a second authentication factor.
3 – Unregister as the personal device of the **user** with ID *UID*, when exiting the *PD* application.

The *User* application can perform the following operations:
1 – Login (first authentication factor).
2 – Request to the *AS* a 4-digit transaction ID *TID* to use for a subsequent operation (list, retrieve, upload, delete or remove) with the *FS*.
3 – Confirm the previous request by sending the verification code (*VC*), obtained from the *PD*, to the *AS* (second authentication factor).
4 – After receiving the *TID*, request the list, retrieve, upload, delete or remove operations to the *FS*.
5 – Display the received replies to the issued requests.
6 – Exit (terminating this instance of the *User* application).

The project tests will include the *AS* and *FS* servers, several *PDs* (one per **user** at a given time) and several instances of the *User* application per **user**.
For the implementation, the application layer protocols operate according to the client-server paradigm, using the transport layer services made available by the socket interface. The interactions between the *AS* and the *PD*, and between the *AS* and the *FS* use the UDP protocol. All other interactions use the TCP protocol.

2. Project Specification

2.1 Personal Device Application (*PD*)

The program implementing the personal device (*PD*) application of the **user** should be invoked using the command:

`./pd PDIP [-d PDport] [-n ASIP] [-p ASport]`,

where:

PDIP this is the IP address of this machine, where the personal device (*PD*) runs, specified in the dot-decimal notation.

PDport this is the well-known port where the *PD* runs an UDP server to accept future *AS* messages with verification codes, as part of the two-factor authentication mechanism. This is an optional argument. If omitted, it assumes the value 57000+GN, where GN is the group number.

ASIP this is the IP address of the machine where the authentication server (*AS*) runs. This is an optional argument. If this argument is omitted, the *AS* should be running on the same machine.

ASport this is the well-known UDP port where the *AS* server accepts requests. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the group number.

Once the *PD* program is running, it waits for a registration command from the **user**. Then it waits for validation codes (*VC*) sent by the *AS*, which should be displayed. The PD application can also receive a command to exit, unregistering the **user**.

The available commands are:

- *reg UID pass* – following this command the *PD* application sends a registration message to the *AS* server, using the UDP protocol, sending the **user**'s identification *UID* (the 5-digit IST student number) and the selected password (*pass*), consisting of 8 alphanumerical characters, restricted to letters and numbers. It also sends the IP *PDIP* and port *PDport* of the *PD*'s UDP server, so that the *AS* can later send verification codes to the *PD*.
  The *UID* is stored in memory for the session duration. The result of the *AS* registration should be displayed to the **user**.

- *exit* – the *PD* application terminates, after unregistering with the *AS*.

2.2 User Application (*User*)

The program implementing the *User* application should be invoked using the command:

`./user [-n ASIP] [-p ASport] [-m FSIP] [-q FSport]`,

where:

ASIP this is the IP address of the machine where the authentication server (*AS*) runs. This is an optional argument. If this argument is omitted, the *AS* should be running on the same machine.

ASport this is the well-known TCP port where the *AS* server accepts requests. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the group number.

FSIP this is the IP address of the machine where the file server (*FS*) runs. This is an optional argument which, if omitted means the *FS* is running on the same machine.

FSport this is the well-known TCP port where the *FS* server accepts requests. This is an optional argument. If omitted, it assumes the value 59000+GN, where GN is the group number.

Once the *User* application program is running, it establishes a TCP session with the *AS*, which remains open, and then waits for the **user** to indicate the action to take using one of the following commands:

- *login UID pass* – after this command the *User* application sends to the *AS* the **user**'s ID *UID* (5-digit IST student number) and a password *pass* (8 alphanumerical characters, restricted to letters and numbers), for validation by the *AS*. The result of the *AS* validation should be displayed to the **user**. The TCP session remains open and the *UID* + *pass* are locally stored in memory for the session duration.

- *req Fop [Fname]* – following this command the *User* sends a message to the *AS* requesting a transaction ID code (*TID*). This request message includes the *UID* and the type of file operation desired (*Fop*), either list (L), retrieve (R), upload (U), delete (D) or remove (X), and if appropriate (when *Fop* is R, U or D) also sends the *Fname*. The **user** should then check the *PD* and wait for a validation code (*VC*) to arrive.

- *val VC* – after checking the *VC* on the *PD* the **user** issues this command, sending a message to the *AS* with the *VC*. In reply the AS should confirm (or not) the success of the two-factor authentication, which should be displayed. The *AS* also sends the transaction ID *TID*. Now the **user** can perform the desired file operation with the *FS*.

- *list* or *l*– following this command the *User* application establishes a TCP session with the *FS* server, asking for the list of files this **user** has previously uploaded to the server. The message includes the *UID*, the *TID* and the type of file operation desired (*Fop*). The reply should be displayed as a numbered list of filenames and the respective sizes.

- *retrieve filename* or *r filename* – following this command the *User* application establishes a TCP session with the *FS* server, to retrieve the selected file *filename*. The message includes the *UID*, the *TID*, the *Fop* and *Fname*. The confirmation of successful transmission (or not) should be displayed.

- *upload filename* or *u filename* – following this command the *User* application establishes a TCP session with the *FS* server, to upload the file *filename*. The message includes the *UID*, the *TID*, the *Fop*, *Fname* and the file size. The confirmation of successful transmission (or not) should be displayed.

- *delete filename* or *d filename* – following this command the *User* application establishes a TCP session with the *FS* server, to delete the file *filename*. The message includes the *UID*, the *TID*, the *Fop* and *Fname*. The confirmation of successful deletion (or not) should be displayed.

- *remove* or *x* – this command is used to request the *FS* to remove all files and directories of this *User*, as well as to request the *FS* to instruct the *AS* to delete this **user**'s login information. The result of the command should be displayed to the **user**. The *User* application then closes all TCP connections and terminates.

- *exit* – the *User* application terminates after closing any open TCP connections.

2.3 *Authentication Server* (*AS*)

The program implementing the Authentication Server (*AS*) should be invoked using the command:

`./AS [-p ASport] [-v]`,

where:

ASport is the well-known port where the *AS* server accepts requests, both in UDP and TCP. This is an optional argument. If omitted, it assumes the value 58000+GN, where GN is the number of the group.

The *AS* makes available two server applications, one in UDP and the other in TCP, both with well-known port *ASport*, to answer requests from the *PD* and the *FS* (in UDP), and the *User* (in TCP) applications.
If the option *v* is set when invoking the program, it operates in *verbose* mode, meaning that the *AS* server outputs to the screen a short description of the received requests and the IP and port originating those requests.
Each received request should start being processed once it is received.

2.4 *File Server* (FS)

The program implementing the File Server (*FS*) should be invoked using the command:

`./FS [-p FSport] [-v]`,

where:

FSport is the well-known TCP port where the *FS* server accepts requests. This is an optional argument and, if omitted, assumes the value 59000+GN, where GN is the number of the group.

The *FS* server accepts TCP requests on the well-known port *FSport*, to answer *User* requests.
If the option *v* is set when invoking the program, it operates in *verbose* mode, meaning that the FS server outputs to the screen a short description of the received requests and the IP and port originating those requests.
Each received request should start being processed once it is received.

3. Communication Protocols Specification

3.1 *PD–AS* Protocol (in UDP)

The interaction between the personal device (*PD*) application and the authentication server (*AS*) is supported on the UDP protocol.
The request and reply protocol messages to consider are:

a) `REG` *`UID pass PDIP PDport`*
Following the *`reg`* command, the *PD* application sends the **user** ID *`UID`* (the 5-digit IST student number) and the password *`pass`* (composed of 8 alphanumerical characters, restricted to letters and numbers) for registration at the *AS* server. It also sends the IP *`PDIP`* and port *`PDport`* of the *PD*'s UDP server, so that the *AS* can later send verification codes to the *PD*.

b) `RRG` *`status`*
In reply to a `REG` request the *AS* server replies indicating the status of the registration request. If the `REG` request was successful (valid **user** ID) the *`status`* is `OK`; if the registration is not accepted the *`status`* is `NOK`.

c) `VLC UID` *`VC Fop [Fname]`*
Following a **user** request for the second factor authentication, to allow an operation, the *AS* sends to the *PD*: the **user** ID `UID` and a validation code *`VC`* (4-digit number), together with the file operation *`Fop`* (either `L`, `R`, `U`, `D`, or `X`), as well as the file name *`Fname`* if appropriate (`R`, `U` or `D`). This information should be displayed by the *PD* application.

d) `RVC` *`status`*
In reply to a `VLC` request the *PD* acknowledges receiving the *`VC`*, replying with *`status`* is `OK` (valid **user** ID); otherwise the *`status`* is `NOK`.

e) `UNR` *`UID pass`*
Following the *`exit`* command, the *PD* application asks the *AS* to unregister this **user**. It sends the *`UID`* and *`pass`* for the *AS* server to check and then unregister the **user**.

f) `RUN` *`status`*
In reply to a `UNR` request the *AS* server replies indicating the status of the unregister request. If the `UNR` request was successful (valid **user** ID) the *`status`* is `OK`; otherwise the *`status`* is `NOK`.

If an unexpected protocol message is received, the reply is `ERR`.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "`\n`".

3.2 *User–AS* Protocol (in TCP)

The interaction between the *User* application and the *AS* server uses the TCP protocol. A TCP connection with the *AS* server is established when the *User* application starts, remaining open until the **user** issues the *exit* or *remove* commands.

The following request and reply protocol messages are considered:

a) `LOG` *`UID pass`*
Following the `login` command, the *User* application sends the *AS* server a message with the **user**'s ID *`UID`* and password *`pass`* for validation.

b) `RLO` *`status`*
In reply to a `LOG` request the *AS* server replies with the status of the login request. If the *`UID`* and *`pass`* are valid the *`status`* is `OK`; if the *`UID`* exists but the *`pass`* is incorrect the *`status`* is `NOK`; otherwise the *`status`* is `ERR`.

c) `REQ` *`UID RID Fop [Fname]`*
Following the *`req`* command, the *User* application sends a request to the *AS* to inform of the **user**'s desire to perform the operation *`Fop`* (either `L`, `R`, `U`, `D` or `X`) on the *FS* server. If the operation is retrieve (`R`), upload (`U`) or delete (`D`) also the file name *`Fname`* is sent. This message initiates the **user**'s second factor authentication procedure. A random natural number of 4 digits is added as a request identifier *`RID`*. Upon receipt of this message, the *AS* will send the `VLC` message to the *PD*.

d) `RRQ` *`status`*
The *AS* server replies informing if the `REQ` request could be processed (valid *`UID`*), a message was sent to the *PD* and a successful *`RVC`* confirmation received. In case of success the *`status`* is `OK`; if the `REQ` request was sent in a TCP connection where a successful login was not previously done the *`status`* is `ELOG`; if a message could not be sent by the *AS* to the *PD* the *`status`* is `EPD`; if the *`UID`* is incorrect the *`status`* is `EUSER`; if the *`Fop`* is invalid the *`status`* is `EFOP`; otherwise (e.g. incorrectly formatted `REQ` message) the *`status`* is `ERR`.

e) `AUT` *`UID RID VC`*
After the **user** checking the *`VC`* on the *PD*, the *User* application sends this message to the *AS* with the *`UID`* and the *`VC`*, along with the request identifier *`RID`*, to complete the second factor authentication. A recently generated *`VC`* will be accepted by the *AS* only once.

f) `RAU` *`TID`*
The *AS* confirms (or not) the success of the two-factor authentication, sending the transaction identifier *`TID`* to use in the file operation with the *FS*. The *`TID`* takes value *`0`* if the authentication failed.

If an unexpected protocol message is received, the reply will be `ERR`.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "`\n`".

3.3 *AS–FS* Protocol (in UDP)

The interaction between the *AS* and *FS* servers uses the UDP protocol.
The following request and reply protocol messages are considered:

a) `VLD UID TID`
Following the *list*, *retrieve*, *upload* , *delete* or *remove* commands issued by the **user** and the subsequent messages (`LST`, `RTV`, `UPL`, `DEL` or `REM`) sent by the *User* application to the *FS*, the *FS* application validates the operation with the *AS* by sending it the *UID* and *TID*.
If the operation to validate is a remove (X), the *AS* delete this **user**'s login information before confirming that the *FS* can remove all the **user**'s files.

b) `CNF UID TID Fop [Fname]`
As a reply to the `VLD` message, the *AS* sends a `CNF` message to the *FS* with the *UID* + *TID* + the requested file operation *Fop* (associated to the *TID* the *AS* had previously authorized), as well as the name (*Fname*) if it is a retrieve (R), upload (U) or delete (D) operation. If the `VLD` request had a problem (*TID* not valid for the *UID*) an error is indicated by replying with *Fop* = E.

If an unexpected protocol message is received, the reply will be `ERR`.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "`\n`".

3.4 *User–FS* Protocol (in TCP)

The interaction between the *User* application and the *FS* server uses the TCP protocol.
The following request and reply protocol messages are considered:

a) `LST UID TID`
Following the *list* command, the *User* application opens a TCP connection with the *FS* server and sends a request asking for the list of files this **user** has previously uploaded to the server. The **user** ID (*UID*) and transaction ID (*TID*) are provided. Before replying, the *FS* sends a message to the *AS* to validate the transaction (`VLD`).

b) `RLS N[ Fname Fsize]*`
After receiving a message from the *AS* validating the transaction (`CNF`), the *FS* reply to a *User* application `LST` request contains the number *N* of available files, and for each file:

- the filename *Fname*, limited to a total of 24 alphanumerical characters (plus '-', '_' and '.'), including the separating dot and the 3-letter extension: "*nnn…nnnn.xxx*";
- the file size *Fsize*, in bytes.

The filenames should be displayed by the *User* application as a numbered list.
If the `LST` request cannot be answered (e.g., no such *UID* or *TID* are available) the reply will be "`RLS EOF`". If the `LST` request is not correctly formulated the reply is "`RLS ERR`". In case of an *AS* validation error the reply is "`RLS INV`".

After receiving the complete reply message, the *User* application closes the TCP connection with the *FS*.

c) RTV *UID TID Fname*

Following the *retrieve* command, the *User* application opens a TCP connection with the *FS* server to retrieve the contents of the file with name *Fname* from the *FS* server. The **user** ID (*UID*) and transaction ID (*TID*) are also provided. Before replying, the *FS* sends a message to the *AS* to validate the transaction (VLD).

d) RRT *status [Fsize data]*

After receiving a message from the *AS* validating the transaction (CNF), and in reply to a RTV request, the *FS* server transfers to the *User* application the contents (*data*) of the selected file, as well as the file size *Fsize* in bytes. If the RTV request was successful the *status* is OK, if the file doesn't exist the *status* is NOK. In case of an *AS* validation error the *status* is INV.

The name and path where the file is stored are displayed by the *User* application.

After receiving the reply message, the *User* application closes the TCP connection with the *FS*.

e) UPL *UID TID Fname Fsize data*

Following the *upload* command, the *User* application opens a TCP connection with the *FS* server and uploads to it the contents of the selected file (*data*), with name *Fname* and size *Fsize* bytes. The **user** ID (*UID*) and transaction ID (*TID*) are also provided. Before replying, the *FS* sends a message to the *AS* to validate the transaction (VLD).

f) RUP *status*

After receiving a message from the *AS* (CNF) validating the transaction, the answer to a UPL request consists in the *FS* server replying with the status of the file transfer. If the UPL request was successful the *status* is OK, if the file already existed the *status* is DUP, if 15 files were previously uploaded by this *User* the *status* is FULL, in case of an *AS* validation error the *status* is INV, otherwise the *status* is NOK.

The *upload* success (or not) is displayed by the *User* application.

After receiving the reply message, the *User* application closes the TCP connection with the *FS*.

g) DEL *UID TID Fname*

Following the *delete* command, the *User* application opens a TCP connection with the *FS* server and requests the deletion of the file with name *Fname*. The **user** ID (*UID*) and transaction ID (*TID*) are also provided. Before replying, the *FS* sends a message to the *AS* to validate the transaction (VLD).

h) RDL *status*

After receiving a message from the *AS* (CNF) validating the transaction, the answer to a DEL request consists in the *FS* server replying with the status of the file deletion. If the DEL request was successful the *status* is OK, otherwise the *status* is NOK.

The *delete* success (or not) is displayed by the *User* application.

After receiving the reply message, the *User* application closes the TCP connection with the *FS*.

i) `REM` *`UID TID`*
Following the *`remove`* command, the *User* application opens a TCP connection with the *FS* server and requests the removal of all its files and directories from the *FS* server, as well as the deletion of the **user** information from the *AS* server. The **user** ID (*`UID`*) and transaction ID (*`TID`*) are also provided. Before replying, the *FS* sends a message to the *AS* to validate the transaction (`VLD`) and requesting the *AS* to remove the **user** information.

j) `RRM` *`status`*
After receiving a message from the *AS* (`CNF`) validating the transaction and confirming the **user** deletion in the *AS*, the *FS* removes all the **user**'s files and directories. It then replies with the status of the operation: `OK` if the `REM` request was successful, and `NOK` otherwise.
The *`remove`* success (or not) is displayed by the *User* application.
After receiving the reply message, the *User* application closes the TCP connection with the *FS*.

All file `Fsize` fields can have at most 10 digits.
If an unexpected protocol message is received, the reply will be `ERR`.
In the above messages the separation between any two items consists of a single space.
Each request or reply message ends with the character "`\n`".

4. Development

4.1 Development and test environment

Make sure your code compiles and executes correctly in the development environment available in `sigma` cluster.

4.2 Programming

The operation of your program, developed in *C* or *C++*, should be based on the following set of system calls:

- Reading user information into the application: `fgets()`;
- Manipulation of strings: `sscanf()`, `sprintf()`;
- UDP client management: `socket()`, `close()`;
- UDP server management: `socket()`, `bind()`, `close()`;
- UDP communication: `sendto()`, `recvfrom()`;
- TCP client management: `socket()`, `connect()`, `close()`;
- TCP server management: `socket()`, `bind()`, `listen()`, `accept()`, `close()`;
- TCP communication: `write()`, `read()`;
- Multiple inputs multiplexing: `select()`.

4.3 Implementation notes

Developed code should be adequately structured and commented.
The `read()` and `write()` system calls may read and write, respectively, a smaller number of bytes than solicited – you need to ensure that your implementation still works correctly.
Both the client and server processes should terminate gracefully at least in the following failure situations:

- wrong protocol messages received from the corresponding peer entity;
- error conditions from the system calls.

5 Bibliography

- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2nd edition, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, chap. 5.
- D. E. Comer, Computer Networks and Internets, 2nd edition, Prentice Hall, Inc, 1999, ISBN 0-13-084222-2, chap. 24.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000
- On-line manual, `man` command
- Code Complete - http://www.cc2e.com/
- http://developerweb.net/viewforum.php?id=70

6 Project Submission

6.1 Code

The project submission should include the source code of the programs implementing the *User*, the *PD*, the *AS* server and the *FS server*, as well as the corresponding *Makefile*. The makefile should compile the code and place the executables in the current directory.

6.2 Auxiliary Files

Together with the project submission you should also include any auxiliary files needed for the project operation together with a *readme.txt* file.

6.3 Submission

The project submission is done by e-mail to the lab teacher, **no later than November 13, 2020, at 23:59 PM**.
You should create a single `zip` archive containing all the source code, makefile and all auxiliary files required for executing the project. The archive should be prepared to be opened to the current directory and compiled with the command `make`.
The name of the archive should follow the format: **`proj_"group number".zip`**

7 Open Issues

You are encouraged to think about how to extend this protocol in order to make it more generic.